Iterative Batched Screening and Active Learning in Drug Discovery

by

Moayad Alnammi

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2021

Date of final oral examination: 1/20/2021

The dissertation is approved by the following members of the Final Oral Committee:
Anthony Gitter, Assistant Professor, Biostatistics & Medical Informatics
AnHai Doan, Professor, Computer Sciences
Xiaojin (Jerry) Zhu, Professor, Computer Sciences
Sebastian Raschka, Assistant Professor, Statistics

Acknowledgments

First and foremost, I thank Allah (God), the Most Gracious, the Most Merciful, for easing my path, and granting me the ability and patience to complete this thesis. I am forever grateful and thankful for the blessings and bounties He has given me. This journey was made easy through His grace and mercy.

I want to especially thank my adviser, Anthony Gitter, for his continued support and guidance throughout my PhD journey. Our weekly meetings were fruitful, engaging, and efficient. These focused weekly exchanges helped spark creativity and independent ideas. The questions he asked helped me see things from different perspectives which was immensely helpful in fleshing out the projects I worked on. His open minded and patient approach to problems allowed for detailed discussions which fostered fruitful advancements. Working with him has given me a good understanding on how to conduct successful research for which I am very grateful.

Special thanks to the Drug Discovery Group of researchers who were monumental to the success of the projects we have collectively collaborated on. Scott Wildman, Spencer Ericksen, and Michael Hoffmann were instrumental in teaching me about drug discovery. Coming from a background in computer science, their help allowed me to navigate and understand the problems in this domain. I am very grateful for their discussions and feedback. I also want to thank co-authors and collaborators from the Small Molecule Screening Facility (SMSF) and James L. Keck's lab for their important contributions to the screening efforts.

I want to thank my colleagues at the Gitter lab for their feedback in our group meetings. I

particularly want to thank Shengchao Liu for our discussions during the projects we worked on. I also want to extend my sincere thanks to the Center for High Throughput Computing (CHTC) for their continued support and efforts. I am thankful for the helpful workflow advice provided by Christina Koch, Lauren Michael, Brian Bockelman, and Josh Karpel. It was through CHTC's compute infrastructure that the work in this thesis was accomplished. I am thankful to King Fahd University of Petroleum and Minerals, Saudi Arabia, for sponsoring my doctoral studies and research.

Finally, I want to thank my mother and father, both of whom have always encouraged my siblings and I towards the pursuit of knowledge. I thank them for the values they have instilled in me. I am thankful to my elder siblings for teaching me the virtue of patience in the face of obstacles, and to my younger siblings for reminding me of the value of perseverance.

Contents

List of Tables

List of Figures

Abstract

Finding safe and effective drugs in the form of compounds is a long and arduous campaign that can stretch to more than a decade. This is mainly exacerbated by the exponential amount of available compounds. The effectiveness of these compounds can only be confirmed via physical testing or screening. Instead of physically screening all available compounds, virtual screening is the use of computational methods to help in carefully selecting compounds for physical screens. In this thesis we explore three virtual screening options for hit finding via four projects.

When historical screening data is available for a target of interest, one can leverage virtual screening methods that model the compound-target interaction data. Given a compound budget and a prospective pool, a question of interest is the effectiveness of such methods in prioritizing hit selection in a followup screen. In such a setting, which we name one round screening, we showcase in chapters 2 and 3 that such methods can succeed in finding hits that far exceed the expectation at random, even when the prospective pool exceeds a billion compounds. An alternative is iterative batched screening, whereby the compound budget is split across multiple rounds of screening. The focus in this setting is to develop a compound selection strategy with the goal of maximizing hits and the diversity of these hits at the end of the iterations. In chapter 5, we formalize this setting and propose a cluster based selection strategy that incorporates concepts from active and reinforcement learning. Following a sequence of rigorous retrospective experiments, a final strategy is used in an iterative screen on a prospective target that mimics a realistic setting. The results of this prospective iterative screen were satisfying, finding 20.28% of the hits in just 4.25% of the prospective pool.

When no screening data is available for a target of interest, we focus on a particular setting where there exists compound interaction data on related targets. The goal is to leverage the related targets' data to prioritize hits for the target of interest. This problem, dubbed the informer set problem, was introduced by Zhang et al. [2]. In chapter 4, we extend their work by providing solutions that were adapted from related fields like matrix completion and machine learning. As a further extension, we apply three scalable methods to a large dataset with missing values ranging from 9,000 to 74,000 compounds and 128 heterogeneous targets. The results expose the difficulties that can arise with such large datasets, however, the insights gained are valuable for directing future works.

In each of these virtual screening options, we formalize the computational problems and objectives. We showcase the success of proposed solutions in realistic and prospective evaluations. In a one round screen, the objective of maximizing hits and the diversity of these hits in a followup screen is straightforward. However, in iterative batched screening, formalism is needed to give rise to a clear framework for developing strategies. This formalism enables us to compare previous iterative screening studies which motivated the development of our proposed novel strategy. Finally, with the continuous growth of online screening data repositories, there is interest in judging the effectiveness of informer set methods on large and incomplete datasets from diverse targets.

— 1 —

Background

1.1 Introduction

Problem Overview

Virtual screening is one of the early processes in the long pipeline of drug discovery. In an overview paper, Mohs and Greig [6], identified as many as 22 processes in the pipeline as shown in Figure 1.1. A simplistic summary of this pipeline is as follows:

1. Identify a **target** of interest that is responsible for some biological effect. For example, HIV Integrase is a protein that facilitates the insertion of the HIV virus into DNA cells. The goal is to develop effective drugs to **inhibit** the function of HIV Integrase.

2. After identifying the target, the next step is to identify small molecules (also called **compounds**, **chemicals**, or **ligands**) that induce the desired effect on the target. Compounds are screened via **in vitro screening** in individual plate wells. This process can be costly and time-consuming as it involves many considerations, most importantly are: selecting compounds to test subject to budget and availability, and determining a suitable **bioassay** which follows an established protocol. The results are in the form of raw signals generally called the **response**. These responses serve as an indicator of the effectiveness of the compound in the assay.

3. Compounds that are highly effective are called **hits** or **actives**, and possibly even **leads** if they will proceed for further preclinical development. Depending on the protocol, a chemist might choose to re-test active compounds to confirm effectiveness. Ultimately, the active compounds with the best potential for development as drugs will proceed as **lead** compounds. Leads are subjected to chemical modifications in an attempt to improve therapeutic potential (called **lead optimization**). The optimization process generally involves identifying and modifying the structural components that determine in vitro effectiveness (**potency**), and from that, tuning physicochemical characteristics to improve in vivo efficacy and safety. Several parameters account for how a lead will move through the body (pharmacokinetics) and elicit a therapeutic/toxic response (pharmacodynamics). These parameters are encapsulated in the abbreviation **ADMET**: absorption, distribution, metabolism, excretion, and toxicity [14, 15, 16, 17] and are assessed by in vitro assays.

4. Optimized leads are then promoted to **in vivo screening**, that is, testing in living organisms. This corresponds to the end of the preclinical development and clinical trials phase in Figure 1.1.

The entire pipeline, from hit-finding all the way to drug release can take around 13 years [18]. After step 2, we have an initial pool of identified hit compounds. These compounds are then further scrutinized in subsequent processes. The attrition rate rises heavily as we go down the pipeline, ultimately, ending with approximately 4.12% of the initial pool of hits [18]. Each false-positive we promote to the next process produces a large unnecessary investment of resources. Thus, there is incentive to optimize this pipeline where ever possible. The more potent and diverse hits we find early on, the greater the likelihood it is that the optimization of these hits will lead to safe and effective drugs for humans.

Let us now focus on step 2: identifying compounds that are effective against the target of interest. For now, we will ignore the costs, availability, and scheduling worries. This simplifies the goal of this step as: select compounds from a pool that will exhibit a significant effect on the target;

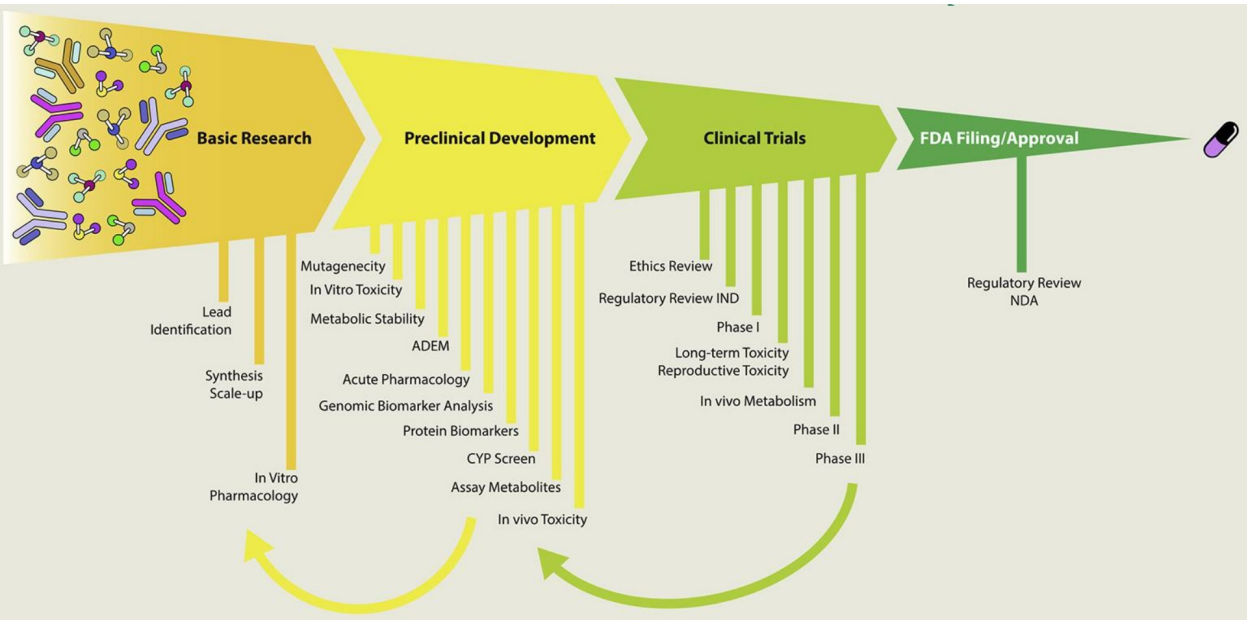


Figure 1.1: Steps involved in the drug discovery pipeline. Starting from an initial pool of many candidate drugs, the processes sequentially prune this pool, ultimately resulting in few viable drugs. This figure is from Mohs et al. [6] under the CC BY-NC-ND 4.0 license (`http://creativecommons.org/licenses/by-nc-nd/4.0/`).

i.e. compounds that are hits. The above simple goal begs a number of clarifications: the amount of compounds to select, the pool, the level of effect, etc. These will be addressed in later sections, but for now, this simplification makes it clear where **virtual screening** comes into play.

Suppose we want to select 10 compounds from a pool of 1,000 compounds. Selection can be aided by chemical computational tools that will **predict a score of the effectiveness** of the compounds in the pool. These scores can then be ranked, and the top 10 ranking compounds are selected. Alternatively, a chemist can use a combination of domain knowledge and computational tools to make the selection. We identify the process of in vitro testing of these selected compounds within wells as **in vitro screening**. The use of computational tools for compound selection is called **in silico screening** or **virtual screening**. With this, we state a general definition:

Definition 1.1: General Definition of Virtual Screening (VS)

Virtual screening is the use of computational methods to help in the selection of compounds for in vitro screening. These computational methods are run on computers, and so, virtual screening is also referred to as in silico screening. Typically, the goal is to maximize hits and the diversity of these hits in the selection.

In this text, we will define more explicit notions for virtual screening depending on the context. **The main focus of this dissertation is research involving this early stage of hit identification in the drug discovery pipeline.** In particular, we focus on research involving the selection of compounds for in vitro screening. In theory, the size of the selection pool is estimated to be 10^{60} compounds [19]. In 2020, commercial libraries have reached up to hundreds of millions to billions of compounds [20, 21, 22]. In addition to the large space, selected compounds incur variable costs: cost of synthesis or vendor purchase, cost of labor, and time. Just from the four projects in this thesis, the cost per compound can be roughly categorized into three cost brackets: in-house collections maintained locally that go for less than $1 USD, outside vendor collections of in-stock compounds ranging from $10 to $200 USD, and outside vendor collections requiring custom chemistry that are on the order of $1000 USD [23, 24, 22]. Because of this, it is not feasible to screen millions of compounds without a substantial investment [18]. Therefore, screening facilities that operate on small budgets can benefit the most from effective virtual screening methods. In the coming sections, we will discuss virtual screening types, scenarios, and options. This will help establish the problems involved in virtual screening and what steps we can take to solve them.

1.2 Types of Virtual Screening Methods

With our definition of virtual screening in Definition 1.1, we can discuss the main types of virtual screening methods. All method types are computational but require certain information to be applicable. They all apply a model in order to score compounds, allowing for rank-selection. But

before discussing the types, let us explicitly state the information[1] related to the problem:

1. **Target Information:** This is any data related to the target of interest. An example is **structural** information, i.e. the 3D physical representation of the target which is a vital piece of the problem. One can also harness information from a *related* target, that is, make use of information available from *another* target that is related to the *current* target of interest (for some notion of *relatedness*).

2. **Compound Information:** This is any data related to the compounds being tested. **Structural** information, is yet again related to shape, and in the context of compounds they are also called **molecular** information. **Chemical** information are physical chemistry (**physicochemical**) properties like total molecular weight, solubility, volatility, lipophilic efficiency, etc. **Biological activity** information refers to the effectiveness of a compound towards *multiple* targets (not just the target of interest); this multi-target biological activity information can be used to describe a compound.

3. **Compound-target Information:** This is data involving the relationship between the target and compounds. Of particular interest to drug discovery is the **biological** information governing the interaction between target and compound. This is the notion of **effectiveness** we alluded to earlier, and is also referred to as **biological-activity** or bioactivity.

Now we provide a simplified overview of the response in an in vitro screen. The binding mechanism governing the interaction between target and compound is not a fixed state, but can rather be considered as a range of loosely-binding to tightly-binding interactions. In an in vitro screen, the compound is introduced into the well at a single concentration, typically measured in micro or nanomolar (μM or nM, respectively). Depending on the bioassay, a suitable sensor is used to gather a signal over some period of time. The signal varies over this period as the interaction between target and compound varies in the bioassay. This is mainly due to the changing **orientation**

[1] In this text, we use **information** and **data** interchangeably.

and/or **conformation** of both target and compound as they interact in the bioassay[2]. The signal is also correlated with how strongly the compound binds, i.e. its **effectiveness** or **binding-affinity**. An effective compound will produce a stronger signal on average over the course of the screening period. Typically in screening, a plate with many wells is used. Standard plate sizes are 96, 384, and 1536 wells. The wells typically have the same bioassay and target, but different compounds will be introduced into each well. A typical followup is to then normalize the signal readings from different wells, this is referred to as the **normalized % inhibition**, but is usually simplified to just **% inhibition** [25]. This produces a dataset of compound % inhibition readings where % inhibition is a continuous value; a higher value indicates that the compound is more effective at interacting with the target.

However, recall that we had a notion of **hits**, that is, compounds with a strong signal. This begs the question: how strong should the signal be for a hit compound? This is a debated subject, but proposed methods typically involve looking at the statistics of the signals of the tested compounds [26, 27, 28, 29]. For example, one common method is to assume a normal distribution and consider all compounds whose signals are above the mean by two or three standard deviations as hits [27]. This produces a dataset of compound binary values indicating hit (1) and non-hit (0) (also referred to as **active** and **inactive**).

Thus, the result of an in vitro screen produces two compound-target bioactivity datasets: continuous and binary. Computational virtual screening methods attempt to model these datasets directly or indirectly. **Ligand-Based** methods learn from the compound-target bioactivity data directly. Whereas **Structure-Based** methods model the physical interaction between target and compounds to produce scores that are correlated with the continuous bioactivity data.

It is worth mentioning that a complete screening of a compound involves multiple screens at varying concentrations. This produces multiple signals at varying concentrations that are then fit to a curve called the **dose-response curve** [30]. In addition to being more robust than a single concentration reading, the curve can be analyzed to better determine the efficacy of a compound.

[2]In this text, **orientation** refers to *rigid-body* rotation, whereas **conformation** refers to the internal structure of the components (target and ligand).

Generally, a strong hit compound exhibits a strong inhibition signal at low concentration (i.e. sub μM). As one can conclude, this is costly when screening thousands of compounds as each compound would require multiple screens. To reduce the cost, initial primary screens would be done at a single concentration to determine promising, strong signal compounds. These promising compounds are then promoted to a secondary round of screening. Even prior to the dose-response curve, one can conduct a secondary screen of promising compounds under the same conditions of the primary screen, but at multiple replicates to confirm the signal. A secondary screen can also be done under different bioassay protocols measuring the same biological effect or other properties like toxicity.

Structure-Based

Structure-based virtual screening models the physical compound-target interaction via some computational simulation or approximation. Thus, it requires a physical 3D structural representation of the target of interest and the compound. Typically, the target structure is acquired via spectroscopy methods like X-ray crystallography and Nuclear Magnetic Resonance (NMR) spectroscopy [31]. These methods produce experimental data that is combined with prior knowledge to produce an *estimate* of the target structure. Furthermore, it is possible to obtain an **estimate** of the target structure as it is interacting with a *known* active compound in a bioassay. This may be preferred as it will hopefully capture the target structure in a favorable binding conformation. This compound-target structure may also be used as a reference in a proposed structure-based method. The process of estimating the target structure is more complicated than what is stated, but the details are beyond the scope of this text. The main takeaway is that structure-based methods require a 3D structure of the target and compound. Because compounds at this stage are typically small molecules, their structures are typically estimated computationally via molecular dynamic laws.

With a structure of both target and compound, one can design a computational method that simulates the binding mechanism. **Docking** programs are popular structure-based virtual screening methods. A docking program uses target and compound structures as input and produces a binding

score as output. Some docking programs only require target structures and will produce compound structures on the fly (from a pool of designated compounds). These docking programs typically follow three steps (these are, again, simplified overviews):

1. From the 3D target structure, generate a *coarse* 3D surface map based on molecular dynamics force-fields. These force-fields are governed by the atomic interactions of the target. An optimization step is conducted to find a **binding pocket or site** that will accommodate the compound's 3D surface; analogous to lock-and-key concept where the target is the lock and the compound is the key. The target structure is held fixed, whereas the compound is allowed to change its conformation to better fit a binding site. The amount of binding sites and conformations tried are typically set as input options by the user.

2. Once a suitable site is found, the program then computes a score that is sometimes taken as an *estimate* of the **binding free energy**. An effective compound will have low binding energy. To be efficient, some programs will estimate this energy by focusing on a subsection of the compound-target interaction in 3D space.

3. Steps 1 and 2 are repeated for every compound in the library. The resulting binding energies are then used to rank the compounds.

With regards to a fixed library, the scores produced by a docking program do not have to coincide approximately with actual binding signals produced by in vitro screens. But rather, the docking scores desirably correlate such that the rankings produced by the docking program are similar to the rankings by in vitro screens. This text will make mention of structure-based methods when appropriate, but it is not the main focus.

Ligand-Based

As previously mentioned, ligand is another name for compounds or molecules[3]. From the namesake, ligand-based methods learn the compound-target bioactivity relationship using *only* compound in-

[3]Ligands typically refer to small molecules that induce an effect on a target.

formation; no target information is incorporated directly. The method work-flow usually follows some **featurization**[4] of the compounds that is used to predict the bioactivity. Once the compounds are featurized, a computational method can then build a model *relating* the features with the bioactivity data.

In the statistical and machine learning domain, the process of *learning* the relationship between features and response/label data is known as **supervised learning**. Machine learning has come a long way particularly due to continuing technological advancements in GPU architectures and the ability to gather large amounts of data. Optimization in GPU architectures allow for building large complex models that can be trained faster. With more and more data, the models are able to generalize better and become less prone to overfitting. Once a supervised learning method has learned the relationship from the given data, the method can take as input a new compound and output the *predicted* bioactivity. **Ligand-based methods and their usage will be the main focus of this text.**

Another worthy mention is that if the featurization is based on the structure of the compound, then this process of relating structural features to bioactivity is known as **Quantitative Structure-Activity Relationship** (QSAR) in the drug discovery domain. Common featurization and machine learning models are discussed in section 1.5.

Interchangeability of Method Types

We identified two main virtual screening methodologies based on the information used: ligand-based virtual screening uses compound features and compound-target bioactivity data, whereas structure-based virtual screening uses target structure and compound structure. Although not covered in this text, it is worth mentioning that there are methods that use all available information: target structure, compound structure and features, and compound-target bioactivity data. These methods make use of some representation/featurization of the target and compound to *learn*, i.e. *predict* the compound-bioactivity relationship. Thus, it can be considered as a supervised learning

[4] A featurization is a representation, usually in the form of numerical vectors, of a compound. This representation is then used by a statistical method to build a model/relationship between the representation and the bioactivity data.

approach. Such methods are not strictly ligand-based nor structure-based, and so, can be considered a hybrid. As such, the literature varies in the type assignment of proposed methods.

Recall that the structure-based virtual screening methods we identified above do not *learn* the bioactivity relationship from the data directly, but rather estimate the binding energy by modeling the physical interaction of the components. One can argue that this physical model was developed using experimental data from *various* compound-target bioactivity datasets; i.e. a physical model that agrees with various targets. While this may be the case, we make the distinction between such methods (structure-based) and methods that learn from the data of the *current* target of interest (ligand-based and hybrid).

One example of a hybrid method is that of a docking program that, instead of estimating the binding free energy, learns the relationship between a given compound-target 3D binding pose and the bioactivity response. To be more specific, using spectroscopy, one can extract ligand-target binding poses and bioactivity, generate the 3D structure, then use a supervised learning approach to learn the relationship between binding pose and bioactivity [32]. Given a new compound, the 3D structure is generated and the *learned* model is used to predict bioactivity. This makes use of compound structure, target structure, and compound-target bioactivity data. The main takeaway is that the lines between ligand-based and structure-based can be blurry, but they all seek to induce a ranking of the compounds that reflects bioactivity.

1.3 Virtual Screening Use Cases

In section 1.2 we identified the type of data and methods in virtual screening. In this section, we define the use cases one might face in terms of the data available, and consequently, which virtual screening method type is applicable. Although this summarizes some of what was previously stated, it will help put things into perspective. In particular, we want to map use cases to applicable methods that will select the compounds to screen (from a pool). We first list the main cases in virtual screening:

- **Case 1:** No target structure and no compound-target bioactivity information is available. For this case, one can apply high-throughput screening, diversity sampling, or an informer set method. These methods can be applied to acquire an initial bioactivity dataset for a ligand-based method.

- **Case 2:** Target structure is available, but no compound-target bioactivity. In this case, since target structure is available, we can apply structure-based methods. Furthermore, using target structure one can apply an informer set method.

- **Case 3:** Compound-target bioactivity is available. In this case, ligand-based methods can be applied.

- **Case 4:** Both target structure and compound-target bioactivity information is available. In this case, ligand-based, structure-based or hybrid methods (or some combination of all three) can be applied.

Note that these cases are not all mutually exclusive. For example, if both target structure and compound-target bioactivity is available, one can opt to just ignore target structure and work only with compound-target bioactivity data.

High-throughput Screening and Diversity Sampling - All Cases

In all cases, one can ignore the information available (or lack thereof) and sample from the space of available compounds. One popular option includes bulk-purchases of compound collections from a vendor. These collections are typically a diverse set of compounds that have shown activity towards a wide range of targets. One can even sample randomly from the space of available compounds. Such screens are called high-throughput screening and can range from 500 to greater than 100,000 compounds [33].

Instead of purchasing an entire collection, diversity sampling can be used to select a diverse set of compounds from a pool. This technique is simple and requires only some measure of diversity; a

common measure is dissimilarity between two compounds. Given a pool U of available compounds, this method selects a set S of n compounds from U that are dissimilar. This can be accomplished in a number of ways, the simplest of which is to select the first compound from U randomly, add it to S, and then select each subsequent compound from U such that it is dissimilar to the compounds in S. Another method is to first cluster the compounds in U (based on dissimilarity), and then select a number of compounds from each cluster or randomly selected clusters. One can think of more elaborate ways, but the main concept is to diversify the set of selected compounds to cover as much of the chemical space as possible. Gaining knowledge about different areas of the chemical space can help in focusing later searches in subsequent screens.

Since this method does not use any information to better select compounds that are likely to be hits, it will often result in screening a large number of inactive compounds. This is due to the small percentage of hits for any particular target, typically ranging from 0.01% to 5% in a diverse pool of compounds [4, 34, 2]. However, this is not necessarily a bad thing, particularly if this method is used as an exploratory technique in conjunction with a more exploitative technique; i.e. splitting the budget between exploration and exploitation. By exploitation, we mean using historical compound-target screening data to learn about hit compounds in order to promote new compound hits for future screens. One way this can be done is by training a supervised learning model on the historical data, and subsequently predicting the hit probabilities of new compounds. On the other hand, exploration can be helpful for supervised learning models as they can further learn from inactive samples and adjust their hypothesis accordingly. Diversity sampling is specifically applicable when starting a screening campaign for the first time and there is no data available to apply a data-driven technique (like ligand-based virtual screening). Finally, diversity sampling can be considered high-throughput screening as we will allude to later.

Informer Set - Cases 1 and 2

In addition to diversity sampling, we identify another option in cases 1 and 2: informer set methods. The informer set use case is as follows: we have a target of interest A, a set of related targets

$\{B, C, D\}$, and a pool of available compounds U. There is no information available for A, but there is compound-target bioactivity data for the related targets $\{B, C, D\}$ on U. The goal is to extract a set of n compounds from the compound-target bioactivity data of $\{B, C, D\}$. These n compounds are selected from U such that their activity towards $\{B, C, D\}$ is *predictive* of the remaining non-informer compounds towards each target. These n compounds are then screened in vitro against A, and the screening data is used to predict the activity of the remaining non-informer compounds against A. Such a set of n compounds are said to be *informative* of the remaining non-informers, hence the name **informer set**. The caveat is that this hopefully works if the targets are related (usually by structure or sequence similarity). Using a set of related targets allows integrating information from multiple sources that are highly or even partially related to the target of interest. For Case 1, these n informer compounds can be used to kick-start a ligand-based method. For Case 2, the target-target structure similarity between the target of interest and the related targets can be used to directly score compounds in a nearest target fashion. In section 4, we formalize the informer set problem, explore appropriate methods, and compare their performance.

1.4 Ligand-based Virtual Screening: Cases and Options

In this section we want to further define the cases associated with the focus of this text: ligand-based methods. Recall from sections 1.2 and 1.3 that ligand-based methods are applicable in cases 3 and 4 where compound-target bioactivity data is available from some previous in vitro screening effort. If no data is available, then, as stated in section 1.3, one can perform high-throughput screening, diversity sampling, or informer set methods to select compounds for in vitro screening towards a target of interest. Once this initial set of compounds have been screened, the initial dataset of compound-target bioactivity can be curated. If there is a desire to screen more compounds, we can proceed with one of two options: **One Round Screening (ORS)** or **Iterative Batched Screening (IBS)**. We give a general definition of ORS and IBS in terms of their objectives:

Definition 1.2: One Round Screening (ORS)

Given a dataset T of compound-target bioactivity, a pool of available compounds U, and a budget B in terms of monetary cost or number of compounds. Train a ligand-based method using T, then select a set S_k of the top k most probable hits from U in accordance with the budget B. Perform an in vitro screen on S_k and evaluate the results.

Definition 1.3: Iterative Batched Screening (IBS)

Given a dataset T_0 of compound-target bioactivity, a pool of available compounds U_0, and a budget B.
Develop an iterative screening **compound selection strategy** F as follows:

1. Set $i := 0$, $T_i := T_0$, and $U_i := U_0$.
2. Using the strategy F, select a set $S_k := F(T_i, U_i, B)$ of k compounds from U_i.
3. Perform an in vitro screen on S_k and evaluate the results.
4. Update the training and pool datasets with current iteration's screening results: $T_{i+1} := T_i \cup S_k$, $U_{i+1} := U_i \setminus S_k$, and $i := i + 1$.
5. Repeat steps 2 to 4 as desired.

As can be seen from Figure 1.2, ORS and IBS have similar components, and differ in terms of context and length. An ORS is usually done in two phases: the first phase screens an initial dataset T using high-throughput screening or diversity sampling, and the second phase trains a ligand-based method to select a set of compounds from a pool U. In contrast, an IBS is planned from its inception as a sequence of screens with the purpose of building a rich training dataset T step-by-step. It is difficult to draw a line between what constitutes an ORS or an IBS. Some ORS are planned to be conducted in iterations where the time period between each screen is a year or more. One can argue that this falls within the definition of IBS.

In this text, we make the distinction between ORS and IBS based on the time period between batches of in vitro screens. We characterize IBS as batches of iterative in vitro screens with short (less than six months) periods between each screen. From a data-driven perspective, gaining small,

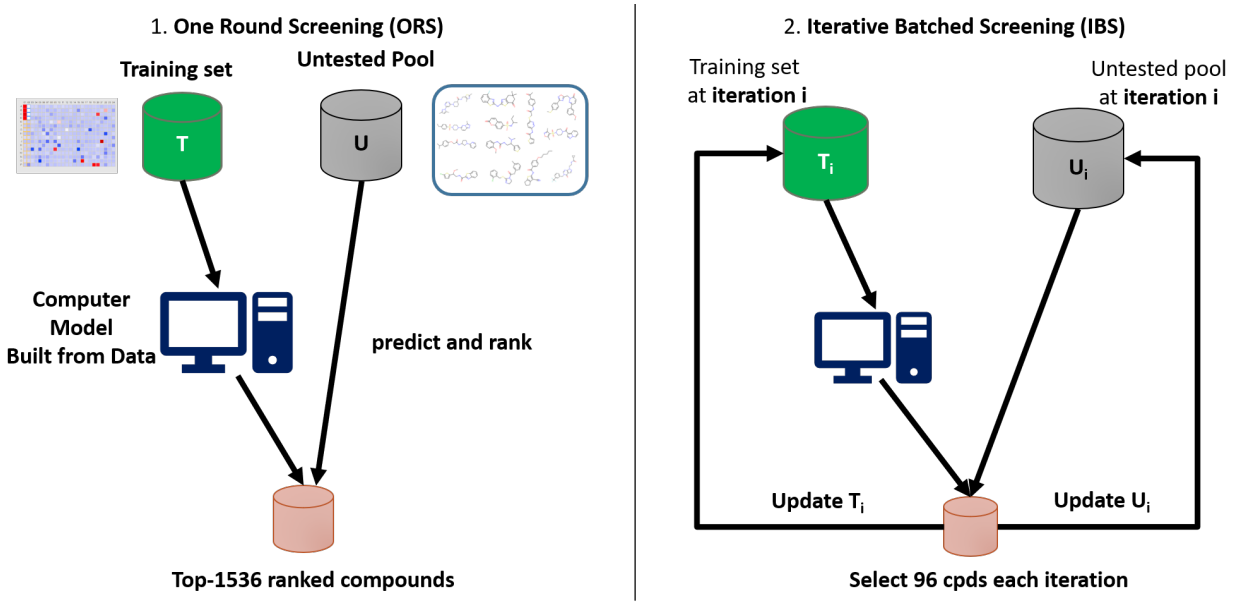


Figure 1.2: Depiction of ORS and IBS side-by-side showcasing the similarities and differences. Starting from an initial training set, both ORS and IBS train a supervised learning model on the training data, then score and rank the prospective pool of compounds. The top k ranked compounds are selected and screened in vitro. The ORS stops here, whereas the IBS proceeds for more iterations, incorporating the newly acquired screening data in its decision-making.

but focused information on the compound-target bioactivity space can help identify more hits overall than a single, large survey of the space. The ligand-based model is able to adapt its hypothesis with each batch of data. Furthermore, compounds are screened in standardized plates of 96, 384, and 1536 wells. The batch sizes typically relate to the plate sizes after accounting for control wells. Of course the batch size can go beyond these single plate sizes by using multiple plates in each iteration.

Finally, there is an important component in an IBS, namely, the compound selection strategy F. To better understand the compound-target bioactivity landscape, over the course of multiple iterations it is imperative to split the screening budget between exploration and exploitation. This is the exploration-exploitation dilemma [35, 36]. In the context of ligand-based methods, **exploration** refers to the screening of compounds for which the method is uncertain of its activity. **Exploitation** refers to the screening of compounds that the method predicts to be highly active. With the general definition of ORS in Definition 1.2, it is fully exploitative; i.e. it selects the most active compounds

according to the ligand-based method. In the IBS Definition 1.3, the strategy F makes use of the ligand-based method (trained on T_i), to score the compounds in U_i. This produces a ranking that can be exploited. However, based on the budget B, perhaps not all the top k compounds have high scores. This indicates that the model is not confident in these *weakly* active compounds, and it may benefit from exploring more of the chemical space. Consequently, one can split the budget between exploiting the highly active compounds, and exploring a diverse collection of *weakly* and *poorly* active compounds. As the ligand-based method acquires more and more data covering diverse areas of the chemical space, it is hoped that an IBS will retrieve more diverse hits in the long run. It is also important to shrewdly manage the exploration budget to avoid spending time and money screening too many inactive compounds. This is the main intuition behind the use of an IBS over ORS.

In chapters 2 and 3 we discuss ORS projects conducted on a protein-protein target of interest. In chapter 5 we propose and evaluate an IBS strategy that features rich customization options to tune exploration versus exploitation.

1.5 Ligand-based Virtual Screening: Featurization and Models

So far we have not discussed any particular ways to featurize/represent a compound and only referred to them by their usage in screening. Ligand-based methods learn the relationship between a compound representation and its bioactivity with respect to a target. Before describing the common featurization schemes that have been applied in ligand-based virtual screening, it is important to understand what we mean by a featurization or a representation. To be concrete, we give the following definition:

Definition 1.4: Featurization of a compound

Given a compound $x \in X$, where X is the *abstract* space of compounds, a **featurization** function G transforms x into a numerical representation. For example, $G : X \to \mathbb{R}^d$ transforms x into a vector $G(x) \in \mathbb{R}^d$. Another example, $G : X \to \mathbb{R}^{m \times n}$ transforms x into an $m \times n$ matrix. More elaborate transformations can be conceived like transforming x into a list of vectors and matrices.

The notion of an abstract space of compounds refers to the physical representation of a compound in reality. Trying to capture this real physical characteristic in a virtual/digitized setting is not feasible, and so, we settle for a discrete representation. With a compound featurization, a ligand-based model can be used to learn the compound-target bioactivity relationship as depicted in Figure 1.3. In this section we will give an overview of how this is done, as well as describe other common featurizations for compounds. For each featurization, we will describe the ligand-based models that can be applied.

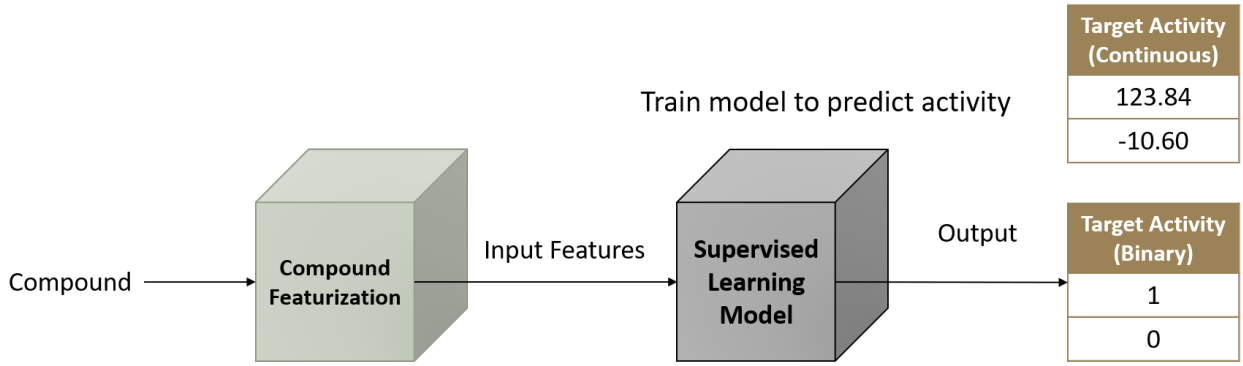


Figure 1.3: The usage of a featurization method to generate compound features. The ligand-based supervised learning model learns the relationship between compound features and bioactivity.

2D Molecular Graph

A popular example of a representation of compounds is the 2D molecular graph as depicted in Figure 1.4. This graph is a standard technique for representing the structure of the compound via atoms and bonds. However, this representation needs to be discretized for consumption by a

computational method. For 2D graphs, we describe two popular methods of achieving this.

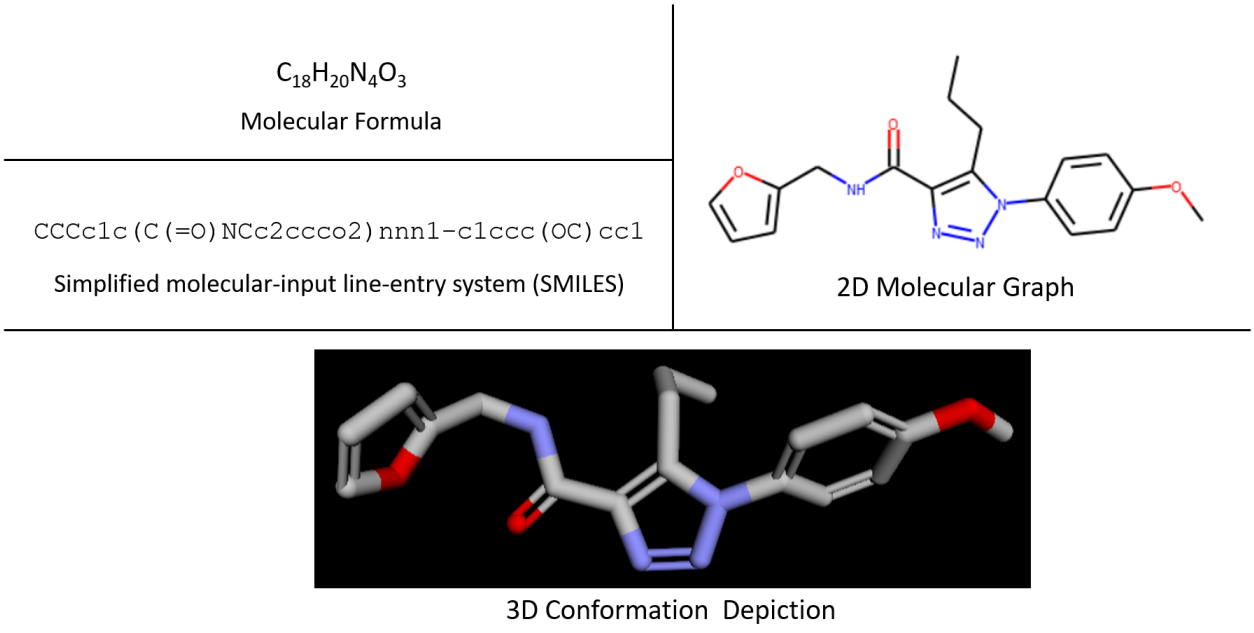


Figure 1.4: Common representations of the same compound: molecular formula, SMILES string, 2D molecular graph, and 3D conformation. These depictions were generated using the RDKit Python package [1].

One way is to represent the 2D graph of a compound as a pixel-image with a fixed resolution; i.e. 250 by 250 pixel-image where each pixel is a number denoting the molecular property at that pixel location. A standard pixel-image has three color channels (red, green, and blue) capturing three pieces of light information. In a computer, for each channel, each pixel contains a number representing the intensity of that type of light (red, green, or blue). In a 2D molecular graph, we want to capture molecular/structural information in the channels. So suppose we have the 2D graph of a molecule, and we set up a rudimentary algorithm for depicting the graph on a 2D discretized plane with a fixed length and width. The next step would be to set up n pixel-image channels of size $l \times w$. Each channel corresponds to a compound property we deem important. For example, an important structural property is the type of atom, and so, one channel will depict the type and locations of the atoms. Numerically, for each atom, we assign a unique numerical identifier (e.g. non-negative integers), and set the pixels where that atom is located to the appropriate atom-identifier. If there are no atoms located at a pixel, we simply assign it a unique no-atom-identifier (e.g. the integer zero).

Typically, each unique number is then transformed into a bit vector of a size equal to the number of unique atoms. This bit vector has exactly one bit that is *on* (i.e. set to 1) at a particular location according to its atom identifier. In other words, if two pixels have the same atom identifier, their *on*-bit is at the same location. This type of bit vector transformation is common for categorical data in machine learning, and is called one-hot encoding. Similarly, other channels can hold information like bonds, valence, atomic weight, or any type of atom/bond information that can be depicted in a 2D image-pixel. This idea was applied by Goh et al. in their deep learning models to predict chemical properties [37, 38]. The particular deep learning model used are called convolutional neural networks (CNN) [39, 40], a popular architecture to learn from images.

Another method of discretizing a 2D graph is by presenting it directly as a graph of nodes (atoms) and edges (bonds). The nodes representing the atoms can be vectors holding atom information like atom-identifier, weight, valence, etc. Similarly, bond information can be held in vectors. The atom-atom bond connection information is represented as a graph adjacency matrix. A deep learning architecture known as graph convolutional networks (GCN) learn from such graph representations [41, 42]. GCNs have been proposed to predict compound-target bioactivity and compound properties [43, 44, 45, 46, 7]. A simplified depiction of the data flow for such a model is shown in Figure 1.5.

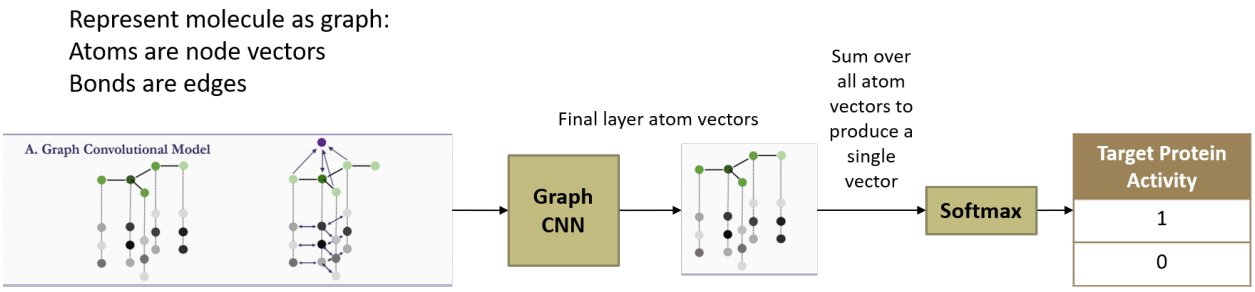


Figure 1.5: Illustration of graph convolutional networks (GCN) that learn the relationship between a compound's graph representation and its bioactivity. The depictions for the graph operations were adapted from Wu et al. [7] under the CC BY-NC 3.0 license (`http://creativecommons.org/licenses/by-nc/3.0/`).

Simplified molecular-input line-entry system (SMILES)

SMILES are single-string ASCII character representation of a compound's structure [47]. SMILES algorithms for generating a string from a 2D structure will start by specifying an atom as a root, and then delineating a spanning tree from that root (rings are broken for the spanning tree formulation). Then starting from the root, it lists the branches in a string format, one branch at a time. The chemical element acronyms (e.g. C for carbon, N for nitrogen, etc.) are used to denote the atoms present in the structure. Numbers are used to denote the broken rings. Other characters are also used to denote bond order, aromaticity, branching, etc. SMILES algorithms tend to be software specific, but the general idea is the same: SMILES strings are generated from the 2D structure and there is a one-to-one correspondence. The one-to-one correspondence property refers to canonical SMILES algorithms. Canonical SMILES can be used to check for uniqueness among compounds in a database; but be sure to use the *same* canonical SMILES algorithm consistently. Figure 1.4 depicts a compound's canonical SMILES string and its corresponding 2D structure using the RDKit Python package [1]. Natural language processing (NLP) models can be used with SMILES to learn compound-target bioactivity relationship. A deep learning architecture known as recurrent neural networks (RNN) can be used to take SMILES strings as input [8]. However, this procedure has not been used extensively in drug discovery, and instead, more focus has been on using generative networks with RNN architectures to output SMILES strings that are similar to known active compounds or compounds with interesting properties [48, 49].

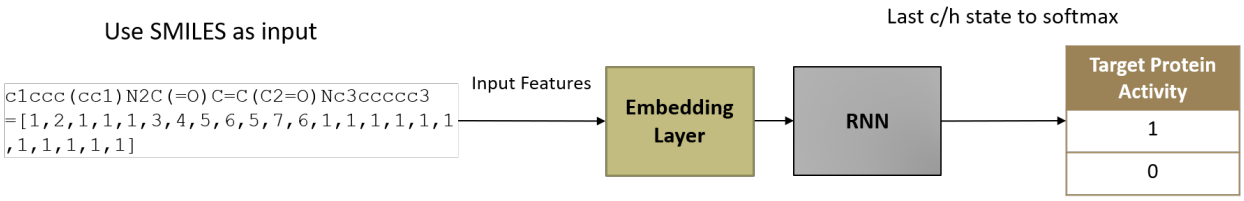


Figure 1.6: Example of an RNN model to learn bioactivity from SMILES. In this example, the RNN outputs its last layer's cell and hidden states (denoted as h and c, respectively) which are passed into a softmax layer to predict the binary activity. More details on RNNs can be found in these references [8, 9].

Molecular Descriptors and Fingerprints

A popular featurization concept for learning bioactivity is that of molecular descriptors. In general, molecular descriptors are numerical identifiers for a property that a compound possesses [50]. Some of these properties can be continuous values like molecular weight, lipophilicity, temperature points, bioactivity, toxicity, etc. Others can be binary values indicating the presence of a substructure like a particular ring structure, combination of atoms, bond type, etc. As long as the molecular property can be represented as a numerical identifier, then it can be used as a descriptor. Thus, a given compound can be represented as a list of its descriptor values.

Fingerprint descriptors are fixed-length bit strings encoding the presence of features in the compound. Fingerprinting algorithms encode the presence of substructures in the bits. They were initially developed to efficiently search large databases for compounds with fingerprints that are similar to a query molecule. This gives a basic similarity ligand-based model that extracts compounds with fingerprints that resemble known active compounds. Methods have been proposed to fuse fingerprints from different fingerprinting algorithms together to improve the similarity measure [51]. It is important to mention that the similarity baseline is based on the Similar property principle that states that similar compounds have similar properties [52]. However, since similarity is dependent on the measurement method used, this can lead to higher (or lower) false-positives when using the similarity baseline with differing similarity measures [53, 54]. Regardless, the similarity baseline provides a good, better than random baseline method for comparison [53].

Circular fingerpint algorithms, a popular one being extended-connectivity fingerpints (ECFP) [55], generate variable-length strings of numeric identifiers for each substructure present in the compound. They usually do the following: generate the 2D structure, then generate a unique numeric identifier for each substructure centered at each atom up to a user-specified radius. Larger molecules will generate longer strings due to having more substructures. Since most machine learning methods require fixed-length vectors, there is a post-processing step that converts the variable-length string to a fixed-length bit string. The unique substructure identifier is hashed to a bit position in the fixed-length bit string, setting that bit position to *on*. An important byproduct

to keep in mind is that this hashing process is non-reversible, that is, a single bit position can correspond to multiple substructures from the same compound or even different compounds. This *collision* becomes worse as the size of the compound database grows (both in terms of number and diversity) relative to the length of the bit string. A simple fingerprinting depiction is shown in Figure 1.7 showcasing the collision between and within two compounds. In ligand-based virtual screening, circular fingerperints have been used to predict bioactivity [34] and toxicity [56] with great success.

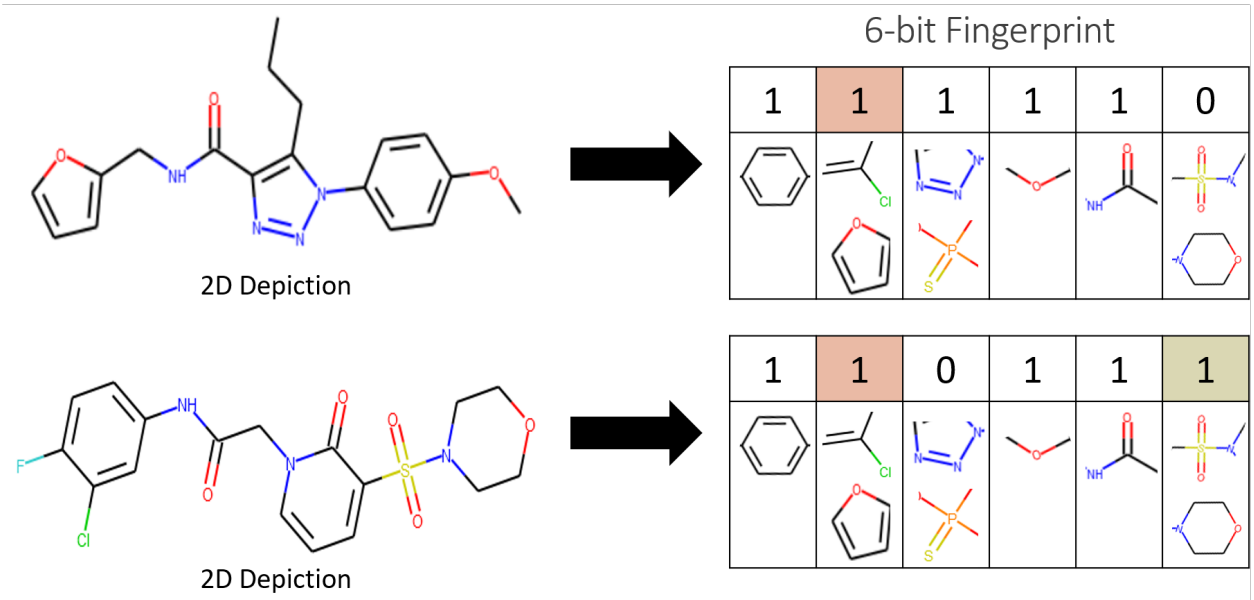


Figure 1.7: Simple fingerprint example showing substructure collision when using fixed-length bit strings. The between compounds collision can be seen in the third bit. The within compound collision of the bottom compound can be seen in the final bit.

1.6 Thesis Overview

This background discussion was intended to provide an overview of virtual screening and its purpose, use cases, important terms, and definitions for a new researcher in the field. The main focus of this dissertation is on the two introduced screening options of ORS and IBS. We are particularly interested in using supervised ligand-based methods in our ORS and IBS compound decision-making. The ORS option is a logical first step to try. In chapter 2, we discuss an ORS case study

on a protein-protein target with a prospective pool of 23,344 compounds. In chapter 3, we take it a step further and extend our analysis on the same target, but this time with two larger prospective pools of ~ 8.1 million and ~ 1.077 billion compounds, respectively. The goal of this extension was to see if ligand-based prioritization can also succeed in massive prospective pools.

In chapter 4, we shift gears towards the informer set problem. This problem was conceived when there is no initial screening data for the target of interest. We extend the work done by others on this problem by evaluating a wide range of solutions adapted from related domains on large datasets. The knowledge obtained from working on these three projects, and the discussions with closely affiliated members of the Small Molecule Screening Facility at UW-Madison provided a solid foundation for developing an IBS strategy. The IBS problem is formalized and a strategy is proposed in chapter 5.

In chapter 6, we summarize the goals and results of each chapter. We highlight the main outcomes, bring things into perspective, and give directions for future work. Finally, the chapters in this thesis follow chronologically, but each is self-contained enough to be read on its own.

— 2 —

Small-scale ORS Case Study on Protein-Protein Target

In this chapter we will discuss an in-house HTS project conducted at UW-Madison by three groups: Dr. Gitter's lab, Dr. Keck's lab, and the Small Molecule Screening Facility (SMSF). The project results were published in a paper [34] from which this chapter is derived, and co-authors contributed content. This project serves as a case study for applying ligand-based supervised learning approaches to the ORS scenario we described in section 1.4. Furthermore, it allows us to determine the efficacy of these methods in an ORS campaign before incorporating them in an IBS campaign.

2.1 Overview

Dr. Keck's lab at UW-Madison was interested in screening compounds for two protein-protein targets: PriA-SSB and RMI-FANCM [57] [58]. *Effective* compounds for these two targets are compounds that would compete with the binding of the protein-protein interaction; i.e. they would block the proteins from interacting, thereby *inhibiting* their function. The Life Chemicals (LC) vendor supplies a diverse collection of pre-plated compounds for HTS screens. The plan was to conduct two major screening phases where the first consisted of 72,423 compounds, and the second phase, 22,434 compounds. Both of these compound sets were purchased from LC. The goal of this project was to investigate the results of a practical pipeline applied to the ORS screening scenario as explained in section 1.5. The project can be summarized in the following steps:

1. For the **Pria-SSB** target, screen 72,423 compounds in vitro under two different bioassay protocols (explained later). For the **RMI-FANCM** target, in vitro screen 49,796 compounds.

2. Gather the results of the in vitro screen. This produces a dataset of continuous % inhibition scores of compound-target bioactivity. Another dataset is produced that *binarizes* the continuous scores to represent compound-target hits (i.e. 1 for hit and 0 non-hit).

3. Identify *good* virtual screening models for promoting a top k list of compounds to screen from the second phase prospective pool of 22,434 compounds. In order to do this, we conduct a stage of hyperparameter-sweeping (HS) and cross-validation (CV) to promote models using the first phase pool of 72,423 compounds as a training set (49,796 compounds for **RMI-FANCM**).

4. In vitro screen the **prospective** pool of 22,434 compounds. Gather the results and prepare the continuous and binary datasets.

5. Evaluate the top k promoted compounds from selected virtual screening models. Interesting question: how well did the estimated *best* model from the CV stage perform on the prospective dataset?

In the following sections we discuss these steps in more depth.

2.2 Dataset Preparation

The PriA-SSB target was screened against the 72,423 compounds under two assays. The first was an AlphaScreen (AS) assay at a single concentration (33.3 μM) used as an initial screen to identify potential hits on the 72,423 compounds. The results were curated to produce a continuous dataset of % inhibition scores: **PriA-SSB AS Continuous**. Compounds with % inhibition $\geq 35\%$ and pass PAINS structural filters [59] were screened in a secondary screen, but this time with a Fluorensence Polarization (FP) assay. PAINS filters are sets of less desirable substructures, and so,

compounds that possess these substructures are regarded as non-hits/inactives. The FP assay results were curated to prepare the **PriA-SSB FP Continuous** dataset. Both **PriA-SSB AS Continuous** and **PriA-SSB FP Continuous** datasets have a binary version we refer to as **PriA-SSB AS** and **PriA-SSB FP**, respectively. The binary threshold for AS and FP were set to $\geq 35\%$ and $\geq 30\%$, respectively (along with passing PAINS filters).

For the RMI-FANCM target, only 49,796 compounds of the 72,423 compounds were screened using an FP assay at a single concentration ($32\mu M$). The results were processed to produce the % inhibition dataset: **RMI-FANCM FP Continuous**. The binary threshold was set to ≥ 2 standard deviations from the mean to produce the binary dataset: **RMI-FANCM FP**. The binary dataset hits must also pass PAINS filters.

The secondary phase of 22,434 compounds were only screened against the PriA-SSB target using an AS assay at a single concentration ($33.3\mu M$); this is similar to the initial PriA-SSB AS screen on the 72,423 compounds. These compounds were screened at a much later date (greater than four months) than the initial 72,423 compounds. To produce the binary dataset, compounds with % inhibition $\geq 35\%$ and pass the PAINS filters were screened (retested) again using an AS assay. Retested compounds that exhibit % inhibition $\geq 35\%$ were deemed hits; all other compounds in the 22,434 set were considered non-hits. This curation produces the binary prospective dataset: **PriA-SSB prospective**. Virtual screening models trained on the **PriA-SSB AS** dataset are used to predict the bioactivity of the **PriA-SSB prospective** compounds. It is important to note that we did not have access to **PriA-SSB prospective** labels when the predictions were generated because the in vitro screen was not performed yet. Table 2.1 summarizes the PriA-SSB and RMI-FANCM datasets.

Finally, there's been recent hype around multi-task neural networks to learn from multiple compound-target bioactivity datasets; i.e. models that predict bioactivity against multiple targets [4] [60] [61] [62]. Ramsundar et al. [4] conducted a study on 128 compound-target bioactivity datasets comparing the performance of multi-task vs. single-task neural networks. One of their results showed that the performance of multi-task networks is better than single-task, *on average*;

although some tasks suffered in the multi-task setting. We employ the same dataset of 128 targets called 128-PCBA (in reference to the source: PubChem BioAssay) [5]. One of the virtual screening model classes used in this project is multi-task networks (MTNN) that incorporate the PriA-SSB, RMI-FANCM, and 128-PCBA datasets. The process of merging these target datasets and splitting into 5 stratified folds is a greedy approach that is described in the paper's appendix [34].

Table 2.1: Summary statistics for the four binary datasets.

Stage	Dataset	% inhibition threshold	# actives	# inactives
Cross-validation	**PriA-SSB AS**	$\geq 35\%$	79	72,344
	PriA-SSB FP	$\geq 30\%$	24	72,399
	RMI-FANCM FP	$\geq$ mean + 2 SD	230	49,566
Prospective	**PriA-SSB prospective**	$\geq 35\%$	54	22,380

2.3 Features and Models

For the models, we specified beforehand four classes of models: random forest (RF) [63], neural networks (NN), influence-relevance-voter (IRV) [64], and structure-based docking programs. A similarity baseline, which sets a compound's activity score as the fingerprint Tanimoto similarity of its nearest training set active, was added for comparison.

For NNs, we used three variants: Single-Task NN (STNN), Multi-Task NN (MTNN) [4], and Single-Task LSTM [65]. Furthermore, the STNN models are distinguished by their training labels as STNN-C for classification (binary dataset) and STNN-R for regression (continuous dataset). For the docking programs, 8 docking programs were used to produce 9 scores. An additional 5 consensus docking (CD) scores were computed as a function of these 9 program scores: max, mean, median, and two consensus measures [66]. We used three featurization methods: 1024-bit Morgan fingerprints [55], SMILES strings, and target and compound 3D structures. The Morgan fingerprints and SMILES were generated using the Python cheminformatics library RDKit [1]. The SMILES strings were used as input into Long-short-term-memory (LSTM) [8] networks (a variant of RNN), non-docking models used fingerprint features, and docking programs used the

compound-target 3D structures.

Table 2.2 summarizes the hyperparameter counts for all models. Table 2.3 summarizes the dataset used for each model type.

Table 2.2: Summary of hyperparameter counts for each stage.

Model	**Hyperparameter Stage**	**Cross-Validation Stage**	**Prospective Stage**
Docking and Consensus Docking (CD) Scores	14	14	1
STNN-C	24	2	1
STNN-R	12	2	1
MTNN-C	24	2	1
LSTM	90	2	1
RF	108	8	1
IRV	5	5	1
Similarity Baseline	1	1	1

Table 2.3: Model-to-dataset summary indicating which models were used for each dataset variant.

Model	Continuous % inhibition	Binary label	PCBA binary labels
Docking			
Consensus Docking (CD)			
STNN-C		✓	
STNN-R	✓		
MTNN-C		✓	✓
LSTM		✓	
IRV		✓	
RF		✓	
Similarity baseline		✓	

2.4 Evaluation Metrics

The ORS scenario is set up to emphasize hit retrieval since we only have one shot at selecting the top k most probable hits. Thus, we prefer metrics that better evaluate this early retrieval property. To that end, a common metric used in virtual screening is **enrichment factor** (EF) which is the ratio between the number of actives found in a model's top k rank of compounds versus the expected

number of actives in a random selection of k compounds. EF answers the question: how much better is the model over a random selection? The k parameter is set by a percentage argument $R \in [0\%, 100\%]$ that denotes the fraction of compounds to select from a database. We denote the EF metric tested at R percent of the database as: EF_R. An issue that arises when comparing EF_R across datasets is that it depends on the total number of actives within a dataset. To simplify the comparison, we introduce the maximum EF_R possible for a given dataset $EF_{max,R}$, and compute the ratio to produce a normalized enrichment factor: NEF_R.

$$EF_R = \frac{\text{\# actives in top } R \text{ ranked compounds}}{\text{\# actives in entire library } \times\ R} \tag{2.1}$$

$$EF_{max,R} = \frac{\min\{\text{\# actives, total \# compounds } \times\ R\}}{\text{\# actives in entire library } \times\ R} \tag{2.2}$$

$$NEF_R = \frac{EF_R}{EF_{max,R}} \tag{2.3}$$

All the virtual screening models produce a ranking of the compounds, from most probably active to least, and so we can compute area under the curve (AUC) metrics. We compute the following AUC metrics: Receiver-operating-characteristic (ROC), precision-recall (PR), Boltzmann-enhanced discrimination of ROC (BEDROC), and NEF_R at $R \in [0\%, 20\%]$ to favor early retrieval. AUC[ROC] measures the overall quality of a model at discriminating between actives and inactives, but not necessarily early retrieval. Due to the typical low number of actives, a more appropriate metric is AUC[PR] which measures the overall quality of a model at identifying actives. AUC[BEDROC] is skewed towards measuring a model's quality at early discrimination of actives and inactives.

Finally, we compute the number of hits found when selecting k compounds: n_{hits} at k tested compounds. The purpose of this is to compare other metrics with n_{hits} at low k values (e.g. 250 compounds) to identify which metric is suitable for early retrieval.

2.5 Pipeline Overview and Results

The pipeline proceeds as follows (see Figure 2.1 for illustration):

1. **Hyperparameter tuning stage** for RF and NN models. We define a set of parameters for RF and NN models, evaluate their performance on a training set, then promote a subset of the top performing parameters.

2. **Cross-validation stage** for promoted RF and NN models, along with IRV and docking models. The purpose of this stage is to use the training data with k-fold cross-validation to gauge future performance. Thus, promoting the best model from each class.

3. **Prospective stage** of using the training data and models promoted from the CV stage to score and rank the bioactivity of the **PriA-SSB prospective** compounds. We evaluate the performance of the promoted models with a focus on early hit retrieval; i.e. number of hits in the top 250 out of 22,434 compounds.

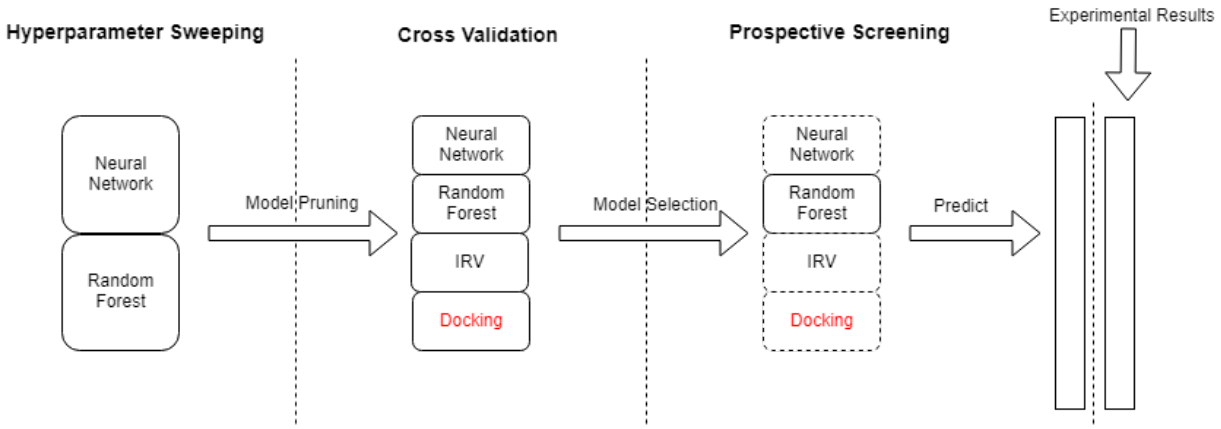


Figure 2.1: Pipeline illustration. In the hyperparameter stage, we prune 108 RF and 150 NN models. The cross-validation stage had a a total of 35 models. A random forest model was deemed the best in the CV stage. The best model from each class was promoted to the prospective stage.

Hyperparameter Tuning Stage

From Table 2.2, we pruned 108 RF models down to 8, 24 STNN-C down to 2, 12 STNN-R down to 2, 24 MTNN-C down to 2, and 90 LSTM down to 2 for the CV stage. In this stage we used 1 fold as the test set, and 4 folds for training. Results on this test fold were used to prune the model based on AUC[PR] metrics.

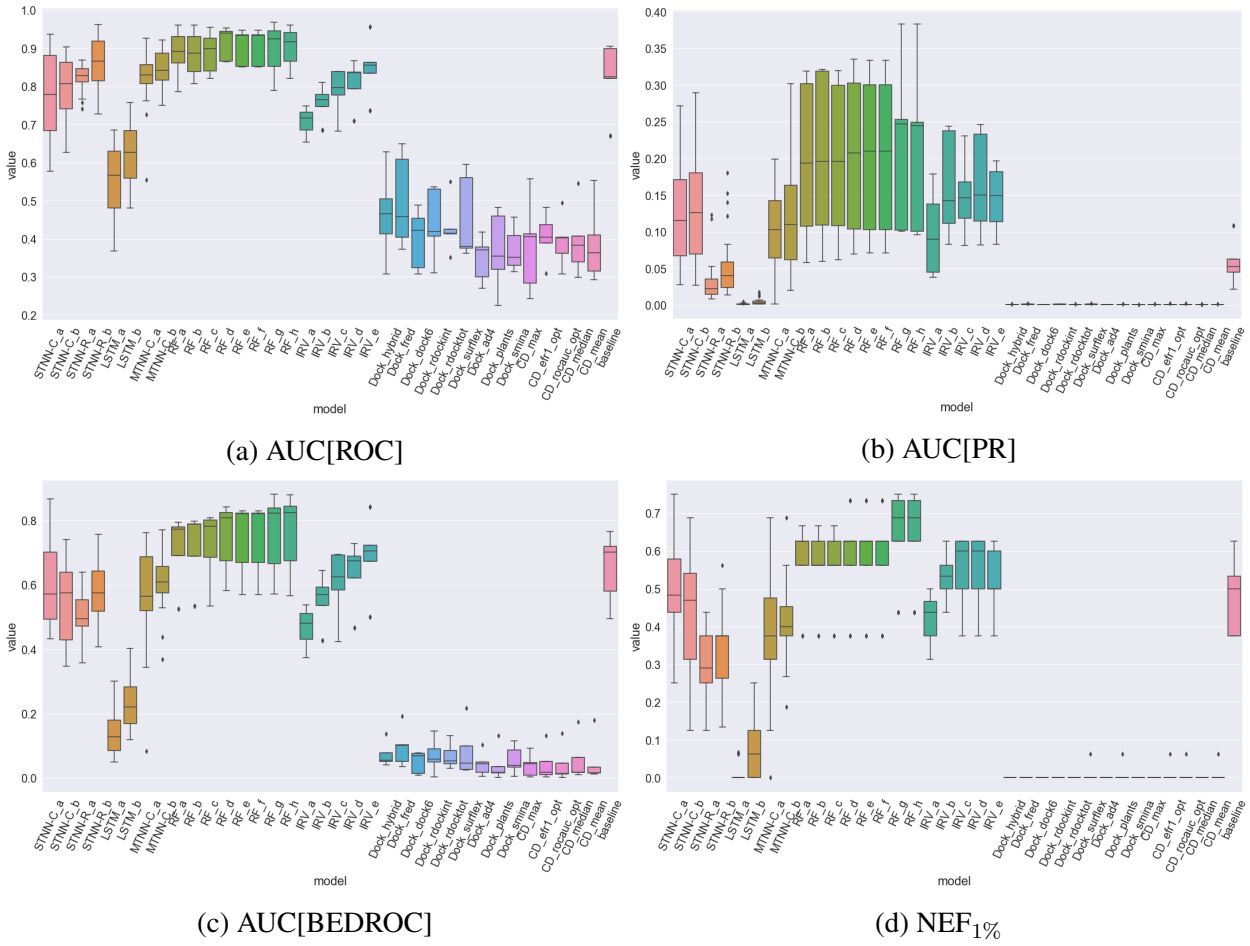
Cross-Validation Stage

For this stage we performed a 5-fold CV evaluation for 36 models (see counts in Table 2.2) on the three datasets: **PriA-SSB AS**, **PriA-SSB FP**, and **RMI-FANCM FP**. Models were evaluated on the following metrics: AUC[ROC], AUC[PR], AUC[BEDROC], AUC[NEF$_R$], EF$_R$, NEF$_R$ at $R \in [0\%, 20\%]$, and n_{hits} at $k \in \{250, 500, 1000, 2500, 5000, 7500, 10000\}$. For every metric, we performed Dunnet-Tukey-Kramer (DTK) multi-comparison test [67] [68]; this was because of the unequal variances and sample sizes (NN models performed 5×4-fold CV) as shown in Figure 2.2. The significance results of DTK, along with fold means for tie breaking, allow us to induce a ranking among the models in terms of wins. This ranking was computed for each metric.

The next question we wanted to answer is: which metric is a good indicator of early hit retrieval? To answer this, we used the rankings produced by each metric and compared it with the ranking produced by n_{hits} at $k \in \{100, 250, 500, 1000, 2500\}$. Using Spearman's rank correlation coefficient, and focusing on **PriA-SSB AS**, we noticed that NEF$_R$ consistently correlated highly with n_{hits} for which the fraction R and the number of tests k coincide. Thus, we made the decision to promote the models that perform the best on NEF$_{1\%}$ in the CV stage. The fraction $R = 1\%$ was selected because we wanted models that prioritize early retrieval; i.e. we want to select the top 250 out of 22,434 compounds from **PriA-SSB prospective**.

Table 2.4 shows the best model on different metrics promoted by two ranking methods: Mean and DTK+Mean. DTK+Mean was used since many models showed no significance when using DTK resulting in many ties in the rankings, and so we used the mean to break ties. We opted for DTK+Mean over the Mean ranking since it also accounts for the variance in the fold results.

Ultimately, the following models from each class were promoted based on $NEF_{1\%}$ to the prospective stage: RandomForest_h, SingleClassification_a, SingleRegression_b, MultiClassification_b, LSTM_b, IRV_d, and ConsensusDocking_efr1_opt. Based on the CV results, we expect RandomForest_h to perform the best in the prospective stage.



(a) AUC[ROC]

(b) AUC[PR]

(c) AUC[BEDROC]

(d) $NEF_{1\%}$

Figure 2.2: **PriA-SSB AS** results for the CV stage on the 36 promoted models. The metrics are: (a) AUC[ROC], (b) AUC[PR], (c) AUC[BEDROC], and (d) $NEF_{1\%}$. Note that docking programs do not use training data; see structure-based virtual screening discussion in section 1.2.

Prospective Stage

The 22,434 compounds from **PriA-SSB prospective** were screened at a much later date, but the compounds *without activity labels* were provided earlier. The 7 promoted models from the CV stage were trained on the 72,423 compounds from **PriA-SSB AS**, and then computed predictions on the

Table 2.4: Top ranked models by means versus DTK+Mean on the three tasks. Model suffix characters refer to specific hyperparameters sets.

Metric	Best by Mean Model			Best by DTK+Mean Model		
	PriA-SSB AS	**PriA-SSB FP**	**RMI-FANCM FP**	**PriA-SSB AS**	**PriA-SSB FP**	**RMI-FANCM FP**
AUC[ROC]	RF_d	STNN-R_a	RF_h	RF_d	STNN-R_a	RF_h
AUC[BEDROC]	RF_h	STNN-R_b	RF_h	RF_h	STNN-R_b	RF_h
AUC[PR]	RF_g	STNN-R_a	RF_h	STNN-C_b	STNN-R_b	STNN-C_b
AUC[NEF]	RF_h	STNN-R_b	RF_h	RF_h	STNN-R_b	RF_h
NEF$_{1\%}$	RF_h	STNN-R_b	RF_h	RF_h	STNN-R_b	RF_h

22,434 compounds from **PriA-SSB prospective**. These predictions and rankings were saved to be evaluated after the screening. Table 2.5 summarizes the early hit retrieval results. Our expectation from the CV stage turned out to be true, the RF_h model was the best prospectively identifying 37 out of 54 hits in its top 250 ranked predictions. Compounds were clustered using two methods: similarity-based clusters (SIM) and maximum-common-substructure clusters (MCS). The SIM method clusters compounds using fingerprints and Ward's hierarchical clustering [69]. The MCS method clusters compounds with similar scaffold structures together. These clustering methods are used to quantify the diversity/novelty of hits; i.e. the number unique clusters containing at least one hit compound. The RF_h model also identifies the most diverse set of hits as denoted by both SIM and MCS.

Figure 2.3 showcases the overlap between the models with regards to hits (total of 54) identified in the top 250. The same eleven hits were identified by five of the models. The RF_h model finds seven more hits than the similarity baseline. Interestingly, the RF_h model finds all hits covered by other models except for six hits that were uniquely covered by STNN-R_b (five) and the baseline (one), respectively. This suggests that we might benefit from ensembling classification and regression models for future work.

2.6 Conclusions and Future Work

The pipeline of a CV stage to judge future performance in the prospective stage was successful for the PriA-SSB target. In the end, RF_h was the best model in the CV stage and the prospective stage

Table 2.5: Number of hits found in the top 250 out 22,434 compounds from **PriA-SSB prospective** using the promoted CV models (along with the similarity baseline). As a measure of hit diversity/novelty, we include the number of similarity-based clusters (SIM) and maximum-common-substructure clusters (MCS) that contained the identified hits.

Model	Hits	Hits not in baseline	SIM clusters	MCS clusters
Experimental	54	–	27	35
Similarity baseline	31	–	14	17
CD_efr1_opt	0	0	0	0
STNN-C_a	21	2	11	13
STNN-R_b	28	8	14	18
LSTM_b	1	1	1	1
MTNN-C_b	27	3	13	17
RF_h	37	7	17	22
IRV_d	29	4	15	18

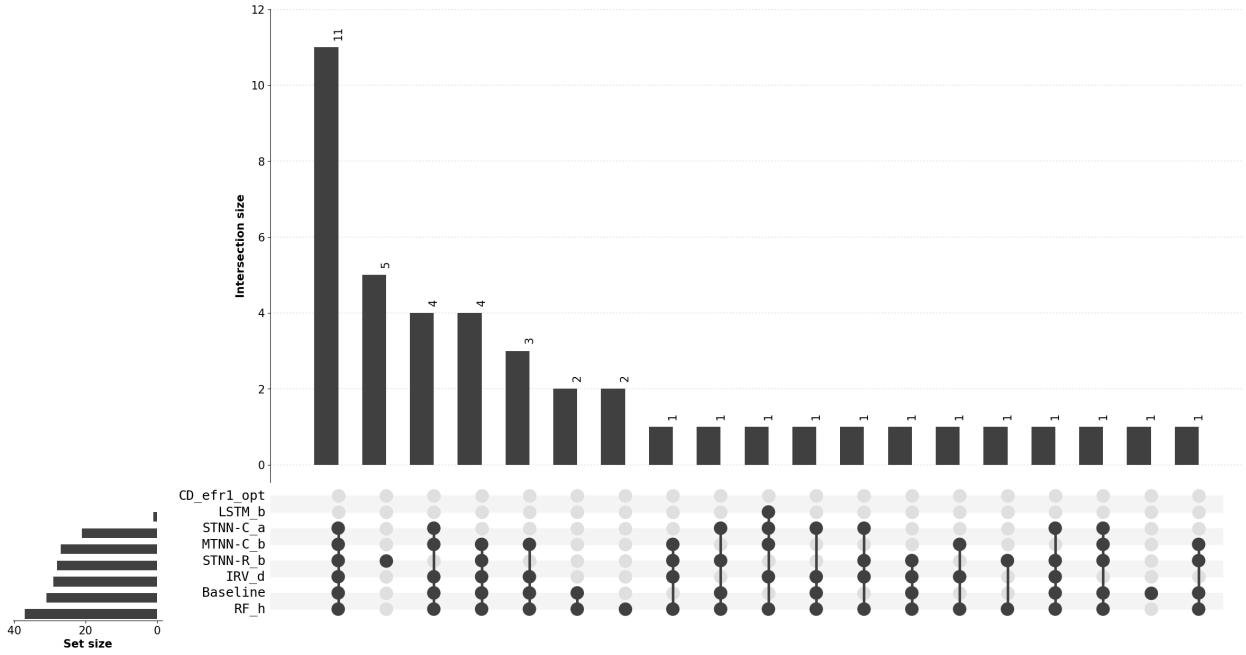


Figure 2.3: UpSet plot [10] comparing the hits overlap between the models and similarity baseline. A total of 43 out of 54 hits were identified by all models.

for early hit retrieval. Enrichment factor metrics (EF_R and NEF_R) were useful for measuring early hit retrieval. On the other hand, AUC[ROC] and AUC[PR] are better suited for comparing *overall* performance of methods; i.e. not for early hit retrieval [70].

Although docking programs had vastly underwhelming performance for this target, we cannot generalize their performance to other targets. Docking programs could still have usefulness in cases

where no bioactivity data is present. Recent hype around deep learning models in drug discovery showed they outperform random forests on other datasets [71, 60, 72]. This was not the case for the PriA-SSB target in this project. This does not refute that deep learning models are superior as that would require comparing models across a wide range of targets. The goal of this project was to evaluate early hit retrieval when following a pipeline for promoting the expected best ligand-based model from a pool of models to the prospective stage.

The RF_h model found 37 out of 54 hits in the top 250 compound prediction scores. Objectively, that is 68.52% of hits in just 1.11% of the prospective pool. However, RF_h failed to find six hits found by STNN-R_b (five) and the baseline (one). Investigating **ensembling** classification and regression models is an interesting avenue for future work.

This project served as a case study for an ORS campaign aided by virtual screening in a practical setting. The early hit retrieval results on this small pool of 22,434 compounds was satisfying for our group. The results supported that supervised learning models are of great help in prioritizing hits. However, the size of the training set in this project is more than three times larger than the prospective pool. A logical followup is applying this ORS setup in a scenario where the prospective pool is much larger. Is it possible to find satisfying results in such a setting? This motivated us to apply a similar pipeline on the same PriA-SSB target, but this time on much larger prospective pools from two commercial vendors. The details of this followup ORS project is discussed in chapter 3.

Another logical consideration is whether or not one round of screening is the best course of action. In this project, we initially screened 72,423 compounds to construct the training set. Then we screened 22,380 compounds and prospectively evaluated prioritizing the top 250 predicted compounds. The total number of hits from both screens is 133. Was it necessary to screen 94,803 compounds to find these 133 hits? Our intuition was that, starting from a very small training set, we can proceed with multiple rounds of screening in order to maximize hit retrieval. In each round (or iteration), we carefully select compounds for screening that are either hits or help us in finding hits. This is the iterative batched screening setting that we introduced earlier in section 1.4. A number of question arise in such a setting like the batch size, the number of iterations, and the selection

strategy. We were again motivated to investigate the efficacy of IBS with a proposed strategy. The details of the IBS strategy and results is discussed in chapter 5.

— 3 —

Massive-scale ORS Case Study on Protein-Protein Target

3.1 Motivation and Overview

This project is an extension of the work done on the protein-protein interaction target PriA-SSB in chapter 2. With the success of the ORS pipeline on the Life Chemicals Inc. (LC) library, it was logical to followup with an investigation on larger prospective pools. In this chapter, we denote the LC training set and prospective from chapter 3 as LC123 and LC4, respectively. The numbers are legacy identifiers that were defined by the Small Molecule Screening Facility (SMSF) at UW-Madison. As discussed in 2019 by Hoffmann et al. [21], public and commercial libraries can range from millions to billions of compounds, with the sizes of these pools increasing each year. To this end, we investigated a virtual screening ORS pipeline (similar to that of chapter 2) when applied to two large commercial prospective pools: Aldrich Market Select (AMS) and Enamine REAL [24, 22] containing 8,187,682 and 1,077,562,987 compounds, respectively. The content in this chapter comes from our publication in preparation, with co-authors contributing content[1]. As such, exact details are glossed over for the sake of brevity and will be contained in the forthcoming publication. The goal of this project was to evaluate the success of ligand-based supervised learning in the ORS scenario on **larger** prospective pools.

The training dataset for this project comes from two libraries: Life Chemicals Inc. (LC) and

[1] Anticipated authors: Moayad Alnammi, Shengchao Liu, Spencer S. Ericksen, Gene E. Ananiev, Andrew F. Voter, James L. Keck, Michael F. Hoffmann, Scott A. Wildman, and Anthony Gitter.

Molecular Libraries Probe Production Centers Network (MLPCN). The LC library consists of the LC1234 compounds used in our previous ORS pipeline presented in section 2.2. The MLPCN compounds were screened using the same protocols as the LC compounds [58]. The primary and retest screens for these libraries were processed in order to determine activity labels (details in section 3.3). We considered a set of supervised learning models, and through a series of cross-validation stages, we promoted a single prospective model. The prospective model and a Similarity Baseline are given similar budgets, and are used to select compounds from the AMS pool. If the prospective model performs well on AMS, then feeling confident, we followup with the larger Enamine REAL pool.

As an overview, the project pipeline is as follows:

1. Process the primary and retest % inhibition data for LC and MLPCN libraries to generate a binary training set.

2. Define a set of machine learning models to consider. We proceed in stages in which we compare the performance of the set of models using the data. In each stage, we **promote** a subset of the top performing models to the next stage. The end goal is to promote a single model for use in prospective screening.

3. Use the training set and cross-validation stages to promote a single prospective model from the set of models.

4. Given a budget k, use the prospective model and the Similarity Baseline to select the top k promoted compounds from the AMS pool. Screen these compounds in vitro, determine hits, and analyze the performance.

5. Given a budget m, use the prospective model to select the top m promoted compounds from the Enamine REAL pool. Screen these compounds in vitro, determine hits, and analyze the performance.

3.2 Related Work

In this section we discuss works that perform large pool ORS studies in a similar manner to our own. In particular, we focus on work that scored large prospective pools using virtual screening, and then subsequently screened a selection of compounds in vitro. In Table 3.1 we summarize five relevant works by the training set size, prospective pool size, prospective experimental size, and the type of virtual screening method employed. Due to the limited table space, we list the virtual screening method type as either ligand-based supervised learning or structure-based docking. However, note that some of these works used additional steps in their selection. For completeness, we summarize these works here, but the main takeaways are the training set and prospective pool sizes in Table 3.1 that provide contrast with the sizes in our work.

In 2017, Kutchukian et al. [73] published an ORS case study where they employed two supervised learning models to prioritize compound selection. Both models are naive Bayes classifiers that predict hit probabilities, but differ in their input features; one uses compound structural fingerprint features and the other uses compound-target biological features (HTSFP). 45000 diverse compounds were initially screened and served as the training data for the models. After training on this set, each model computed predictions on a $\sim$2.7 million compound pool. The top 7500 predictions from each model were then screened in vitro. Out of the 15000 screened compounds, 600 were determined as hits. The overlap between the selection sets of the two models was small; only 363 out of 15000 compounds.

In 2019, Lyu et al. [74] used docking methods to prioritize compound selection against two targets of interest. For the first target, compound docking scores were computed on a 99 million compound pool. 51 compounds were selected on the basis of docking score, dissimilarity to known hits, and diversity. Only 44 (of the 51) were synthesized and screened, yielding 5 hits. For the second target, they had two goals: find novel hits and analyze the relationship between hit rates and docking scores. Docking scores were computed on a 138 million compound pool which were then binned. 549 compounds were selected from bins of high, middle, and low docking scores. Their conclusion was that hit rates decrease at lower docking scores.

In 2020, McCloskey et al. [75] conducted a large prospective pool study of ~ 88 million compounds via DNA encoded small molecule libraries (DELs) [76] on three protein targets. These compounds come from two libraries: a proprietary library (XVL) of ~ 83.2 million compounds and a commercial library (Mcule [23]) of ~ 4.6 million compounds. The training set of ~ 1 million disynthons was developed via DEL selection, tested against the targets, and classified into five classes based on their inhibition. Two main classes of models were trained: random forests (RF) and graph convolution neural networks (GCNN) [63, 42]. The final RF and GCNN models computed predictions on the ~ 88 million prospective pool. 1885 compounds were selected based on prediction score and clustering. After screening, the results showcase that the GCNN outperforms the RF in hit rate across all three targets.

In 2020, Stokes et al. [77] employed a deep learning model to prioritize antibiotic drugs in three screens whereby the training set and prospective pool increase with each screen. An initial training set was constructed by screening 2335 compounds against E. coli growth inhibition. After training the model, predictions were computed on the first prospective pool of 6111 compounds. The top 99 predicted compounds were screened, and 51 were deemed as hits. These 99 compounds were then added to the training set and used to retrain the model. Compound predictions were then computed on the second pool of 9997 compounds. The top 200 and bottom 100 predicted compounds were screened, but none showed inhibition against E. coli. The authors note that this may be due to low prediction scores on this library (the maximum being 0.37). Finally, the model was retrained with the added data, and predicted scores for ~ 107 million compounds. They selected 23 compounds that had greater than 0.8 prediction score and less than 0.4 Tanimoto similarity to any known antibiotic. These compounds were screened against five bacterial species, and 8 of the 23 showed inhibition against at least one of the targets.

In 2020, Gorgulla et al. [78] introduced a multi-stage virtual screening system called VirtualFlow that uses docking programs to score a large prospective pool. To showcase VirtualFlow, they applied it to a protein-protein target of interest. The prospective pool combined Enamine REAL [22] and ZINC15 [20] databases for a total of ~ 1.3 billion compounds. 590 compounds were experimentally

screened, and 69 were deemed as hits (followup optimization protocols refined this number). The prospective pool size used in this study is the largest we are aware of that proceeds in an ORS manner; i.e. employs virtual screening to select compounds, and then screen them in vitro. However, note that virtually screening this large prospective pool using VirtualFlow took 4 weeks and $\sim$8000 cores; i.e. $\sim$5.38 million CPU hours.

Table 3.1: Summary of works that involved prospective screening aided by a virtual screening method. The virtual screening method types are designated as ligand-based (LB) or structure-based (SB). We also list our work in this chapter for contrast.

Paper	Year	Training Set Size	Prospective Pool Size	Prospective Experimental Size	Method Type
Kutchukian et al. [73]	2017	45K	$\sim$2.7M	15K	LB Supervised Learning
Lyu et al. [74]	2019	-	99M	44	SB docking
		-	138M	549	
McCloskey et al. [75]	2020	1M	$\sim$88M	1885	LB Supervised Learning
Stokes et al. [77]	2020	2335	6111	99	LB Supervised Learning
		2434	9997	300	
		2734	$\sim$107M	23	
Gorgulla et al. [78]	2020	-	$\sim$1.3B	590	SB docking
This work	2021	427300	$\sim$8.187M	1024	LB Supervised Learning
		427300	$\sim$1.077B	68	

Finally, although not a prospective screening study, Hoffmann et al. [21] discussed the problems associated with large compound pools. Well defined rules enable generation of massive virtual libraries. However, many of these virtually constructed compounds are difficult to synthesize. The paper discusses this issue and the synthetic procedures adopted by commercial vendors to reduce failures. They recommend the use of virtual screening methods in selecting compounds from these pools. The paper also provides a concise summary of proprietary, public, and commercial compound pools and their sizes (ranging from 10^6 to 10^{20} compounds).

3.3 Datasets Overview and Preparation

As mentioned previously, the MLPCN compounds were screened using the same protocol as the LC library. There were five screens that were conducted: LC123 primary, LC123 retest, LC4 primary, MLPCN primary, and MLPCN retest. Retest screens were conducted on primary compounds from the same library that had a primary % inhibition $\geq 35\%$ and pass PAINS filters [59]. Table 3.2 summarizes the compound counts for each library involved in this project. Note that a single compound in LC1234 and MLPCN can have two or more % inhibition readings from their primary and retest screens. The training set is constructed by merging the LC1234 and MLPCN primary and retest readings such that each unique compound has a single entry. For each compound, its readings are grouped, and it is defined as a hit if it passes all the following filters: the median % inhibition of primary screens is $\geq 35\%$, the median % inhibition of retest screens is $\geq 35\%$, and the compound does not match a PAINS filter [59]. Finally, the training dataset was split into 10 folds based on both library and binary activity.

The training set was used in building and promoting supervised learning models in the cross-validation stages. For all models, inputs were the 1024-bit RDKit Morgan fingerprint features [1, 79]. Furthermore, the training set compounds were clustered using a custom implementation of the Taylor-Butina algorithm [80, 81] at 0.2, 0.3, and 0.4 Tanimoto distance thresholds. The algorithm assigns compounds that are within the threshold distance of each other to the same cluster. As a result, clustering is used as a proxy for novelty or diversity, and in particular, in computing the total number of unique clusters containing hits in the top selections. For example, let's say there are 10 hits in the top 100 compounds selected by a model, the number of unique clusters in those 100 compounds that contain at least one hit gives us a measure of the diversity of the hits. In other words, it quantifies how many hits come from a unique cluster. In the two prospective stages, the AMS and Enamine REAL compounds were featurized similarly, and use the promoted prospective model to compute hit predictions.

Table 3.2: Summary of libraries used in the chapter 3 ORS project. The training set merges the primary and retest datasets of LC1234 and MLPCN libraries. In the training set, there are a total of 554 hits.

Stage	Library	# Compounds
-	LC123 (Primary and Retest)	74,763
-	LC4 (Primary)	25,278
-	MLPCN (Primary and Retest)	337,104
Cross-validation Model-Selection Prospective-training	Training set	427,300
Prospective-prediction	AMS	8,187,682
Prospective-prediction	Enamine REAL	1,077,562,987

3.4 Pipeline Summary

Table 3.3: Summary of model classes and the counts of their hyperparameter sets as they are promoted through the stages. At the end, we promote a single top performing prospective model.

Model	Cross-Validation	Model Selection	AMS Prospective	Enamine Prospective
RF-C	216	21	1	1
XGB-C	1000	21	-	-
XGB-R	1000	21	-	-
NN-C	432	20	-	-
NN-R	432	20	-	-
Similarity Baseline	-	1	1	-
Model-Based Ensemble	-	13	-	-
Max Vote Ensemble	-	13	-	-
Total	3080	130	2	1

The pipeline is similar to the previous ORS project in chapter 2. We consider the following machine learning model classes: random forests (RF) [63], eXtreme Gradient Boosting (XGB) [82], neural networks (NN), and ensembles. The C and R post-fix notation denote classification and regression learning; i.e. classification on active/inactive classes and regression on % inhibition. The Similarity Baseline was added as a benchmark for comparison. A range of hyperparameter sets are sampled for each model class as seen in the cross-validation column in Table 3.3. The **cross-validation stage** conducts 4-fold cross-validation with varying splits on the first 8 folds (out of the 10 training

folds) and promotes the top 20 models (with ties) from each model class based on $NEF_{1\%}$. Recall from section 2.4 that this metric judges a model's early retrieval performance in the top 1% of a prospective pool. From Figure 3.1, the top 21 RF-C models outperform the top 20 from other model classes on the mean 4-fold $NEF_{1\%}$ metric.

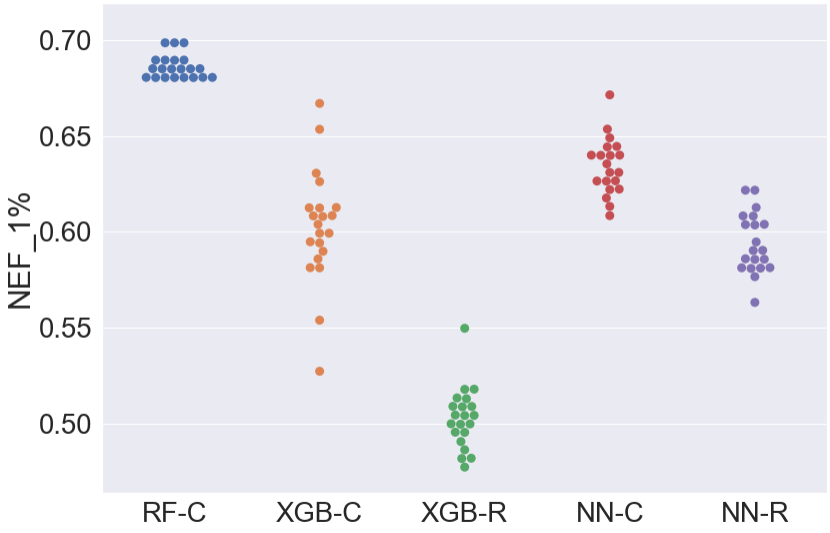


Figure 3.1: Cross-validation mean of 4-fold performance based on $NEF_{1\%}$ for the top 20 hyperparameter sets (with ties) from each model class.

The **model selection stage** makes use of all 10 training folds, and adds the Similarity Baseline and two ensemble variants for evaluation. These ensembles differ in how they incorporate their voters whereby one uses the max voter and the other uses a linear combination of the voters. The first 8 folds are used for training and the last two folds are used for validation and testing, respectively. Table 3.4 shows the $NEF_{1\%}$ results on the test fold for the top performer from each model class. The top RF-C model has the highest $NEF_{1\%}$ value.

Table 3.4: Model selection stage's $NEF_{1\%}$ results of the top performing model from each model class on the test fold.

Model	$NEF_{1\%}$
RF-C	0.636
XGB-C	0.582
XGB-R	0.418
NN-C	0.582
NN-R	0.491
Similarity Baseline	0.400
Model-Based Ensemble	0.618
Max Vote Ensemble	0.618

The intent was to select the prospective model based on the test fold performance in the model selection stage. This would correspond to the RF-C model with $NEF_{1\%}$ performance as shown in Table 3.4. However, another RF-C model with a different hyperparameter setting was selected inadvertently. The selected RF-C model was the best performer in the cross-validation stage based on mean 4-fold $NEF_{1\%}$ performance. In the model selection stage, its $NEF_{1\%}$ performance on the test fold was 0.618 which is greater than or equal to the top performers from other model classes.

3.5 AMS Prospective Summary

As aforementioned, we promoted an RF-C model to the prospective screening stage. We also promote the Similarity Baseline for a benchmark comparison. In this prospective stage, both models are trained on the training set, and subsequently used to compute hit predictions on the entire AMS pool of 8,187,682 compounds. For each of the RF-C and Similarity Baseline, we select the top 1500 predicted compounds. This produces two sets of 1500 compounds that contain overlapping compounds. We apply cost, availability, and delivery time filters to each set. The two filtered sets are merged for a total of 1028 unique compounds which were ordered from AMS.

Only 1024 of these compounds were successfully screened in two replicates. Figure 3.2 shows the distribution of the % inhibition readings for each replicate. The means were around 50% and 70% for replicates 1 and 2, respectively. Rather than sticking to the training activity cutoff of 35%,

we wanted to be more stringent in our activity criteria and set it to 50%. Thus, an active compound must pass the 50% threshold in both replicates and not match a PAINS filter [59].

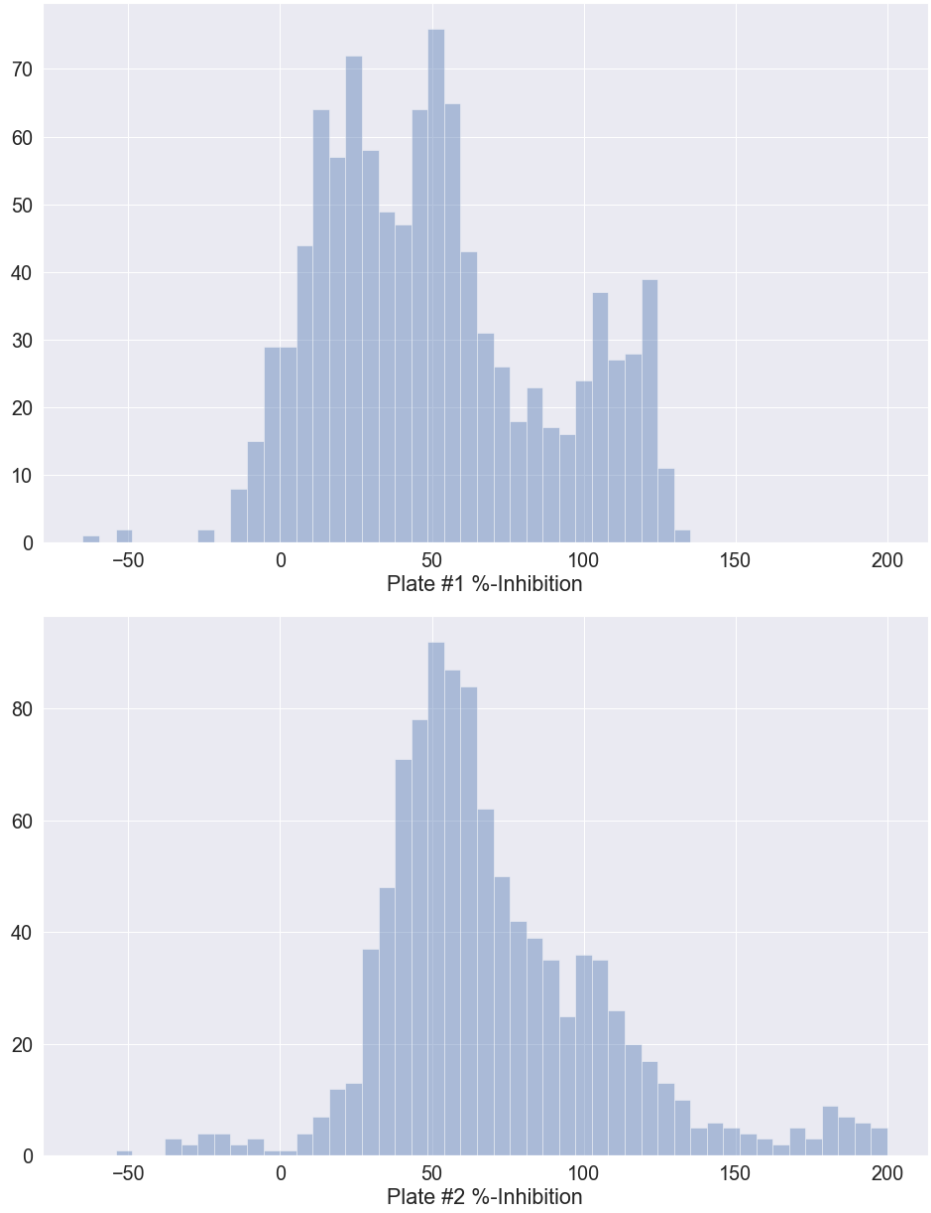


Figure 3.2: Replicate % inhibition distributions of the 1024 ordered AMS compounds.

The training set consisted of 427,300 compounds with 554 hits, and so we expect 1.3 hits in a random sample of 1024 compounds. Table 3.5 summarizes the hit results of the 1024 AMS compounds depending on the selector model. In total, 412 hits were found among the 1024 compounds. Both models have a similar budget of 701 and 705 compounds for RF-C and Similarity Baseline,

respectively. There is overlap in selecting 382 compounds, resulting in both models agreeing on 181 hits. However, looking at non-overlapping selections (i.e. selected by one model but not the other), the RF-C model finds roughly two times as many hits as the Similarity Baseline (156 and 75 hits, respectively).

Novelty is also important for directing future lead optimization efforts. For novelty we make use of the Taylor-Butina [80, 81] clustering of the training set and the 1024 AMS compounds. We use two metrics for novelty: unique cluster hits and novel cluster hits. Unique cluster hits are the number of unique clusters with at least one active; this measures overall diversity of a selector. Novel cluster hits is the number of unique clusters with at least one active, but these clusters have not been encountered (i.e. do not exist) in the training set; this measures the ability to extend to hits beyond the training set hits. Table 3.6 summarizes these two novelty metrics based on the selector. On the harder novel cluster hits metric, the RF-C model stands out as it finds 44 novel hit clusters to the Similarity Baseline's 13.

The diversity of the selection by the RF-C model is further supported by Figure 3.3, showcasing that the Tanimoto distance distribution between hits and their nearest training set actives had almost two times the range under RF-C than the Similarity Baseline. As such, the RF-C is able to extend beyond the training set clusters and find distant hits relative to the Similarity Baseline. Although both models are non-linear, the RF-C is perhaps better suited for the fingerprint features as it can build its decisions on a subset of the bits. Recall that these bits correspond to substructures that exist in the compound. One hypothesis is that the RF-C can determine that two substructures from two very distant hit compounds are important for activity. Thus, when a new compound with these two substructures is passed to the RF-C, it is given a high probability score. However, more testing with RF models is needed to confirm this.

Objectively, the RF-C model clearly outperforms the Similarity Baseline in terms of total hits and the two novelty metrics. Furthermore, both models exceed the expectation of 1.3 hits in a random sample of 1024 compounds (from 0.13% training set hit rate).

Table 3.5: Summary results of the 1024 AMS ordered compounds when grouped by the selector.

Selector	Count	Hits	Misses	Hit Rate (%)
RF-C or Similarity Baseline	1024	412	612	40.23
RF-C	701	337	364	48.07
Similarity Baseline	705	256	449	36.31
RF-C and Similarity Baseline	382	181	201	47.38
RF-C but not Similarity Baseline	319	156	163	48.90
Similarity Baseline but not RF-C	323	75	248	23.22

Table 3.6: Summary of cluster hit metrics on the 1024 AMS selected compounds. Taylor-Butina clustering was used with a cutoff of 0.4.

Selector	Unique Cluster Hits	Novel Cluster Hits
RF-C or Similarity Baseline	169	72
RF-C	142	61
Similarity Baseline	115	30
RF-C and Similarity Baseline	88	19
RF-C but not Similarity Baseline	72	44
Similarity Baseline but not RF-C	40	13

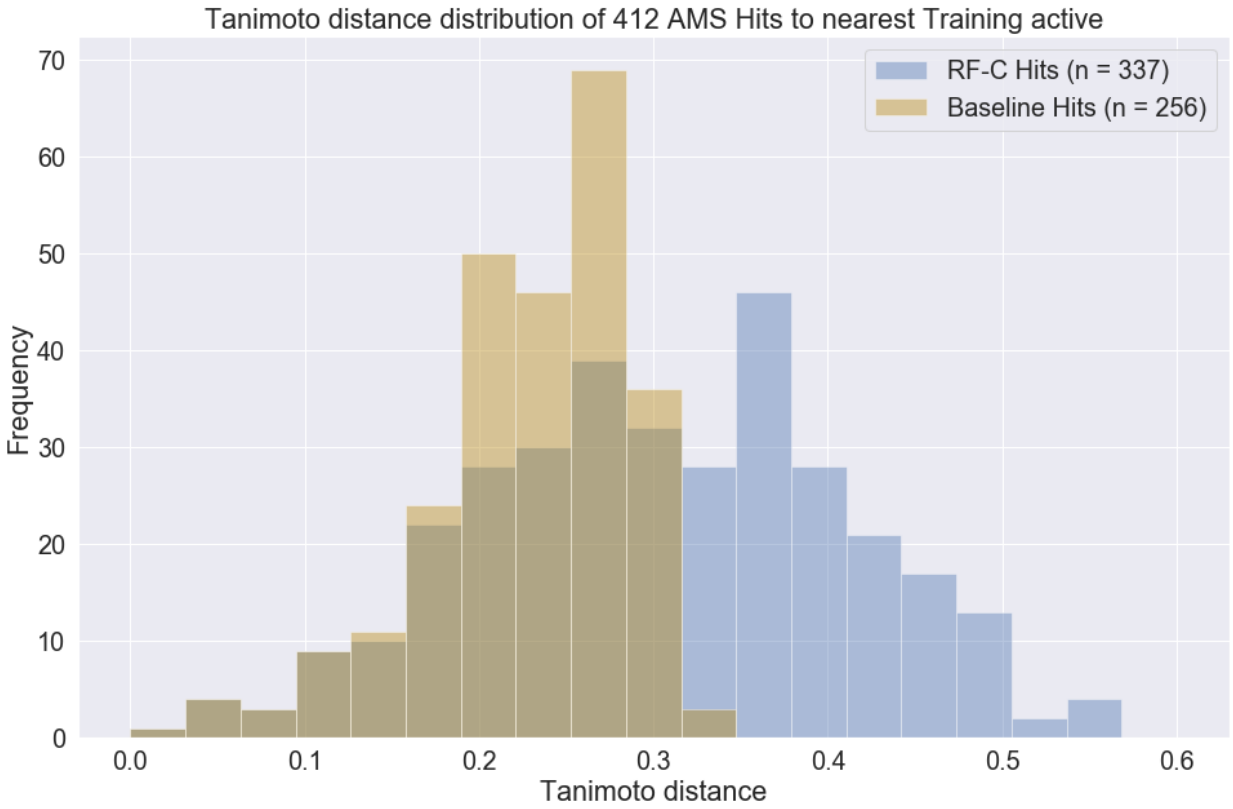


Figure 3.3: Tanimoto distance distribution of the 1,024 AMS hit compounds to their nearest training set actives color-coded according to the selectors RF-C (blue) and the Similarity Baseline (tan).

3.6 Enamine REAL Prospective Summary

The results of the 1024 AMS prospective screen were satisfying for the team. Following this, we decided to prioritize compounds from the Enamine REAL pool of 1,077,562,987 compounds. We used the same AMS prospective RF-C model that was trained on the 427,300 training set compounds. That is, we do not add the 1024 AMS screen results to the training. The trained RF-C model was used to predict the hit probabilities of the Enamine REAL pool. Due to the large size of the pool and the parallelizability of compound prediction, we split the pool prediction among 18 compute nodes. With this setup, predictions on the entire Enamine REAL pool were completed in less than three days.

After computing the predictions, we proceeded with the following selection steps:

1. Select the top 10,000 ranked predictions from the Enamine REAL pool. This pruned the list from 1,077,562,987 to 10,000 compounds.

2. We requested quotes for these 10,000 compounds from Enamine. Due to availability and delivery constraints, only 5,620 compounds were available. This pruned the list from 10,000 to 5,620 compounds.

3. Due to budget constraints, we decided to select 100 diverse compounds. The training set, 1024 AMS compounds, and the 5,620 Enamine REAL compounds were clustered using Taylor-Butina with a 0.4 cutoff. We kept clusters that did not contain training or AMS actives in order to promote hit finding from new clusters. This pruned the list from 5,620 to 2,679 compounds.

4. Apply PAINS filter, Inpharmatica filter, and Lipinski's rule of five. Pan Assay Interference Compounds (PAINS), introduced by Baell et al. [59], are compounds which contain substructures that typically interact with multiple targets rather than just the target of interest. As a result, PAINS compounds typically appear as hits (false-positives) in many screens. The Inpharmatica filter is also a set of substructures that tend to give false-positive readings as defined by Inpharmatica Ltd. The Lipinski rule of five specify common properties that drug-like compounds exhibit [15, 16]. The PAINS filter uses RDKit's FilterCatalog [1]. The Inpharmatica and Lipinski filters use rd_filters (`https://github.com/PatWalters/rd_filters`) [83, 84, 85]. This pruned the list from 2,679 to 1,604 compounds.

5. Select compounds that have a fingerprint Tanimoto distance greater than or equal to 0.35 from their nearest training or AMS active. This pruned the list from 1,604 to 1,354 compounds.

6. Select the highest predicted compound from each of the remaining clusters. This pruned the list from 1,354 to 311 compounds.

7. In a greedy, iterative manner, select the 100 most distant compounds. We used the fingerprint Tanimoto distance and selected the first compound as the one with the highest prediction score.

For iterations 1 to 100, we select the compound with the largest mean Tanimoto distance to the already selected compounds. This pruned the list from 311 to 100 compounds.

8. We sent quote requests for these 100 compounds to Enamine. 90 of these compounds were available and within our budget.

9. We requested the 90 compounds, but only 68 were successfully synthesized and delivered.

After receiving the 68 compounds, screening was done in four replicates at (33 µm) under the same protocols as the training and 1024 AMS screens. Compound hit identification was based on the median of the four % inhibition readings. Figure 3.4 shows the median % inhibition distribution of the 68 compounds. We used the same stringent 50% threshold of 1024 AMS screen for determining hits. With the training set hit rate of 0.13%, a random sample of 68 compounds is expected to yield 0 hits. The results vastly exceed this expectation with 31 hits out of 68 as summarized in Table 3.7. Furthermore, due the nature of our selection process of excluding training and AMS active clusters, the diversity cluster metrics are equal to the number of hits. Illustrations of these 31 and comparison with their nearest training or 1024 AMS actives can be seen in Table 3.8. The hits have an edge over misses in having more compounds with Tanimoto distances exceeding 0.55 to their nearest Training or AMS active as seen in Figure 3.5.

Table 3.7: Summary of the 68 screened compounds from the prioritized Enamine REAL compounds.

Selector	Count	Hits	Misses	Hit Rate (%)
RF-C Enamine REAL	68	31	37	45.59

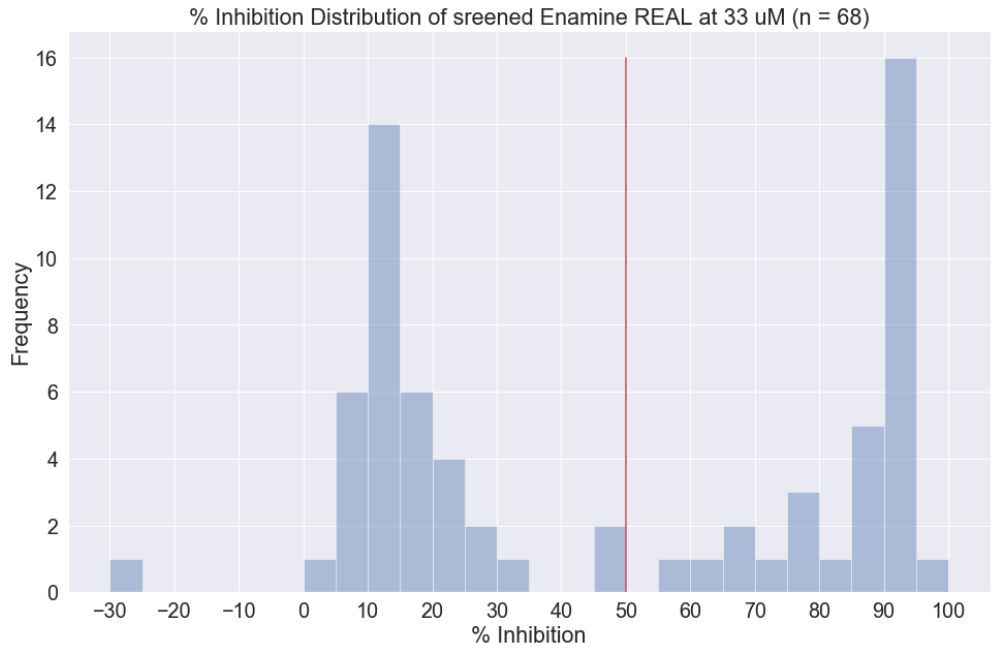


Figure 3.4: Median % inhibition distribution of the 68 screened Enamine REAL compounds at 33 µm along with the 50% hit cutoff defined as a red line. The median was taken from the four replicate screens.

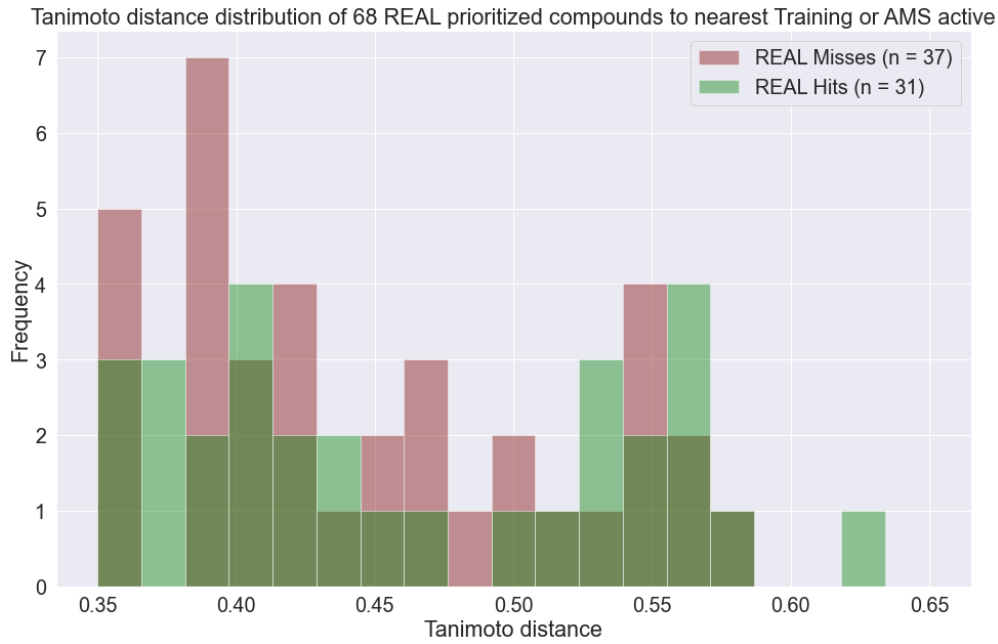


Figure 3.5: Distribution of the Tanimoto distance between the 68 screened Enamine REAL compounds and their nearest Training or AMS active.

Table 3.8: Illustration of the 31 hit Enamine REAL compounds and their nearest Training or AMS active. The similarity maps showcase the prospective compound with green highlights around atoms to indicate areas of similarity with their nearest training active. Larger radius highlights around the atom indicates larger degree of similarity. Similarly, red highlights indicate dissimilarity. These molecular graphs and similarity maps were generated using RDKit [1].

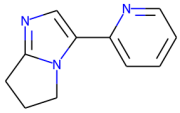
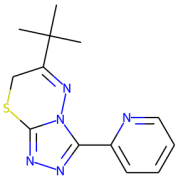
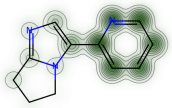
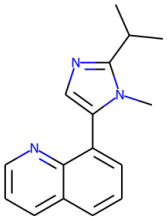
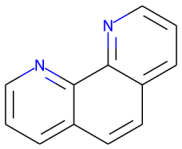
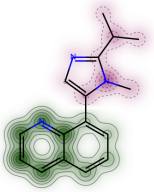
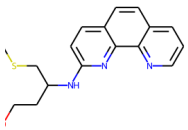
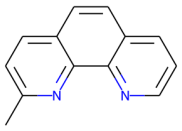
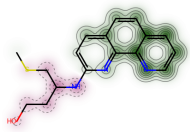
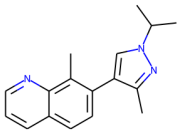
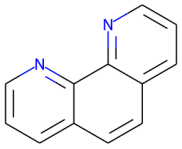
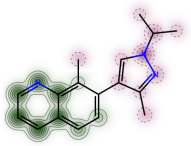
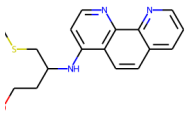
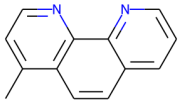
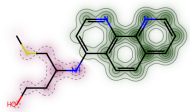
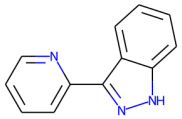
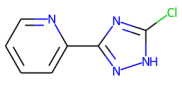
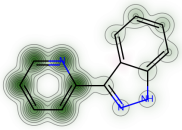
Prospective Compound	Nearest Training Active	Similarity Map	Tanimoto Distance
			Training: 0.62
			Training: 0.58
			Training: 0.57
			Training: 0.57
			Training: 0.56
			AMS: 0.56
Continued on next page			

Table 3.8 – Continued from previous page.

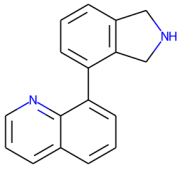
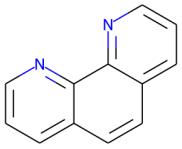
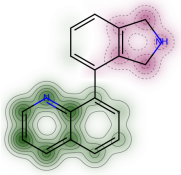
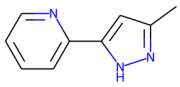
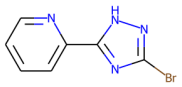
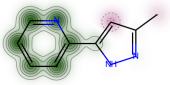
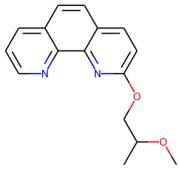
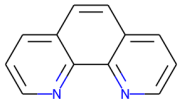
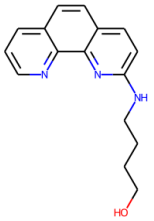
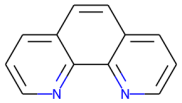
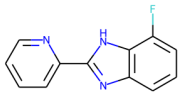
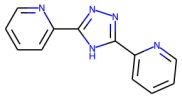
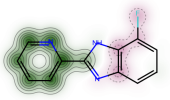
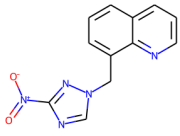
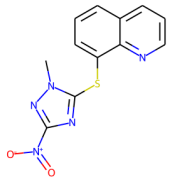
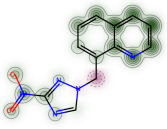
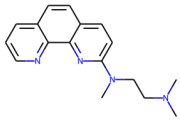
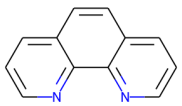
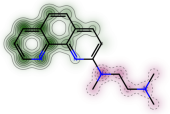
Prospective Compound	Nearest Training Active	Similarity Map	Tanimoto Distance
			Training: 0.55
			AMS: 0.54
			Training: 0.53
			Training: 0.53
			Training: 0.53
			Training: 0.51
			Training: 0.50
			Continued on next page

Table 3.8 – Continued from previous page.

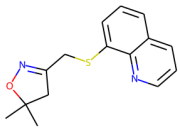
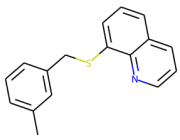
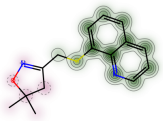
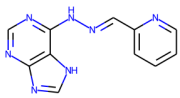
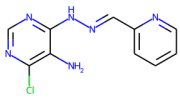
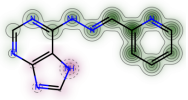
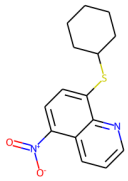
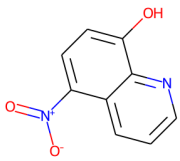
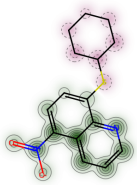
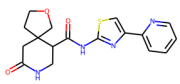
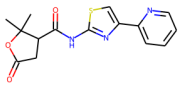
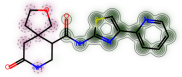
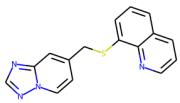
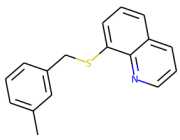
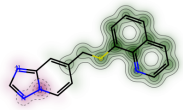
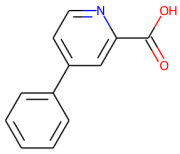
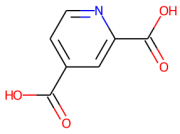
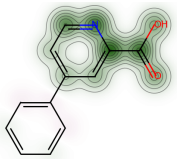
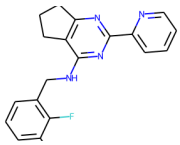
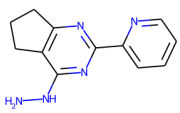
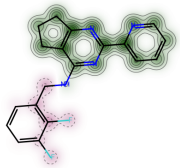
Prospective Compound	Nearest Training Active	Similarity Map	Tanimoto Distance
			Training: 0.47
			Training: 0.45
			Training: 0.44
			AMS: 0.44
			Training: 0.43
			Training: 0.41
			AMS: 0.41
Continued on next page			

Table 3.8 – Continued from previous page.

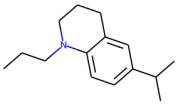
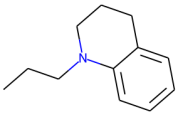
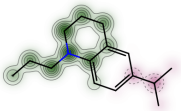
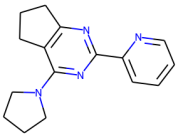
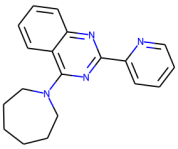
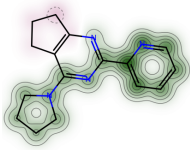
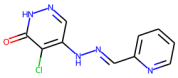
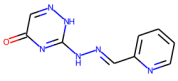
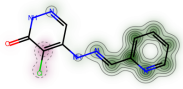
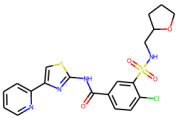
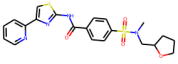
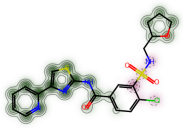
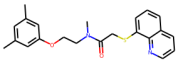
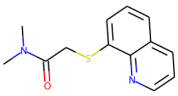
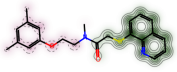
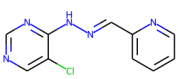
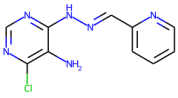
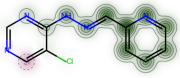
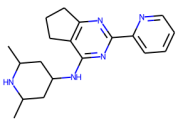
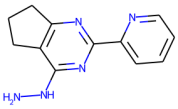
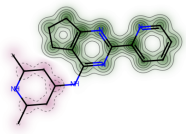
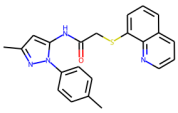
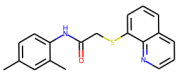
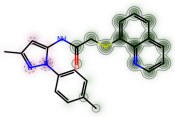
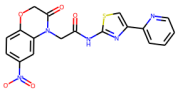
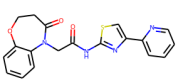
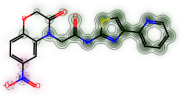
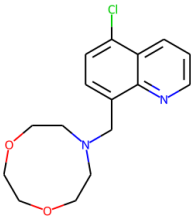
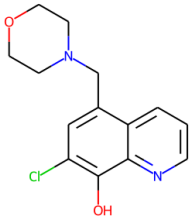
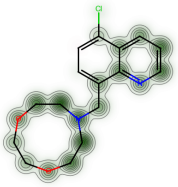
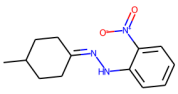
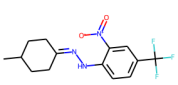
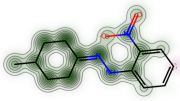
Prospective Compound	Nearest Training Active	Similarity Map	Tanimoto Distance
			Training: 0.41
			Training: 0.41
			Training: 0.41
			Training: 0.39
			Training: 0.38
			Training: 0.38
			AMS: 0.38
Continued on next page			

Table 3.8 – Continued from previous page.

Prospective Compound	Nearest Training Active	Similarity Map	Tanimoto Distance
			Training: 0.37
			AMS: 0.36
			Training: 0.35
			Training: 0.35

3.7 Conclusions and Future Work

This followup ORS case study showed that it is possible to achieve great results on massive prospective pools using supervised ligand-based models. From the AMS pool of 8,187,682 compounds, the RF-C and Similarity Baseline models selected 1024 compounds, 412 of which were hits. From the larger Enamine REAL pool of 1,077,562,987 compounds, the RF-C model along with diversity filters selected 68 compounds, 31 of which were hits. Given that the hit rate in the training set is 0.13%, these results vastly exceed the expectation of random sampling.

The hit determination was based on thresholding at specific values which can be contentious. However, the typical goal of these initial screens is to triage or filter down promising compounds. Followup screens would need to be conducted to determine the potency and safety of these compounds at various concentrations.

The manual correspondence between our team and commercial vendors in order to purchase the selected compounds was a tedious effort. These commercial libraries are dynamic and can change from week to week depending on the availability of materials, ease of delivery, and costs. An ideal customer platform would be one where these commercial libraries are accessible in real-time and maintained by the vendors themselves. A team making use of ligand-based models can tap into these real-time pools programmatically, thereby facilitating the integration between customer code and commercial pools. While there are libraries like ZINC15 [20] that list compounds from various vendors and is available online, it is not maintained by the vendors themselves. Although the independent maintainers have an update policy in place, the vendor catalogs in ZINC15 are not guaranteed to be up-to-date (see update policy `http://wiki.docking.org/index.php/ZINC15:update_policy`).

The diversity metrics used in this project were based on clusters that were defined using the Taylor-Butina algorithm [80, 81]. This algorithm is in turn highly dependent on the fingerprinting algorithm used, the cutoff distance, and the ordering in which the compounds are processed. Regardless, some form of a reasonable diversity metric is encouraged in order to assess the quality of the selection. For the AMS selection, no diversity filters were imposed, but many of the RF-C hits had larger Tanimoto distances than the Similarity Baseline hits as seen in Figure 3.3. Furthermore, the Enamine REAL selection by RF-C imposed diversity filters and found hits with Tanimoto distances at 0.62 (see Figure 3.5).

The success in this project, as well as that reported by related work that we covered in section 3.2, is more evidence that screening aided by virtual screening is preferable to that of a random high-throughput screen, particularly when dealing with massive prospective pools. The ORS studies done by others as shown in Table 3.1 provides size comparison with our work. We attribute the

success of our results to the size and quality of the training set consisting of 427,300 compounds and 554 hits. Furthermore, the prospective pool size used here and by Gorgulla et al. [78] are two ORS case studies that we are aware of with more than one billion compounds. Despite such massive prospective pools, both studies showcase favorable hit-finding results. A consideration for future work is to investigate the relationship between training set and the prospective pool in terms of sizes and quality.

The appeal of using supervised learning models is that many of them can be parallelized to score a large number of compounds independently. The ~ 1.077 billion Enamine REAL pool was scored in less than three days by distributing the scoring of compounds among 18 compute nodes. Compare this with the work done by Gorgulla et al. where they scored ~ 1.3 billion compounds in 4 weeks using 8000 compute nodes [78]. This is in main part due to the structure-based docking approach they took which is more computationally demanding than a ligand-based supervised learning approach. As noted in Hoffmann et al. [21], commercial libraries are expected to grow rapidly as synthesis techniques evolve. As a pertinent example, the ~ 1.077 billion compound Enamine REAL library used in this study was accessed in October 2019. As of November 2020, the library has grown to ~ 1.36 billion compounds. Thus, it is important that the computational costs of virtual screening methods are able to scale with the growing libraries.

— 4 —

Informer Set Methods – Cycle-based Importance Sampling Pipeline (CISP) and Matrix Completion (CustomMC)

Recall that in order to use ligand-based methods to prioritize compound selection, we need to have bioactivity data. Lacking such data, and depending on the availability of other types of information, one can opt towards other data acquisition methods. Diversity sampling, as explained in section 1.3, can be applied in any case. If target structure is available, structure-based methods can be used for prioritization. If bioactivity data for other targets that are related to the target of interest is available, an informer set method can construct an informative set of compounds for the initial screen. The hope is that this informer set of compounds can be used to score the activity of the remaining non-informer compounds. With the availability of large open source databases like PubChem and ChEMBL [5, 85], it is easier to search data on related targets. As will be discussed in section 4.2, the informer set problem is not formalized in the domain of drug discovery. The goal of this project is to formalize the problem in the context of virtual screening, propose and adapt solutions from related domains, and compare the performance of these solutions on real datasets. We also extend the work done by Zhang et al. and Yu et al. [2, 86] by applying informer set methods on significantly larger datasets. We plan to publish the results of this chapter in the future[1].

[1] Anticipated authors: Moayad Alnammi, Spencer S. Ericksen, Daniel Pimentel-Alarcón , Scott A. Wildman, and Anthony Gitter.

4.1 Informer Set Problem Overview

We have a new target of interest with no bioactivity data to exploit. Fortunately, we have a set of targets for which we do have bioactivity data with regards to a pool of compounds. We believe that these set of targets are *related* to the target of interest, and thus, can help us in prioritizing which compounds to screen from the pool. The *relatedness* of the targets is just an assumption, the nature of which typically depends on the informer set solution. To formalize the problem, we first define the parameters involved.

Suppose we have bioactivity data on m compounds tested on p targets. Denote the m compounds as X. Denote the p-target bioactivity matrix as $Y \in \mathbb{R}^{m \times p}$. Note that in Y, rows refer to compounds and columns refer to targets. Denote the **unknown/missing** column of the target of interest as $Y_{missing} \in \mathbb{R}^m$. Denote the known submatrix of related targets as $Y_{known} \in \mathbb{R}^{m \times (p-1)}$. Denote the relationship knowledge between the missing target and the known targets as W. Here W can be explicit like a $p \times p$ similarity matrix, or a general assumption like the missing target clusters into one of the known targets' clusters. Given these parameters, we formalize the informer set problem as follows:

Definition 4.1: Informer Set in Virtual Screening

Let InfAlg(X, Y, W, k) be an informer set algorithm that takes in X, Y, W, and k where k is a positive integer. InfAlg proceeds in two steps (an example is shown in Figure 4.1):

1. Given X, Y, W, and k, InfAlg selects a subset $Z \subset X$ of k informer compounds and obtains its corresponding labels $Y_{missing}[Z]$ (via in vitro screening).
2. Using the information gathered from the informer set labelling $Y_{missing}[Z]$, InfAlg predicts the bioactivity of the remaining non-informers of $Y_{missing}[X \setminus Z]$.

An optimal informer set algorithm is one that maximizes an evaluation function of interest: EVAL(InfAlg, X, Y, W, k). For example, maximizing the total number of hits in the selected informer compounds Z and the top 10% of predicted non-informers. This is formalized in the following optimization:

$$\text{InfAlg}^* = \underset{\text{InfAlg}}{\arg\max}\, \text{EVAL}(\text{InfAlg}, X, Y, W, k) \tag{4.1}$$

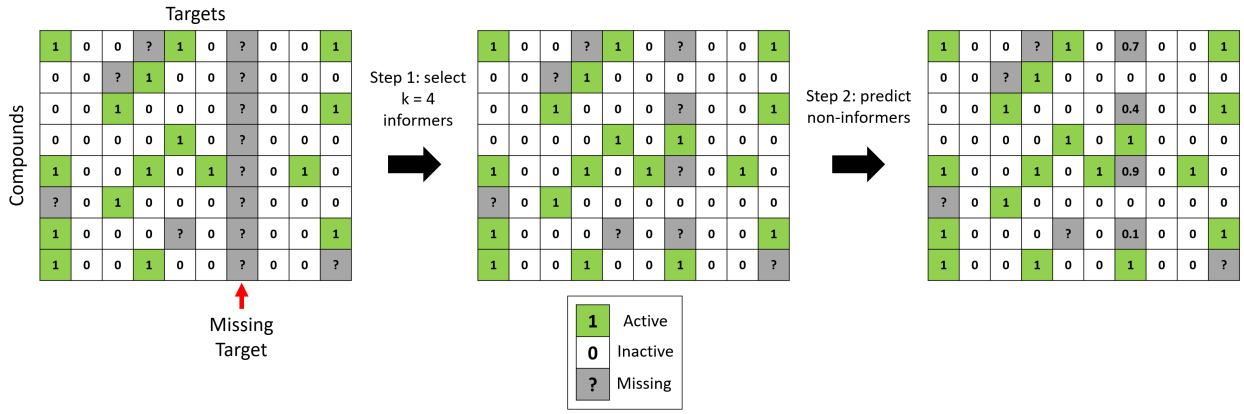


Figure 4.1: Illustrative example of the two steps an informer set algorithm performs: selecting the k informer compounds and obtaining their labels, then predicting the remaining non-informers. Notice that the multi-target bioactivity matrix can have missing values not just for the *missing* target.

In the above formulation, notice the required two step proceeding of the informer set algorithm, this is intentional. Otherwise, we would consider all possible algorithms that do or do not perform step 1 or even an iterative version of this. More importantly, in practice, the first step is followed by in vitro screening of the selected informer compounds. The screening information is then fed back to the informer set algorithm so that it can learn from this and score the remaining non-informer compounds. The optimization in Equation 4.1 is taken over all possible two-step solvers, and so finding the optimum is in general a combinatorial problem due to the selections involved. Nonetheless, we propose solutions that were adapted from domains with similar problems. Furthermore, as we will see in one of the datasets used in this project, the bioactivity submatrix Y_{known} can also have missing values as illustrated in Figure 4.1. Some of the proposed informer set algorithms can deal with missing values in the known submatrix, while others operate under the notion that it is complete. In section 4.3 we discuss various informer set algorithms used in this project based on the side information they incorporate in order to perform steps 1 and 2.

4.2 Related Work

We first focus on related work in the field of drug discovery. In 2016, Paricharak et al. [87] proposed an active learning guided approach for determining an informer set of compounds. Their definition

of an informer set is different than what is defined in this project, as their method requires an initial set of active and inactive labels (1000 in their study). Once the initial dataset is defined, the method would then use active learning to select the next set of uncertain compounds, adding it to the training set of informers, and iteratively repeat this process until no uncertain compounds are left.

Methods that use compound and protein features to predict properties (like bioactivity) are known as proteochemometric modelling (PCM) [88]. The focus in this domain has been in developing better descriptors for compounds, proteins, and compound-protein interactions [88, 89]. As an example, in 2017, Cichonska et al. [11] proposed a kernel based regularized least-squares method encoding the compound-compound and protein-protein similarities of the compounds and proteins in the bioactivity matrix. The model would then predict the bioactivity of a compound-target pair. Furthermore, they identified four scenarios related to multi-target bioactivity datasets (see Figure 4.2). Their focus was on solving the first two scenarios: Bioactivity Imputation and New Drug. The New Target scenario is relevant to our problem of informer sets. Also related to proteochemometric modelling, in 2020, Cichonska et al. analyzed the results of the IDG-DREAM drug-kinase binding prediction challenge [90]. The challenge focused on compound-target potency predictions for kinase targets. Participants were evaluated on two unpublished compound-target bioactivity matrices in two rounds. There were no restrictions on the type of data or model that can be used. The top performers included deep learning, graph convolutional networks, gradient boosting decision trees, ridge regression, and kernel learning. Some of these models made use of target features like amino acid sequences, while others relied only on compound features like fingerprints or molecular graphs.

In 2019, Zhang et al. [2] proposed three informer set methods: Regression Selection (RS), Coding Selection (CS), and Adaptive Selection (AS). They studied these methods on two kinase datasets curated from the Published Kinase Inhibitor Sets (PKIS): PKIS1 (366 compounds and 224 targets) [3] and PKIS2 [91] (415 compounds and 406 targets). Retrospective cross-validation performance was reported for PKIS1 using leave-one-target-out. Prospective evaluation was conducted on two kinase targets not part of PKIS1 and PKIS2: Mycobacterium tuberculosis PKnB

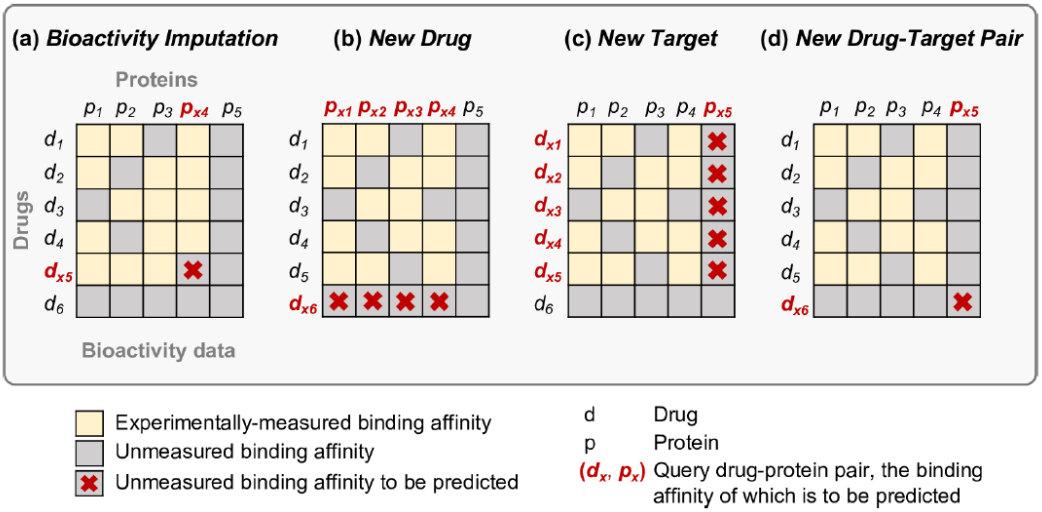


Figure 4.2: Illustration of the four scenarios for multi-target bioactivity matrix datasets indicating the desired predictions. This figure is from Cichonska et al. [11] under the CC BY 4.0 license (`http://creativecommons.org/licenses/by/4.0/`). Their paper proposed a method for solving the first two scenarios: Bioactivity Imputation and New Drug. The focus of informer set methods is the New Target scenario.

and Epstein-Barr BGLF4. Using the PKIS1 dataset, the goal was to develop solutions that would: generate an informer set of size 16 for each of the two held out targets, generate labels for those 16 via in vitro screening, and then score and rank the remaining non-informer compounds. The same was done using the PKIS2 dataset. Of interest was that the three proposed solutions operated only on the bioactivity matrix; i.e. no compound information or features were used. Using only bioactivity information was a constraint imposed on the newly developed methods to see how such a method would compare with a baseline that uses both compound and bioactivity features. An important assumption made by the proposed methods was that the targets are related in the sense that they can be clustered by their bioactivity profiles, and within each dense cluster, there is a good informer set of compounds for targets within that cluster. Their methods were compared against six baselines that are based on nearest neighbor techniques using the compound fingerprint features. The results favor RS, CS, and AS over the baseline methods.

In 2020, Yu et al. [86] introduced an informer set method called BOISE as a followup to the work by Zhang et al. [2]. The BOISE method incorporates much from statistical decision theory and formulates the problem as a risk minimization of multi-Bernoulli trials. The details of the method

are quite involved, however, of note is that this method can be applied to bioactivity matrices with missing data. They compare BOISE against the methods in Zhang et al. on two datasets PKIS1 and GSDC1 [2, 3, 92]. As aforementioned, PKIS1 consists of 366 compounds and 224 targets. GSDC1 consists of 304 drugs and 987 targets, but with 15.1% of the matrix missing. BOISE had the best performance in a leave-one-target-out fashion. Finally, the main drawback, as the authors point out, is the high computational cost of BOISE. As an example, running BOISE on one PKIS1 target to select 16 informers took 300 CPU hours. The authors seek to address this in future work in order to scale BOISE to larger datasets.

Now we shift our focus to related works in machine learning. A similar concept to an informer set in machine learning can be found in machine teaching. In an overview by Zhu et al. [93], they define machine teaching as an optimization problem involving a teacher and a student learner. The problem can be formulated in many ways, but in general, the goal of the teacher is to construct an optimal training set D for the learner. This gives rise to the bilevel optimization as shown in Zhu et al.:

$$\begin{aligned} \min_{D,\hat{\theta}} \quad & \text{TeachingRisk}(\hat{\theta}) + \eta \text{TeachingCost}(D) \\ \text{s.t.} \quad & \hat{\theta} = \text{MachineLearning}(D) \end{aligned} \tag{4.2}$$

Where $\hat{\theta}$ denotes the model learned by the learner given the training set D, TeachingRisk($\hat{\theta}$) quantifies the risk of the learner's model, η is a regularization constant, and TeachingCost(D) quantifies the cost of the teacher's selected training set D. Machine teaching encompasses a wide range of problems. Of close relation to informer sets is machine teaching applied in education. As discussed in Zhu et al., the goal in this setting is to construct an *informative* training set for n students. For example, to optimize the training set for the average student, we have the following optimization

adapted from Zhu et al.:

$$\min_{D,\{\hat{\theta}_1.\hat{\theta}_2,...,\hat{\theta}_n\}} \quad \frac{1}{n}\sum_{i=1}^{n} \text{TeachingRisk}(\hat{\theta}_i) + \eta\text{TeachingCost}(D) \tag{4.3}$$

$$\text{s.t. } \hat{\theta}_i = \text{MachineLearning}_i(D) \text{ for } i = 1, 2, ..., n$$

In the context of the informer set problem, the targets are the students, the training set is the informer set of compounds, the average student is the held-out/missing target, and we add an additional constraint of $D = k$ for the number of informer compounds. Under this setting, the informer set algorithm would be one that employs a supervised learning model that selects informers in order to minimize the teaching risk in this optimization. Finally, as mentioned by Zhu et al., the optimization is in general NP-hard. Efficient solutions rely on submodularity assumptions [94], however showing that the informer set problem is submodular is difficult and will not be explored in this chapter.

Another concept from machine learning that is related is that of core-set selection [95]. Given a training set D and a supervised learning model A, a core-set is a subset $Z \subset D$ of size k that minimizes the generalization risk. Put simply, can we get away with a smaller training set, but still achieve acceptable performance? We will discuss this concept in more detail when we introduce the MTNNSupervised informer set solution.

The **significance** of this project is as follows:

- We hypothesize that using a complex model and incorporating side information like compound features and target relatedness in the form of similarity weights to the missing target can further improve the results. This is in contrast to the methods introduced in Zhang et al. [2] which were constrained to only the bioactivity data.

- We adapt standard matrix completion and imputation algorithms to the informer set problem in a two-step manner. The first step uses a simple heuristic to select k informers. The second step uses standard techniques like matrix completion or imputation to score the remaining

non-informers. This allows us to easily add these standard methods to the pool of solutions under consideration.

- We analyze performance of a wide range of solution avenues. The methods used in this project are drawn from matrix completion, supervised learning core-sets, clustering, and imputation solutions. Some of these solutions make use of only the multi-target bioactivity matrix, while others incorporate side information like compound features and target-target similarity weights. In doing so, we want to draw conclusions on the helpfulness of such side information.

- The methods proposed by Zhang et al. [2] do not handle missing values in the known bioactivity matrix. In this project, there are two proposed solutions CISP and CustomMC that can handle this situation. The BOISE method proposed by Yu et al. [86] handles missing data, but does not scale well to larger datasets. In this project, several proposed methods scale easily. Thus, a proposed solution should be able to handle missing data and scale to larger datasets.

4.3 Adapted Existing Methods

Overview

Here we describe various informer set algorithms that were adapted from known matrix completion, supervised learning, and imputation algorithms. The algorithms differ in the type of side information they use for the two step procedure of: selecting k informer compounds and obtaining their labels, then scoring the remaining non-informer compounds. First, we briefly list the type of side information available:

- **Compound Features:** These are compound descriptors like SMILES [47], graphs, fingerprints, physicochemical properties, etc. Some of these were described in section 1.5.

- **Target Features:** These are target descriptors like 3D structures or amino acid sequences for proteins.

- **Compound-Compound Matrix:** This is the compound by compound similarity or weight matrix that serves as a proxy for compound *relatedness*. For example, the Tanimoto distance between the fingerprints of two compounds can be used as a proxy for structural similarity.

- **Target-Target Matrix:** This is the target by target similarity or weight matrix that serves as a proxy for target *relatedness*. If target protein sequences are available, then one way to compute target-target similarity is via sequence alignment scores [96].

- **Compound-Target Bioactivity Matrix:** This is the main compound-target matrix of bioactivity labels. The informer set algorithm is given this matrix and proceeds in two steps: select k informer compounds whose labels are obtained for the missing target, then score the remaining non-informer compounds.

Recall that an informer set algorithm performs two steps as seen in Figure 4.1. Many of the adapted algorithms cannot perform the first step due to stringent assumptions. To get around this problem, we use a helper heuristic for the first step. We define two **simple heuristics** for selecting k informer compounds (an example is shown in Figure 4.3):

- **Most active across known targets (MAct):** This heuristic sums the number of times each compound is active across the known targets. The largest k compound sums are selected as the informers. The hope is that a compound that is active across many targets will also be active on the missing target. This set might not be the most *informative* set as a followup model may get a better understanding of the missing target's labelling with a good mix of actives and inactives. However, it is unlikely that we gather many actives using this procedure when we have many targets. Nonetheless, it is a simple heuristic that will hopefully pick up some actives.

- **Weighted average of most similar known targets (SeqSim):** This heuristic uses the target-target similarity matrix to compute a weighted average of the top q most similar known targets to the missing target. For this project, we use the pairwise sequence alignment scores between the known protein targets and the missing protein target [96]. Specifically, let the similarity scores of the missing target to a known target i be w_i, let the set of indices corresponding to the top q scores be Q, then the missing target column is set as:

$$Y_{missing} = \frac{\sum\limits_{i \in Q} w_i Y_i}{\sum\limits_{i \in Q} w_i} \tag{4.4}$$

  Figure 4.3 shows an example of this heuristic with $q = 2$. Notice how the missing target is set to the weighted average of the first and fourth known target columns since they have the two largest similarity scores. The top k prediction scores of the missing target are selected as the informer compounds. Now for step two, we can either continue using the weighted average scores as the non-informer scores, or feed the k informer compounds to a followup method. This heuristic assumes that the target-target similarity scores reflect how similar two target columns are. Protein sequence alignment similarity scores have been used in models to help find binding ligands [97, 98, 99].

In total, there are 21 competing methods. 9 of these methods are from Zhang et al. [2], 5 are from the fancyimpute Python package [100], and 7 are custom implemented methods. Table 4.1 summarizes the name, solution type, and type of side information used by each informer set method. We discuss each of these methods in the next sections. The project's Github repository can be consulted for any ambiguity about the implementation details: `https://github.com/Malnammi/informer-set-drug-discovery`.

Low-Rank Matrix Completion Methods

Matrix completion is the problem of filling in the missing cells of a matrix. Thus, it is relevant to our informer set problem. Proposed solutions have been successful and applied in commercial

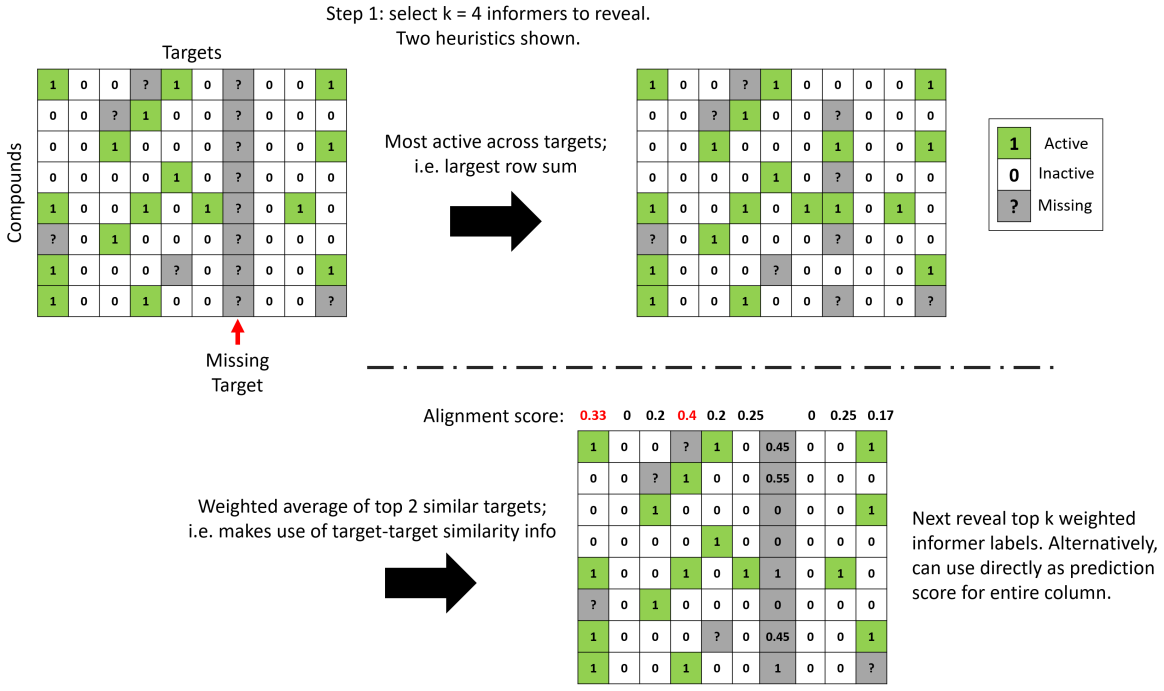


Figure 4.3: Illustrative example of using two simple heuristics to perform the first step of selecting k informer compounds and obtaining their labels. The two heuristics used in this project are: most active across known targets (largest row sum) and weighted average of top q most similar targets. The target-target similarity weights are the pairwise sequence alignment scores. Ties are broken arbitrarily; i.e. we select the first in the top sorted implementation.

products, however they require assumptions on the matrix that are violated in our case. These solutions typically assume that the matrix is low-rank. Given that M is the matrix to complete, these solutions attempt to find a low-rank matrix that agrees with the observed entries Ω. This is often expressed in the following optimization (or a variation of this) [101]:

$$\begin{aligned} \min_{X} \quad & \text{rank}(X) \\ \text{subject to} \quad & X_{ij} = M_{ij} \;\; \forall (i,j) \in \Omega \end{aligned} \tag{4.5}$$

Another major assumption is that every row and every column in the matrix must have at least one visible entry. In other words, no missing row or column, which is violated since we have an entire target column missing. If an entire column is missing, then this missing column can be arbitrarily set [101]. To see why, consider the singular value decomposition of an $m \times n$ matrix $M = U\Sigma V^T$ into

Table 4.1: Summary of the informer set methods in this project and the type of side information they use. ✓denotes required information, whereas [✓] denotes optional usage. Solutions adapted from the fancyimpute Python package are prefixed with FCI.

Method Name	Cpd-Target Bioactivity Matrix	Cpd Features	Target-Target Matrix	Cpd-Cpd Matrix	Requires first step helper heuristic?	Solution Type
FCINuclearNormMin	✓		[✓]		✓	Matrix Completion
FCISoftImpute	✓		[✓]		✓	Matrix Completion
FCIMatrixFactor	✓		[✓]		✓	Matrix Factorization
SimpleMC	✓		[✓]		✓	Matrix Completion
CustomMC	✓					Matrix Completion
RFSupervised	✓	✓			✓	Supervised Learning
MTNNSupervised	✓	✓	✓			Supervised Learning
CISP	✓	✓	[✓]			Supervised Learning
Adaptive Selection	✓					Target Clustering
Coding Selection	✓					Target Clustering
Regression Selection	✓					Target Clustering
TargetClusteringRF	✓					Target Clustering
BC_l	✓	✓		✓		Nearest Neighbor
BC_s	✓	✓		✓		Nearest Neighbor
BC_w	✓	✓		✓		Nearest Neighbor
BF_l	✓	✓		✓		Nearest Neighbor
BF_s	✓	✓		✓		Nearest Neighbor
BF_w	✓	✓		✓		Nearest Neighbor
FCIKNN	✓		[✓]		✓	Nearest Neighbor
PairwiseSeqSimilarity	✓		✓			Nearest Neighbor
FCIIterativeImputer	✓		[✓]		✓	Imputation

left-singular vectors $U \in \mathbb{R}^{m \times m}$, right-singular vectors $V \in \mathbb{R}^{n \times n}$, and singular values $\Sigma \in \mathbb{R}^{m \times n}$. In Figure 4.4, we show two examples of similar decomposition for a matrix M. Notice that the only decomposition difference is in the first column of V^T, resulting in a different completion for the first column of M. Thus, a missing column can be filled in randomly without altering the other columns. Another way to look at it is that the columns of M are a linear combination of the columns of U. That is, every column i of M can be written as $M_i = \sum_{i=1}^{m} c_i U_i$ where c_i are the linear coefficients. So it is easy to see how the missing column can have arbitrary coefficients and not affect the other columns.

$$M = U\Sigma V^T$$

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.41 & 0.5 & -0.57 & -0.5 \\ 0.57 & -0.5 & 0.41 & -0.5 \\ 0.41 & -0.5 & -0.57 & 0.5 \\ 0.57 & 0.5 & 0.41 & 0.5 \end{bmatrix} \begin{bmatrix} 2.68 & 0 & 0 & 0 \\ 0 & 1.41 & 0 & 0 \\ 0 & 0 & 0.92 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.43 & 0.74 & 0.37 & 0.37 \\ 0 & 0 & -0.71 & 0.71 \\ 0.9 & -0.35 & -0.18 & -0.18 \\ 0 & 0.58 & -0.58 & -0.58 \end{bmatrix}$$

$$\begin{bmatrix} 5 & 1 & 0 & 1 \\ 6 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 11 & 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.41 & 0.5 & -0.57 & -0.5 \\ 0.57 & -0.5 & 0.41 & -0.5 \\ 0.41 & -0.5 & -0.57 & 0.5 \\ 0.57 & 0.5 & 0.41 & 0.5 \end{bmatrix} \begin{bmatrix} 2.68 & 0 & 0 & 0 \\ 0 & 1.41 & 0 & 0 \\ 0 & 0 & 0.92 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 4.53 & 0.74 & 0.37 & 0.37 \\ 3.26 & 0 & -0.71 & 0.71 \\ 3.56 & -0.35 & -0.18 & -0.18 \\ 2.11 & 0.58 & -0.58 & -0.58 \end{bmatrix}$$

Figure 4.4: Singular value decomposition of a matrix M where only the first column of V^T is changed, resulting in a different first column for M. This explains how if the first column of M is missing, then it can be completed arbitrarily without altering the other columns.

One way around this violation is to assume restrictions on the missing column. Yang et al. [102] proposed a low-rank completion on missing columns or rows but with the assumption that the matrix has local structure like in pixel images. In pixel images, the position of a column or row matters in forming structural relationships between adjacent pixels. However, in the informer set problem, the columns in the matrix are targets, and the position of the target column (e.g. first or last column) is arbitrary. Thus, we must place other target relationship assumptions that are not guaranteed to hold. For example, we can assume that the missing target lives in the subspace of the known targets, and so, can be completed by using the SVD decomposition of the known targets submatrix.

Another way around the missing column violation is to reveal some of the cells of the missing target using a first step helper method (e.g. Figure 4.3), then subsequently complete the matrix. This is convenient, since we define informer set algorithms as performing two steps: select k informer compounds to reveal their missing target labels, then score the remaining non-informers. For example, we can use a simple informer set selector like selecting the top k most active compounds across known targets. After revealing the missing target labels for these k informers, we can run a low-rank matrix completion solution to fill in the remaining non-informers.

Now we list the low-rank completion methods used from the fancyimpute Python package [100]. We give a brief overview; the details of these methods can be found in the corresponding references. All methods require a first step helper heuristic and only make use of the target-compound matrix.

- **FCINuclearNormMinimization** [101, 100]: Convex relaxation via nuclear norm minimization as proposed by Candes et al [101]. The nuclear norm is the sum of singular values.
- **FCISoftImpute** [103, 104, 100]: An iterative soft thresholding solution to the nuclear norm convex relaxation. It is efficient for large matrices with 10^5 to 10^6 rows/columns and 10^6 to 10^8 entries [103].

Matrix Factorization Methods

Matrix factorization is famously used in recommender systems [105]. It assumes that the matrix $M \in \mathbb{R}^{m \times n}$ factorizes into two matrices $M = AB$ where $A \in \mathbb{R}^{m \times r}$ and $B \in \mathbb{R}^{r \times n}$. r is known as the latent dimension. We use the fancyimpute implementation and refer to it as **FCIMatrixFactorization** [100].

Other Matrix Completion Methods

We devised two matrix completion methods that make use of singular value decomposition (SVD) concepts: **SimpleMC** and **CustomMC**. But first, let us define the matrix index slicing notation that will be used. This slicing notation is similar to the one used in the NumPy Python package. Given a matrix $M \in \mathbb{R}^{10 \times 10}$ and a set of indices $J = \{0, 4, 5\}$, the submatrix $M[J, :] \in \mathbb{R}^{3 \times 10}$ is the matrix consisting of the rows 0, 4, and 5 of M. Similarly, the submatrix $M[:, J] \in \mathbb{R}^{10 \times 3}$ is the matrix consisting of the columns 0, 4, and 5 of M. The indices before the comma indicate which row indices to keep, and the indices after the comma indicate which column indices to keep.

SimpleMC assumes that the missing target belongs to the column space of the known target matrix. Consider the binary valued bioactivity matrix, let the known targets submatrix be $Y_{known} \in \{0, 1\}^{m \times (p-1)}$, and let the missing target column be $Y_{missing} \in \{0, 1\}^m$. Compute the

SVD decomposition of the known submatrix: $Y_{known} = U\Sigma V^T$. Note that each column of Y_{known} is a linear combination of the columns U. The linear coefficients for each target column are given in $X = (\Sigma V^T)$ as follows:

$$Y_{known}[:, i] = UX[:, i] \text{ for each column } i \tag{4.6}$$

We assume that $Y_{missing}$ lives in the subspace spanned by the columns of U, and so, we want to find its linear coefficients $x \in \mathbb{R}^m$ (there is no guarantee for this assumption). To do this, we need to solve $Ux = Y_{missing}$, however the labels of $Y_{missing}$ are unknown. This requires a second assumption in order to proceed. SimpleMC assumes that there exists a set of k informer compounds, whose indices are identified by the set $InfSet$, that are enough to reconstruct Y_{known}. Specifically, consider the submatrix of known targets consisting of just the informer compounds: $Y_{known}[InfSet, :]$. Then, if we solved for $W \in \mathbb{R}^{m\times(p-1)}$ in:

$$Y_{known}[InfSet, :] = U[InfSet, :]W \tag{4.7}$$

This W coefficient matrix can also be used to approximate the full known matrix: $Y_{known} \approx UW$. In other words, the k informer compounds are enough to find the coefficients. To further limit the degrees of freedom in the solution fit, we truncate the SVD decomposition to the rank r at which a threshold percentage of the variance is explained; i.e. we use $U[InfSet, : r]$. Based on this, given the $InfSet$, we are interested in solving the following for $x \in \mathbb{R}^m$:

$$Y_{missing}[InfSet] = U[InfSet, : r]x \tag{4.8}$$

We use a least-squares solution to find x, and subsequently score the remaining non-informer labels as:

$$Y_{missing} \approx U[:, : r]x \tag{4.9}$$

The only issue left is how to select the k informer compounds. For SimpleMC we use a simple heuristic helper like those shown in Figure 4.3. To summarize, the algorithm proceeds in the following steps:

1. Compute the SVD decomposition of the known submatrix: $Y_{known} = U\Sigma V^T$.
2. Compute the variance explained by *each* singular value σ_i:

$$\text{var}(\sigma_i) = \frac{\sigma_i^2}{\sum_j \sigma_j^2} \tag{4.10}$$

3. Compute the rank r at which a threshold percentage of the variance is explained:

$$\sum_{i=1}^{r} \text{var}(\sigma_i) < p \text{ where } p \in (0, 1) \tag{4.11}$$

4. Truncate U down to the r largest singular values: $U[:, :r]$.
5. Select k informer compounds using a first step heuristic helper method.
6. Use the least-squares method to solve $Y_{missing}[InfSet] = U[InfSet, :r]x$ for x.
7. Score the remaining non-informers of the missing target as: $\hat{Y}_{missing} = U[:, :r]x$

CustomMC is another custom completion solution that approaches from the row space rather than the column space. It will be described in section 4.4.

Supervised Learning Methods

In this project, some methods employ machine learning classification models to learn the missing target labelling given the compound features. However, this requires training data labels, and for that we use the k informer compound labels of the missing target. This in turn requires a procedure to select the k informer compounds. Once the k informer missing target labels are obtained, a

classifier is trained on these labels and used to predict missing target scores on the remaining non-informer compounds. **RFSupervised** uses a first step heuristic method to select the k informers (see Figure 4.3). The k informer missing target labels are obtained and used to train a random forest model with 1024-bit RDKit Morgan fingerprint features as input [1, 55]. The random forest uses the scikit-learn python package [106].

A more complex approach can be used for selecting the k informers. It is based on the assumption that across all p targets (including the missing target), there exists a set of k informer compounds such that training a classifier on these k informers for any of the targets, yields a *good* classifier *on average*. That is, we want to attain a good classifier for any of the p target learning tasks using the same k informers as the training set. Note that maximizing performance on one target may yield a set of k informers that is different than when trying to maximize on another target. So the intention is to *try* to maximize performance across all targets simultaneously via pooling of information. These k informers can then be used to train a model on the missing target informer labels and predict on the non-informers. This is the basic idea behind the proposed Cycle-based Importance Sampling Pipeline **(CISP)** algorithm which will be described later in section 4.5.

MTNNSupervised is a method that also uses this assumption. Its approach is based on the concept of a core-set in machine learning [107, 108, 95]. A core-set of a dataset $S = \{(x_0, y_0)$, (x_1, y_1), ..., $(x_m, y_m)\}$, where x_i are features and y_i are labels, is a subset $Z \subset S$ of size $k < m$ such that a classifier h trained on either sets yields minimal performance difference. In other words, the subset Z is representative of the entire dataset S and is good enough for learning the task. This can be concretely defined in terms of the generalization error as follows (adapted from Coleman et al. [95]):

$$Z^* = \underset{Z \subset S, |Z|=k}{\arg\min} \left\| \mathbb{E}_{(x,y)}\Big[\ell(h(x;S),y)\Big] - \mathbb{E}_{(x,y)}\Big[\ell(h(x;Z),y)\Big] \right\|_2 \tag{4.12}$$

Where ℓ is the loss of interest and $h(x;A)$ is the prediction of classifier h on instance x when trained on subset A. MTNNSupervised attempts to find a core-set of size k among all known targets, and

uses this core-set to learn the missing target. To do this it uses a multi-task neural network (MTNN) that takes in compound fingerprint features and predicts the labels of all known tasks (see example in Figure 4.5). Informer compounds are selected as the top k most uncertain compounds averaged over all the target predictions. The assumption here is similar to that in active learning, i.e. the most uncertain compounds lie near the decision boundary [109, 95, 110]. Specifically, the following are the steps:

1. Let there be p targets and m compounds. Let the compound featurization of the m compounds be X. Let the known target submatrix be $Y_{known} \in \{0,1\}^{m \times (p-1)}$. Let the missing target column be $Y_{missing} \in \{0,1\}^{m}$.

2. Construct a MTNN with a defined architecture that takes in X as input and Y_{known} as output. Specify the output layer activation units as the sigmoid function. Train for a number of iterations using stochastic gradient descent. Use the binary cross-entropy loss for all target outputs. If the pairwise sequence similarity weights are available, use them to weigh the target losses. In particular, we weight the target losses by their similarity to the missing target. The Keras python package is used for implementation [111].

3. After training, compute final compound predictions on the known targets by doing a forward pass on the trained MTNN with X as input. Compute the mean prediction of each compound by aggregating across the targets. From the mean compound predictions, compute the mean compound uncertainty using the sampling uncertainty formula from active learning (see Equation 5.6).

4. Select the k compounds with the largest mean uncertainty as the informer set. Obtain the corresponding k labels in the missing target $Y_{missing}$.

5. Train a new single-task neural network on the missing target with the same architecture as the MTNN. To clarify, this single-task network is trained from scratch and does not reuse the weights from the MTNN. The k informer compound labels for the missing target is used as the training set. After training, compute missing target predictions on the non-informers.

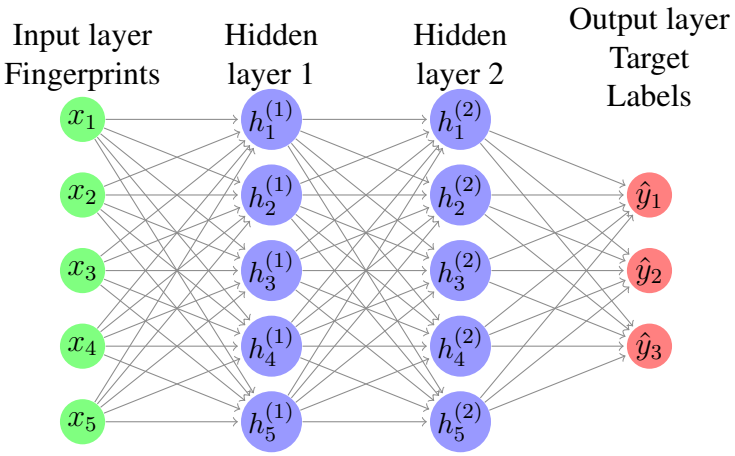


Figure 4.5: Multi-task neural network (MTNN) example with 5-bit fingerprint feature inputs and 3 target label outputs.

Target Column Clustering Methods

Target clustering methods cluster the target bioactivity column vectors into N neighborhoods based on their distances. They assume that the missing target column vector clusters into the N neighborhood clusters; i.e. its distance is close. However, since the missing target is unknown, these methods use the k informer compounds to make a decision on the missing target's cluster assignment. Zhang et al. [2] introduces three target clustering methods for which we provide a brief overview:

- **Regression Selection:** First, it clusters the known target columns Y_{known} using K-means [112] into N clusters. Second, it trains a multinomial logistic regression model using the target's column vector as input and the one-hot encoding of its cluster membership as output. The model learns the weight coefficients being associated with each compound label, and thus, aims to find the k highest weighted compounds. A penalty is applied to promote sparsity (i.e. a few highly weighted compounds). The details of the iterative process can be found in Zhang et al. [2]. After finding the k informers, another multinomial logistic regression model for predicting the cluster membership is trained, but this time only the k informers are used as input features. Given the missing target's k informer labels, the model is used to predict its

probability membership to all N clusters. Finally, the missing target's non-informer labels are set to the probability-weighted average of the N cluster means. It is interesting to note that this method operates on the % inhibition continuous scores rather than on the binary activity.

- **Coding Selection:** This method proceeds by first defining an informer set A and space partition π (notation used in Zhang et al. [2]). π partitions the space of possible informer set vector labelling into N partitions corresponding to clusters. Thus, the clustering of a target is based only on the labels of its informer set compounds. Given an informer set A and partition π, it assigns a score using an objective function that evaluates how well cluster targets minimize their non-informer label disagreement; i.e. we want targets in the same cluster to have similar non-informer labels. So in short, the informer set A and partition π define the cluster assignment of targets. Finding a minimum A and π for the objective function is combinatorial, and so, the approximate solution uses an adapted Monte Carlo search. Once A and π are found using the known targets, the missing target's informer labels are obtained and clustered accordingly. Foregoing details, the missing target's non-informers are scored as the mean of cluster members.

- **Adaptive Selection:** This method adapts components from Regression Selection and Coding Selection. Suppose the size of the informer set is k. Similar to regression selection, the method starts by clustering the known target columns Y_{known} using K-means [112] followed by multinomial logistic regression model for predicting cluster assignment. However, it aims to find the $q < k$ highest weighted compounds. These q compounds are called the base informer set. Subsequently, it finds the remaining $k - q$ informers via an iterative selection process. Each iteration selects the next informer such that it minimizes the distance between non-informer labels and the mean of informer labels. The hope is that the q base informers provide predictive clustering, and the k full informers provide prediction on non-informers. With the k informers, the missing target informers are obtained and the scoring

of non-informers proceeds similarity to coding selection.

TargetClusteringRF is a custom method adapted from regression selection [2]. After clustering the known targets using K-means into N clusters, instead of multinomial logistic regression, a random forest model is used to predict cluster assignment given a target column. The next step is to identify the k informer compounds that are predictive of this cluster assignment when given a target's column vector. To this end, we use the feature importance of the random forest model available in the scikit-learn implementation [106]. According to scikit-learn, the importance of a feature is a value indicating how well that feature helps in increasing the quality of a split. Note that features in this case correspond to compound labels; i.e. we want the k most important compounds in the random forest. These k informer compounds are then used to build another random forest that takes in those k compound labels as input and predicts the cluster assignment. Subsequently, the missing target's informer labels are revealed and passed into the random forest to predict its cluster assignment. The remaining non-informers are scored as the weighted average of cluster members weighted by their informer label distances. Thus, the main difference between Regression Selection and TargetClusteringRF is the choice of cluster predictor. However, this model choice is noteworthy since multinomial logistic regression is linear whereas the random forest is non-linear.

Nearest Neighbor and Imputation Methods

Zhang et al. [2] introduced six baseline methods that in one form or another make use of nearest neighbor calculations. First, they define two ways by which the k informers are selected: **chemometric selection (BC)** and **frequent hitters selection (BF)**. BC clusters the compounds using fingerprint features into k clusters, and then selects one representative compound from each of the k clusters to serve as the informer set. BF selects the k compounds that are the most active across the known targets; similar to the first step heuristic in Figure 4.3. Compounds are featurized as 1024-bit RDkit Morgan fingerprints [1, 55]. Secondly, they define three followup procedures for scoring the missing target's non-informers:

- **Simple Expansion (s):** Scores the non-informers as the Tanimoto similarity to their closest informer active.
- **Loop Expansion (l):** Loops in a round-robin fashion on the informer actives. For each iteration's currently selected informer active, it finds and assigns the nearest unscored non-informer the corresponding Tanimoto similarity. The looping continues until all non-informers are assigned a similarity score.
- **Weighted Expansion (w):** Scores the non-informers as the weighted average of k informer labels. The weights used for each non-informer is the Tanimoto similarity to the k informers.

There are two methods to select the k informers and three methods to score the non-informers. Thus, in total there are six baseline methods: $\mathbf{BC}_s$, $\mathbf{BC}_l$, $\mathbf{BC}_w$, $\mathbf{BF}_s$, $\mathbf{BF}_l$, and $\mathbf{BF}_w$.

FCIKNN is a simple K-nearest neighbor approach from the fancyimpute package [100]. First, it uses a first step heuristic to select the k informers. Second, it finds the missing target's K-nearest neighbor known target columns using only the k informer labels. The missing target's non-informers are scored as the weighted average of its K-nearest neighbors. Note here that lowercase k refers to the informer compounds and uppercase K refers to the target column neighbors, which can be different numbers (e.g. $k = 16$ informers and $K = 5$ neighbors).

FCIIterativeImputer [100, 106] imputes missing column features as a function of known column features over a number of iterations. To do this, it trains a regression model that takes in the known target columns as input and predicts the missing target as output. In particular, each compound's known target labels are the input and its missing target label is the predicted output. It uses a first step heuristic to select the k informers which are used for training. The missing target's non-informers are scored as the regression model's predictions.

PairwiseSeqSimilarity is the same as the first step heuristic method shown in Figure 4.3. It requires a way to compute the similarity between the known targets and the missing target. This similarity measure is assumed to reflect the bioactivity; i.e. similar targets have similar bioactivity profiles. In the case of protein targets, the protein sequences can be compared to compute pairwise

sequence alignment scores [96]. The missing target's entire column is scored as the weighted average of the K nearest known targets based on alignment score. As an example, the PKIS1 dataset's protein sequences were downloaded from UniProt [12], and the local pairwise alignment scores were computed using the EMBOSS Water program [13].

4.4 Description of CustomMC

Here we will again use the matrix index slicing notation that is similar to the NumPy Python package. As a reminder, given a matrix $M \in \mathbb{R}^{10\times10}$ and a set of indices $J = \{0, 4, 5\}$, the submatrix $M[J, :] \in \mathbb{R}^{3\times10}$ is the matrix consisting of the rows 0, 4, and 5 of M. Similarly, the submatrix $M[:, J] \in \mathbb{R}^{10\times3}$ is the matrix consisting of the columns 0, 4, and 5 of M. The indices before the comma indicate which row indices to keep, and the indices after the comma indicate which column indices to keep.

In contrast to SimpleMC, CustomMC looks at the singular value decomposition (SVD) from the row space. From the SVD of the known submatrix $Y_{known} = U\Sigma V^T$, each row of Y_{known} is a linear combination of the rows in V^T. The linear coefficients are given in $(U\Sigma)$. Following this line of thinking, we assume that the rows of Y_{known} can be approximated by a linear combination of k informer rows (rows correspond to compounds) whose indices we denote as $InfSet$. In other words, any row i can be approximated as:

$$Y_{known}[i, :] \approx w^T Y_{known}[InfSet, :] \qquad \text{for some } w \in \mathbb{R}^k \tag{4.13}$$

Thus, the known submatrix is approximated as (also see Figure 4.6):

$$Y_{known} \approx W Y_{known}[InfSet, :] \qquad \text{for some } W \in \mathbb{R}^{m\times k} \tag{4.14}$$

Finding the best informer set $InfSet$ that minimizes the least-squares error reconstruction is combinatorial, and so, we opt for a relaxed modification that is not guaranteed to be the same. The

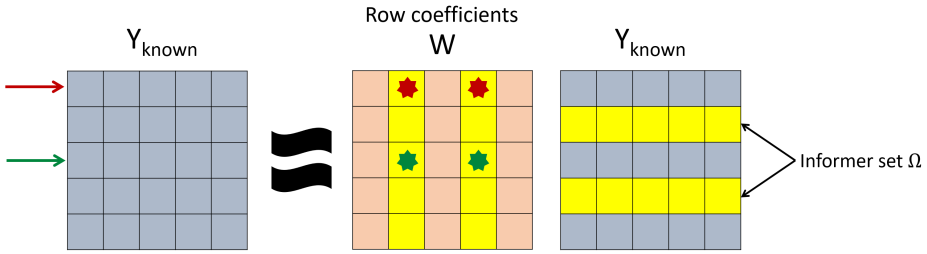


Figure 4.6: Illustration of the Y_{known} approximation via informer set compounds denoted in yellow and their corresponding weights. As an example, the first row denoted by the red arrow is approximated by a linear combination of the two informer rows (denoted in yellow) weighted by the two red-starred weight values. Similarly for the third row denoted by the green arrow.

modified optimization is as follows:

$$\Omega^*, W^* = \underset{\Omega, W; \|\Omega\|_1 \leq k}{\arg\min} \|Y_{known} - W\Omega Y_{known}\|_F^2 \tag{4.15}$$

Where $\Omega \in \{0,1\}^{m \times m}$ is a diagonal matrix whose elements sum to k; 1 indicates which row or compound we want to add to our informer set. $W \in \mathbb{R}^{m \times m}$ represents the coefficients to be optimized. See Figure 4.7 for an illustration. Note that for a matrix M, $M\Omega$ zeros out the unwanted columns, whereas ΩM zeros out the unwanted rows. Thus, $\Omega^* Y_{known}$ keeps only the k informer rows that can reconstruct Y_{known} with least error. To solve this via gradient descent, we allow Ω to be a real-valued diagonal matrix, but we apply a sigmoid function to restrict it in the range from 0 to 1. Furthermore we apply an L_1 penalty to the diagonal elements of Ω to promote sparsity. That is, we solve the following for $\Omega \in \mathbb{R}^{m \times m}$ and $W \in \mathbb{R}^{m \times m}$:

$$\hat{\Omega}, \hat{W} = \arg\min_{\Omega, W} \frac{1}{mn} \left\| Y_{known} - W f(\Omega) Y_{known} \right\|_F^2 + \lambda \left\| \Omega \right\|_1 \tag{4.16}$$

$$= \arg\min_{\Omega, W} \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(Y_{ij} - W_{i\cdot} f(\Omega) Y_{\cdot j} \right)^2 + \lambda \left\| \Omega \right\|_1 \tag{4.17}$$

$$f(\Omega) = \begin{bmatrix} \frac{1}{1+e^{-\Omega_1}} & 0 & \dots & 0 \\ 0 & \frac{1}{1+e^{-\Omega_2}} & \dots & 0 \\ 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & \frac{1}{1+e^{-\Omega_m}} \end{bmatrix} \tag{4.18}$$

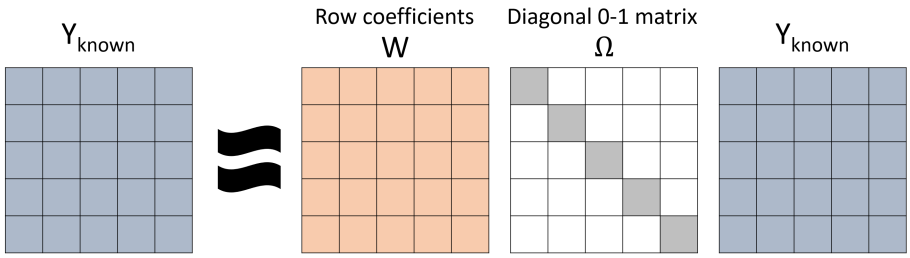


Figure 4.7: Illustration of the known target submatrix Y_{known} approximation using weights matrix W and diagonal informer selector matrix Ω.

Only the diagonal elements of Ω are tune-able. We solve this via gradient descent for N iterations using the TensorFlow [113] Python package. After every $n << N$ iterations, we compute the mean performance of the model across known targets on a metric of interest (e.g. NEF), and save the model parameters with the highest metric value observed thus far. With the returned $\hat{\Omega}$, we keep the top k diagonal elements setting them to 1, and zero out the remaining elements. Denote this adjusted diagonal matrix as $\dot{\Omega}$. These k elements correspond to the k informer rows (compounds). With the k informers identified, we can complete scoring the non-informers of the missing target. Let $Y_{missing}[\dot{\Omega}]$ denote the missing target column with the k informer labels revealed, then we complete $Y_{missing}$ as follows (see Figure 4.8):

$$\hat{Y}_{missing} = \hat{W} Y_{missing}[\dot{\Omega}] \tag{4.19}$$

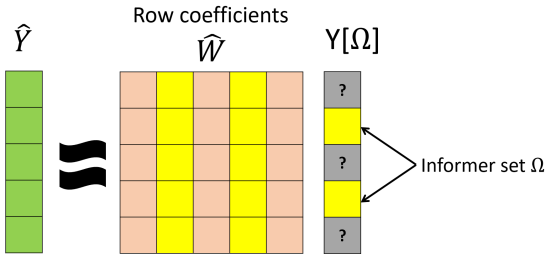


Figure 4.8: Illustration of the missing target completion denoted as $\hat{Y}$ with approximated weights $\hat{W}$. The yellow highlights indicate the informer label weights used to complete each element of the held target.

The above formulation minimizes the mean-squared-error (MSE) which is suitable for continuous values. In the case of binary values, the cross-entropy loss (or log-loss) is more suitable. The following is the optimization for a binary bioactivity matrix. Note how we apply another sigmoid to the predictions as is the case in supervised classification in machine learning.

$$\hat{\Omega}, \hat{W} = \arg\min_{\Omega, W} \frac{-1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} \Big(Y_{ij} \log(f(W_{i\cdot} f(\Omega) Y_{\cdot j})) + \\ (1 - Y_{ij}) \log(1 - f(W_{i\cdot} f(\Omega) Y_{\cdot j})) \Big) + \lambda \left\| \Omega \right\|_1 \tag{4.20}$$

4.5 Description of Cycle-based Importance Sampling Pipeline (CISP)

Inspiration for CISP

The idea for the CISP algorithm came about as a result of a post-analysis experiment of the ORS project from chapter 2. Recall the parameters of the project: the target is **PriA-SSB**, the binary training dataset is **PriA-SSB Training** with 79 hits and 72,344 non-hits, and the binary prospective dataset is **PriA-SSB Prospective** with 54 hits and 22,380 non-hits. Further recall that the RF_h model achieves the best performance prospectively, finding 37 out of the 54 hits in the top 250 predictions.

In the post-analysis experiment, we wanted to answer the following question: using RF_h, can we reduce the size of **PriA-SSB Training** and achieve similar performance on **PriA-SSB Prospective**; i.e. similar to 37 out of 54 hits on the top 250 predictions? The brute-force method of doing this would be to select each subset, train the model, and evaluate its performance. If for example we wanted the best subset of size 40, then we would need to evaluate 72,344 choose 40 subsets which is not feasible. Instead, we thought of a random sampling approach that can be summarized as follows:

1. Set the size of the desired subset as k. Set the number of runs n_{runs}.

2. For each run, randomly stratify sample a set Z of k points from **PriA-SSB Training**. Train RF_h on Z, then compute the number of hits in the top 250 predictions on **PriA-SSB Prospective**. Recall that RF_h was the best performing random forest model in the ORS case study in chapter 2. Stratify sampling is important since the percentage of actives in **PriA-SSB Training** is small, and so we need to ensure that actives are sampled via stratified sampling.

Figure 4.10 summarizes the results of 100 sample runs for different sizes of k. Of particular interest is that for small sizes of $k < 10,000$ out of $72,423$ compounds, the model is still able to achieve similar performance compared to when training on the entire training dataset. We further

scrutinize these runs by looking at the frequency of actives and inactives that appear in *good* runs. *Good* runs are those that achieve better than the similarity baseline performance; i.e. 31 out 54 hits in the top 250 predictions (see Table 2.5). Figure 4.11 shows the frequencies for the active indices that appear in the *good* runs that sample fewer than $10,000$ compounds. We use the **frequencies** as an **importance** metric. By selecting those actives and inactives that pass a certain frequency threshold, the sampling procedure can be repeated with this *reduced* training dataset. We continued this process until we had **20 actives and 20 inactives** that achieve 35 hits out of 54 in the top 250 predictions (see Figure 4.9 for depiction).

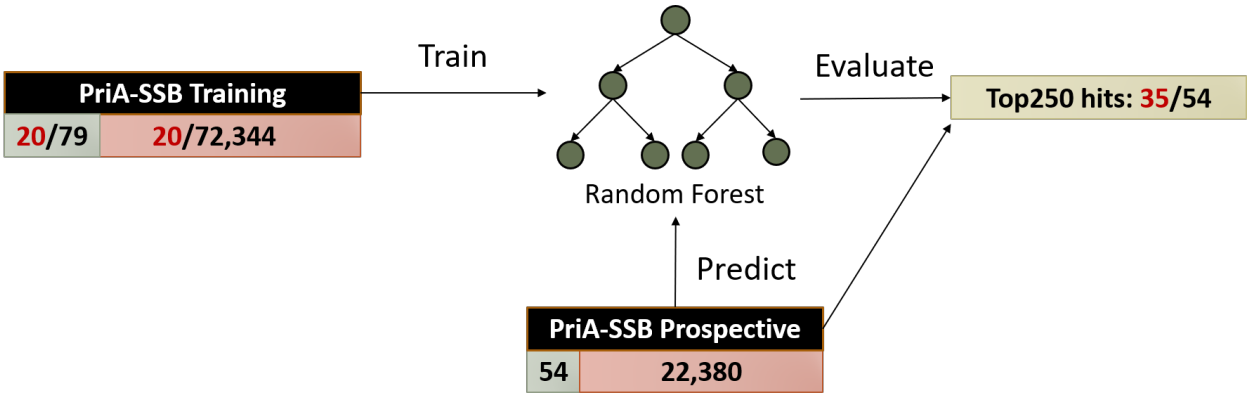


Figure 4.9: Depiction of the final reduced training dataset of 20 actives from 79 (olive) and 20 inactives from 72,344 (red) acquired by repeated random sampling and frequency thresholding. The RF_h model, trained on these 20 actives and 20 inactives, predicts on the prospective dataset and evaluates the top 250 hits to achieve 35 out of 54 hits. For comparison, training on all the **PriA-SSB Training** dataset achieves a top 250 score of 37 out of 54 hits.

Despite using only 40 compounds, we are only 2 hits shy of achieving the same performance of RF_h when trained on all $72,423$ compounds. However, it is important to discuss the major **limitations** of this approach:

- The *important* compounds were determined by repeated evaluations on the same prospective dataset. Thus it is directly overfitting the prospective data. This means we cannot use these compounds confidently to generalize to other compounds beyond the prospective set. Fortunately this was not the goal; the goal was to determine which of the training compounds are more important towards the prospective set. It is important to keep this goal in mind when

using this approach. We wanted to see how much reduction in the training set size would still give us good results on the prospective set. This was mainly done to satisfy curiosity and compare the learning of the random forest model with the similarity baseline.

In order to remove the prospective overfitting bias, we have two options as mentioned earlier. One method is to specify a validation set that is a subset of the training set which is used for evaluating runs. This would reduce the size of the training set; the set from which we select informers. Another method is to select informers from all the training set compounds, and use the remaining non-informer training compounds as the validation set for run evaluation. This allows using all training set compounds for selection, but direct comparison of runs is not possible; since different runs have different samples resulting in different validation sets. However, if the training set size is large enough and the sample size is small enough, this would result in a large enough validation set to get a reasonable approximation of the generalization error (from PAC learning theory).

- We found a small set of compounds that are predictive on the prospective dataset. Curiosity aside, the practical value of this is not directly applicable to the **PriA-SSB** case study since the cost of labelling has already been paid. However, if there was another *related* target to PriA-SSB, we might be able to assume that this small set of *informative* compounds on PriA-SSB is also *informative* on the related target. Taking this a step further, if we have a target with no data but multiple related targets with known bioactivity data, then we can construct a multi-target *informative* set for the no-data target. This is the idea behind the CISP algorithm as we will explain next.

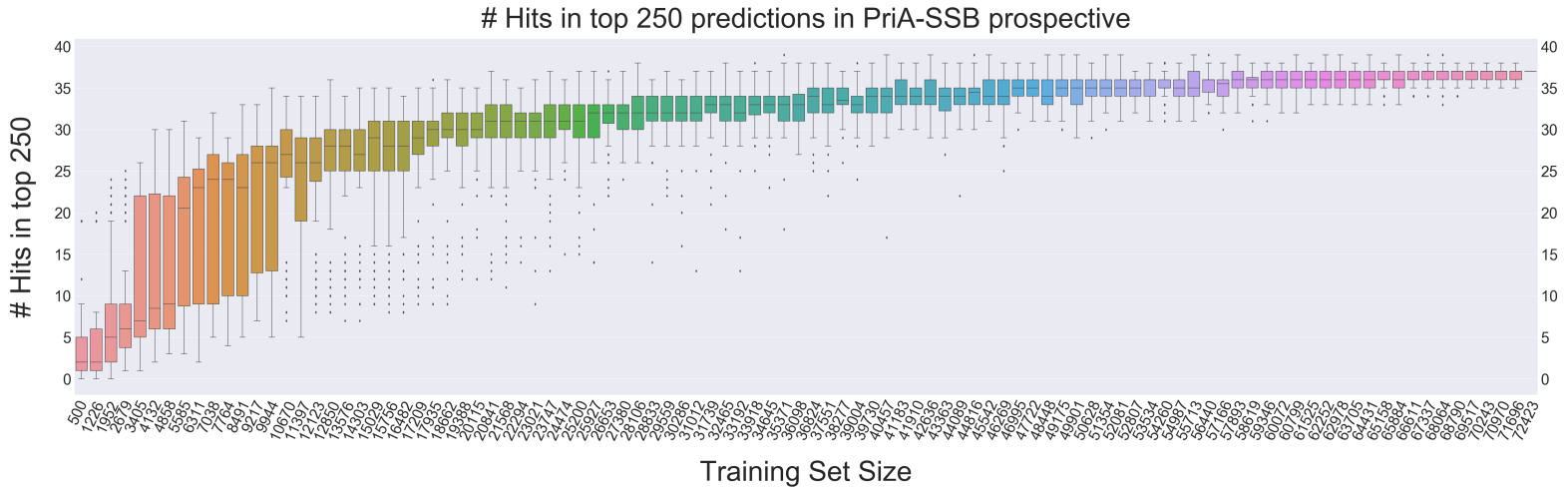


Figure 4.10: Number of hits in the top 250 predicts on **PriA-SSB Prospective** using RF_h trained on samples of **PriA-SSB Training**. Box plots summarize the results of 100 runs for each sample size k.

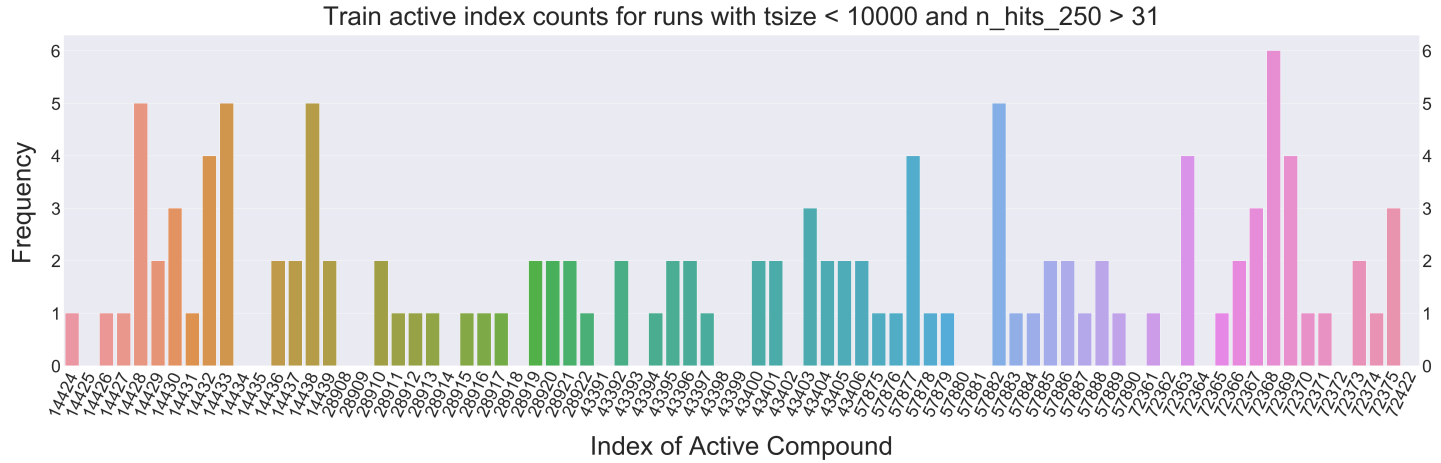


Figure 4.11: Active compound frequencies that appear in *good* runs. We limit counts to only runs that sampled fewer than $10,000$ compounds and achieved better than 31 out of 54 hits on the prospective dataset (31 hits is the baseline similarity performance).

CISP Pipeline Description

To help understand the CISP pipeline we will use the PKIS1 dataset by Zhang et al. [3, 2]. This dataset consists of **366** compounds and **224** targets from the kinase target class. PKIS datasets are suitable for learning off-target inference where the missing target is also of the kinase class. Consider the first target in PKIS1 as the missing target $Y_{missing}$ and the remaining 223 targets as known.

The **metric** we want to maximize is the normalized enrichment factor at 20%: $\mathbf{NEF}_{20\%}$ (see section 2.4 metric discussion). This metric helps judge method performance in identifying early retrieval of actives. We fix the learning algorithm to be a random forest (RF) model and 1024-bit RDKit Morgan fingerprint features for the compounds [106, 63, 1, 55]. Fingerprints with random forest models achieve competitive performance in VS and are used in state-of-the-art method comparisons [4, 34]. Denote the 366 compound fingerprint feature matrix as $X \in \{0,1\}^{366\times 1024}$ and the PKIS1 label matrix as $Y \in \{0,1\}^{366\times 224}$. The **goal** is to construct an informer set $Z \subset X$ for $Y_{missing}$ of size $k = 16$ compounds that achieves good performance in terms of $\text{NEF}_{20\%}$ when the RF model is trained on the informers Z and predicts on the non-informers $X \setminus Z$.

The CISP algorithm proceeds in cycles or iterations, starting from the full training dataset and slowly pruning it down with each cycle to the desired informer set size. The pruning process is similar to the inspiration idea described earlier using sampling and frequencies in *good* runs. Compounds are clustered according to a clustering algorithm like Taylor-Butina or Bemis-Murcko scaffolds [1, 114, 81, 80]. Clustering allows us to measure the diversity of samples and the actives during evaluation. Sampling is done in a stratified manner based on the cluster ID and the binary label of the compounds. This ensures that samples are both diverse and informative.

As a high level overview, Figure 4.12 illustrates the cycling process and the three main stages involved: pre-sampling, sampling, and analysis stage. Suppose an arbitrary target from among the 224 PKIS1 targets is the missing target and the remaining 223 targets are known. The goal is to select k informers for the missing target, obtain the corresponding labels, and then predict the missing target bioactivity of the remaining non-informers. The pre-sampling stage is responsible

for gauging the performance of the current training set. Since the missing target labels are unknown, we compute the average performance across the known targets when using the current training set to predict the bioactivity of the non-training set. This performance can be used as an indication to stop the cycling if performance is satisfactory. The sampling stage is responsible for finding important training set compounds for each of the 223 targets, individually. That is we conduct a similar stratify sampling strategy as in the inspiration idea in order to identify smaller training sets that achieve good performance on the remaining non-training compounds. In the analysis stage, we process the performance of the sampling runs for the 223 targets. Since we conduct sampling runs for each of the 223 targets individually, this produces 223 sets of important indices that need to merged together. This merging will pool together compounds that appeared as important training indices in many of the 223 targets, and subsequently prune them down based on frequency counts.

Now we describe the three stages in detail. Note that we define a number of hyperparameters that will be explored in the experiments.

- **Pre-sampling stage:** For each of the 223 known PKIS1 targets, use the current cycle's training set $T_i = (T_X \subset X, T_Y \subset Y)$, train the RF model on T_i and evaluate $\text{NEF}_{20\%}$ performance on the remaining compounds $(X \setminus T_X, Y \setminus T_Y)$. This stage evaluates the utility of the current informer set T_i. Note that for the first cycle there is no pre-sampling stage since the training set T_i is all of the 366 compounds in PKIS1.

- **Sampling stage:** Define a sample size m based on a combination of the size of T_i and number of clusters remaining. For each of the 223 known PKIS1 targets, randomly stratify a sample $S = (S_X, S_Y)$ of size m from the training set T_i based on cluster and target labels. Train an RF model on S and evaluate $\text{NEF}_{20\%}$ performance on the remaining compounds $T_i \setminus S$. Repeat this sample process n_{runs} times for each target (e.g. 1000 sample runs for each target). The details of the sampling can be found at the end of this section.

- **Analysis stage:** Analyze the results from the sampling stage via two functions: **Good Runs Filter** and **Important Index Filter**. The **Good Runs Filter** identifies, for each of the 223

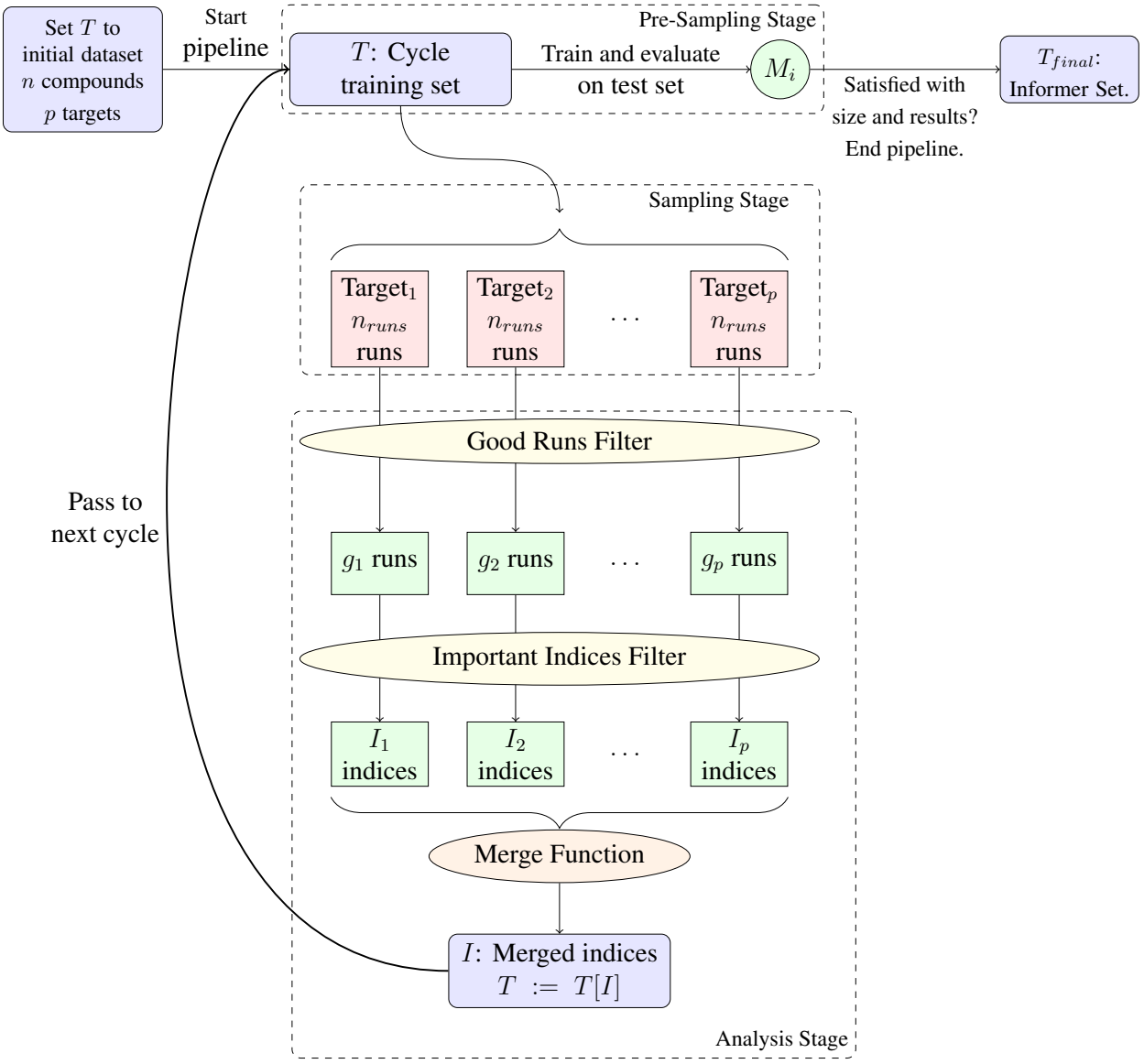


Figure 4.12: The CISP cycle-based informer set method illustrating the three stages within each cycle: pre-sampling, sampling, and analysis stage. Every cycle reduces the size of the informer set. The pipeline can be stopped as soon as an informer set of satisfactory size or performance is achieved.

targets, the runs that achieve $\text{NEF}_{20\%}$ performance exceeding the τ_{GRF} quantile of runs (e.g. the top 10% of runs). This is hoped to identify the samples that are *informative*. The **Important Index Filter** identifies, for each of the 223 targets, the indices that are important among the samples of the good target runs. By index we mean the index of the compound that uniquely identifies each compound in the 366 compound dataset. The importance of an index is determined by its **frequency count**; i.e. the number of times the compound appeared in **all** good sample runs. Naturally, we want to favor compounds that appeared many times in good runs. To ensure healthy representation of both actives and inactives in the next cycle, we use thresholds τ_a and τ_b for actives and inactives, respectively. These thresholds are set dynamically as the fraction between the number of unique active and inactive indices that appear in good runs (an example will follow shortly). Only those active compounds with frequency counts exceeding the τ_a quantile are promoted to the next step, similarly for inactive compounds using τ_b. We can also opt to normalize compound frequency counts as the fraction between the number of times a compound appears in good runs versus the number of times the compound was sampled in all runs (i.e. good and bad runs). We denote the option to normalize frequency counts as a true/false variable: W_{IIF}.

To exemplify this, consider a target with 4 unique actives and 16 unique inactives that are *good* indices, that is a quantile of $\tau_a = 0.2$ and $\tau_b = 0.8$ for active and inactives, respectively. From among the 4 unique actives, we select those with frequency counts that exceed that $\tau_a = 0.2$ quantile among the 4 actives. In other words, the top 80% of frequency counts for the 4 actives, which corresponds roughly to 3 actives. Similarly, we select inactives with frequency counts that exceed the $\tau_b = 0.8$ quantile, which corresponds roughly to 3 inactives. If we require an informer set of size 16, then the remaining 10 compounds are taken from the inactives. In the end, we ensure that a healthy amount of actives are promoted.

Once we identify important active and inactive indices for each target, we group all 223 targets' important indices via a **Merge Function**. We group all important actives from the 223 targets, compute the frequency of each active index, then threshold using τ_{MF}. A similar

process is done for inactive important indices. The idea is that we want to promote actives and inactives that appeared frequently as important indices in many of the targets. The qualifying active and inactive indices are then grouped together and set to the next cycle's training set.

Note that the **Merge Function** might result in discarding all actives for some of the targets (for not having sufficient frequency). In this case, the target is dropped from the pipeline in subsequent cycles. The reasoning is that this target is not in majority agreement with the remaining targets. To alleviate this problem, we can incorporate **target weights** if we suspect that some targets are **related** to $Y_{missing}$. For example, the pairwise sequence similarity matrix can be used to assign weights to indices from each known target. This means that targets that are more similar to the missing target will more likely pass on their important indices to the next stage.

This three-stage iterative process can be repeated until a satisfactory size and performance is achieved. Alternatively, if the desire is to return a fixed informer set of size k, then the cycles are iteratively run until the merge function either: identifies exactly k indices or fewer than k. In the latter case, the merge function corrects this by identifying the top k most important indices according to the frequency counts.

Figure 4.12 illustrates the CISP pipeline highlighting the three stages, inputs, outputs, and the cycling process. The algorithm terminates with the informer set T_{final} of k compounds. **Not depicted** is the final evaluation of this informer set against $Y_{missing}$. We evaluate by training the RF on the k informer compound features and missing target labels, then use this trained RF to compute non-informer predictions.

As for the sampling step in the Sampling stage, it is done as follows:

1. Define the minimum number to sample from each cluster as `min_sample_per_cluster`. Define the percentage to sample from each cluster as `cluster_sample_factor`.

2. Sample the minimum required samples from each cluster, if and only if, the ratio of total minimum samples and total compounds do not exceed the `sample_threshold`. This is

to avoid sampling a large number of compounds; e.g. sampling 100 compounds when there are only 120 compounds available.

3. Determine the number of compounds per cluster. If minimum sampling was done in step 2, then subtract the minimum sampled from each cluster's count. The total sample size is defined as the total cluster counts multiplied by the `cluster_sample_factor`.

4. Sample from each cluster proportionally until the sample size is exhausted; i.e. more sampling quota is given to clusters with more compounds. Note that sampling tries to maintain actives and inactives ratios within each cluster.

4.6 Experiment PKIS1

PKIS1 Overview

The dataset used in this experiment is the Published Kinase Inhibitor Sets (PKIS): PKIS1 (366 compounds and 224 targets) from Drewry et al. [3]. We use the adapted version from Zhang et al. [2] for ease of comparison with their methods. There are two versions of the bioactivity matrix: continuous inhibition and binary activity. The binary version is used for all the informer set methods except for the 9 methods from Zhang et al. and the continuous variant of CustomMC. The binary activity was determined per target by thresholding the inhibition at two standard deviations above the mean. For the pairwise sequence similarity matrix, the protein sequences were downloaded from the UniProt website [12] and local pairwise sequence alignment scores were computed using EMBOSS Water [13]. Table 4.2 showcases the UniProt ID and the hit counts of the 224 targets. Note that many of the targets are mutants which we assign the same UniProt ID, usually the non-mutant *wild-type*. However, this will result in perfect alignment for the mutants. Figure 4.13 is the heatmap of the resulting pairwise sequence matrix. The alignment scores can range from 0 (none) to 1 (perfect) alignment; for PKIS1 the alignment scores ranged from 0.235 to 1.

The **first goal** of this experiment is to perform parameter tuning on CISP and other methods using 5 random targets from PKIS1 as the missing target. The values of the parameter sets will be defined in the next section. The CISP parameter sets mainly adjust the quantile thresholds (τ_{GRF}, W_{IIF}, and τ_{MF}). In terms of the computational cost, running CISP on just 1 missing target is dominated by the 1000 sampling runs for each of the 223 known targets. Thus, we exploit the independence of runs in the sampling stage and distribute the work load across compute nodes. Fortunately, due to the small size of the PKIS1 dataset, the 1000 sampling runs for each task can be run on a single node in reasonable time. If, for example, six cycles were needed to get $k = 16$ informers, then this would be 6×223 compute nodes. Therefore, to reduce computational costs, we limit the fine-tuning to five randomly selected targets from the 224 PKIS1 targets: ABL1, CK1a, ITK, PASK, and ZAP70. Once good parameters have been determined, we can move on to the second goal of running on all 224 PKIS1 targets in a leave-one-target-out fashion.

The **second goal** is to evaluate the 21 informer set methods in the same spirit as in Zhang et al. [2]. In it, one of the evaluations was the leave-one-target-out performance on PKIS1 targets. Specifically, for each informer set method, we run the method 224 times. Each run sets one of the 224 targets as $Y_{missing}$ and uses the remaining 223 targets to construct an informer set Z of size $k = 16$. The informer set Z is then used to rank order the remaining 350 compounds. Evaluation was done using three metrics: AUC[ROC], a normalized enrichment factor metric at 10% ($\text{NEF}_{10\%}$), a hit novelty metric at 10% ($\text{FASR}_{10\%}$), and hits in the top 10%. Recall that $\text{NEF}_{10\%}$ is a normalized ratio between the hits retrieved by a method in its top 10% compound ranking versus the hits expected at random. However, note that the Zhang et al. version of $\text{NEF}_{10\%}$ differs than what was introduced in section 2.4. It is adjusted as follows:

$$NEF_{10\%} = 1 + \frac{\frac{EF_{10\%}-1}{EF_{max,10\%}-1}}{2} \tag{4.21}$$

This adjustment returns a value between 0.5 and 1 that reflects performance relative to a random-guesser; i.e. a value greater than 0.5 indicates a better than random method. The Fraction of

Active Scaffolds Retrieved ($FASR_{10\%}$) metric measures the fraction of unique active scaffolds in a method's top 10% of compound rankings. The hits in the top 10% metric include informer actives but disregard informer inactives in the top 10% budget. According to Zhang et al., compound scaffolds were defined using RDKit's MurckoScaffold [1, 114].

Table 4.2: Summary of the hit counts in the PKIS1 dataset consisting of 224 targets and 366 compounds. This dataset was taken from Zhang et al. [2] which was adapted from Drewry et al. [3].

Task ID	UniProt ID	# Hits	Hit %
ABL1	P00519	31	8.47
ABL1_E255K	P00519	32	8.74
ABL1_H396P	P00519	32	8.74
ABL1_M351T	P00519	33	9.02
ABL1_Q252H	P00519	32	8.74
ABL1_T315I	P00519	27	7.38
ABL1_Y253F	P00519	31	8.47
AKT1	P31749	9	2.46
AKT2	P31751	7	1.91
AKT3	Q9Y243	11	3.01
ALK	Q9UM73	19	5.19
AMPKA1_A1B1G1	Q13131	19	5.19
AMPKA2_A2B1G1	P54646	17	4.64
ARG	P42684	31	8.47
ARK5	O60285	28	7.65
Aurora.A	O14965	32	8.74
Aurora.B	Q96GD4	28	7.65
Aurora.C	Q9UQB9	37	10.11
AXL	P30530	21	5.74
BLK	P51451	26	7.10
BMX	P51813	24	6.56
BRAF	P15056	17	4.64
BRAF_V599E	P15056	19	5.19
BRK	Q13882	28	7.65
BRSK1	Q8TDC3	25	6.83
BRSK2	Q8IWQ3	28	7.65
BTK	Q06187	13	3.55

Continued on next page

Table 4.2: Summary of the hit counts in the PKIS1 dataset consisting of 224 targets and 366 compounds. This dataset was taken from Zhang et al. [2] which was adapted from Drewry et al. [3].

Task ID	UniProt ID	# Hits	Hit %
CAMK1D	Q8IU85	10	2.73
CAMK2A	Q9UQM7	11	3.01
CAMK2D	Q13557	17	4.64
CAMK4	Q16566	11	3.01
CDK1_cyclinB	P06493	26	7.10
CDK2_cyclinA	P24941	31	8.47
CDK2_cyclinE	P24941	28	7.65
CDK3_cyclinE	Q00526	25	6.83
CDK4_cyclinD	P11802	24	6.56
CDK5_p35	Q00535	27	7.38
CDK6_cyclinD3	Q00534	33	9.02
CHEK1	O14757	13	3.55
CHEK2	O96017	13	3.55
CK1a	P48729	25	6.83
CK1.g1	Q9HCP0	10	2.73
CK1.g2	P78368	13	3.55
CK1.g3	Q9Y6M4	15	4.10
CK2	P68400	24	6.56
CLK2	P49760	31	8.47
CLK3	P49761	11	3.01
CRAF	P04049	12	3.28
CSK	P41240	16	4.37
DAPK1	P53355	8	2.19
DCAMKL2	Q8N568	10	2.73
DDR2	Q16832	27	7.38
DYRK1A	Q13627	32	8.74
DYRK1B	Q9Y463	29	7.92
DYRK2	Q92630	23	6.28
EGFR	P00533	43	11.75
EGFR_L858R	P00533	32	8.74
EGFR_L861Q	P00533	27	7.38
EGFR_T790M	P00533	14	3.83
EGFR_T790M.L858R	P00533	18	4.92

Continued on next page

Table 4.2: Summary of the hit counts in the PKIS1 dataset consisting of 224 targets and 366 compounds. This dataset was taken from Zhang et al. [2] which was adapted from Drewry et al. [3].

Task ID	UniProt ID	# Hits	Hit %
EPHA2	P29317	21	5.74
EPHA3	P29320	12	3.28
EPHA4	P54764	29	7.92
EPHB2	P29323	30	8.20
EPHB3	P54753	10	2.73
EPHB4	P54760	22	6.01
ERBB2	P04626	23	6.28
ERBB4	Q15303	37	10.11
FER	P16591	22	6.01
FES	P07332	18	4.92
FGFR1	P11362	16	4.37
FGFR2	P21802	20	5.46
FGFR3	P22607	15	4.10
FGFR4	P22455	15	4.10
FGR	P09769	26	7.10
FLT1	P17948	24	6.56
FLT3	P36888	35	9.56
FLT3_D835Y	P36888	26	7.10
FLT4	P35916	26	7.10
FMS	P07333	30	8.20
FYN	P06241	26	7.10
GRK6	P43250	14	3.83
GRK7	Q8WTQ7	11	3.01
GSK3A	P49840	34	9.29
GSK3B	P49841	35	9.56
HCK	P08631	25	6.83
HIPK1	Q86Z02	27	7.38
HIPK4	Q8NE63	26	7.10
IGF1R	P08069	22	6.01
IKKA	O15111	13	3.55
IKKB	O14920	17	4.64
IKKE	Q14164	18	4.92
INSR	P06213	22	6.01

Continued on next page

Table 4.2: Summary of the hit counts in the PKIS1 dataset consisting of 224 targets and 366 compounds. This dataset was taken from Zhang et al. [2] which was adapted from Drewry et al. [3].

Task ID	UniProt ID	# Hits	Hit %
IRAK4	Q9NWZ3	7	1.91
IRR	P14616	13	3.55
ITK	Q08881	7	1.91
JAK1	P23458	13	3.55
JAK2	O60674	17	4.64
JAK3	P52333	19	5.19
JNK2	P45984	20	5.46
KDR	P35968	14	3.83
KIT	P10721	24	6.56
KIT_D816V	P10721	10	2.73
KIT_T6701	P10721	29	7.92
KIT_V560G	P10721	31	8.47
LCK	P06239	28	7.65
LOK	O94804	30	8.20
LRRK2	Q5S007	21	5.74
LRRK2_G2019S	Q5S007	26	7.10
LTK	P29376	22	6.01
LYNA	P07948	28	7.65
LYNB	P07948	28	7.65
MAP4K2	Q12851	21	5.74
MAP4K4	O95819	32	8.74
MAPK1	P28482	17	4.64
MAPK3	P27361	8	2.19
MAPKAPK2	P49137	19	5.19
MAPKAPK3	Q16644	33	9.02
MARK1	Q9P0L2	18	4.92
MARK3	P27448	16	4.37
MARK4	Q96L34	19	5.19
MEK1	Q02750	18	4.92
MELK	Q14680	27	7.38
MER	Q12866	19	5.19
MET	P08581	18	4.92
MINK	Q8N4C8	31	8.47

Continued on next page

Table 4.2: Summary of the hit counts in the PKIS1 dataset consisting of 224 targets and 366 compounds. This dataset was taken from Zhang et al. [2] which was adapted from Drewry et al. [3].

Task ID	UniProt ID	# Hits	Hit %
MKNK1	Q9BUB5	20	5.46
MNK2	Q9HBH9	20	5.46
MRCKA	Q5VT25	14	3.83
MRCKB	Q9Y5S2	14	3.83
MSK1	O75582	17	4.64
MSK2	O75676	18	4.92
MSSK1	Q9UPE1	10	2.73
MST1	Q13043	28	7.65
MST2	Q13188	23	6.28
MST4	Q9P289	11	3.01
MUSK	O15146	21	5.74
NEK1	Q96PY6	14	3.83
NEK2	P51955	10	2.73
NEK6	Q9HC98	12	3.28
NEK7	Q8TDX7	4	1.09
NEK9	Q8TD19	15	4.10
P38alpha	Q16539	40	10.93
P38beta	Q15759	29	7.92
P38delta	O15264	14	3.83
P38gamma	P53778	10	2.73
p70s6K1	P23443	21	5.74
PAK1	Q13153	7	1.91
PAK2	Q13177	7	1.91
PAK3	O75914	10	2.73
PAK5	Q9P286	8	2.19
PAK6	Q9NQU5	6	1.64
PAR.1Balpha	Q7KZI7	16	4.37
PASK	Q96RG2	11	3.01
PDGFRA	P16234	19	5.19
PDGFRA_D842V	P16234	28	7.65
PDGFRA_T674I	P16234	34	9.29
PDGFRA_V561D	P16234	22	6.01
PDGFRB	P09619	22	6.01

Continued on next page

Table 4.2: Summary of the hit counts in the PKIS1 dataset consisting of 224 targets and 366 compounds. This dataset was taken from Zhang et al. [2] which was adapted from Drewry et al. [3].

Task ID	UniProt ID	# Hits	Hit %
PDK1	Q15118	14	3.83
PHKG1	Q16816	22	6.01
PHKG2	P15735	10	2.73
PI3K.alpha	P42336	15	4.10
PI3K.delta	O00329	13	3.55
PI4K.beta	Q9UBF8	8	2.19
PIM1	P11309	20	5.46
PIM2	Q9P1W9	12	3.28
PIM3	Q86V86	26	7.10
PKA	P17612	21	5.74
PKC.alpha	P17252	16	4.37
PKC.beta1	P05771	20	5.46
PKC.beta2	P05771	18	4.92
PKC.eta	P24723	18	4.92
PKC.gamma	P05129	18	4.92
PKC.iota	P41743	10	2.73
PKC.theta	Q04759	16	4.37
PLK1	P53350	23	6.28
PRAK	Q8IW41	7	1.91
PRKD1	Q15139	19	5.19
PRKD2	Q9BZL6	27	7.38
PRKD3	O94806	27	7.38
PRKG1	Q13976	22	6.01
PRKG2	Q13237	27	7.38
PRKX	P51817	18	4.92
PTK5	P42685	13	3.55
PYK2	Q14289	28	7.65
RET	P07949	32	8.74
RET_V804L	P07949	31	8.47
RET_Y791F	P07949	30	8.20
ROCK1	Q13464	20	5.46
ROCK2	O75116	25	6.83
RON	Q04912	13	3.55

Continued on next page

Table 4.2: Summary of the hit counts in the PKIS1 dataset consisting of 224 targets and 366 compounds. This dataset was taken from Zhang et al. [2] which was adapted from Drewry et al. [3].

Task ID	UniProt ID	# Hits	Hit %
ROS	P08922	31	8.47
RSK1	Q15418	21	5.74
RSK2	P51812	21	5.74
RSK3	Q15349	22	6.01
RSK4	Q9UK32	20	5.46
SGK1	O00141	13	3.55
SGK2	Q9HBY8	11	3.01
SGK3	Q96BR1	15	4.10
SNF1LK2_QIK	Q9H0K1	20	5.46
SNF1LK_SIK	P57059	23	6.28
SPHK1	Q9NYA1	17	4.64
SPHK2	Q9NRA0	9	2.46
SRC	P12931	29	7.92
SRMS	Q9H3Y6	16	4.37
SRPK1	Q96SB4	11	3.01
SYK	P43405	12	3.28
TBK1	Q9UHD2	20	5.46
TEC	P42680	24	6.56
TIE2	Q02763	25	6.83
TNK1	Q13470	29	7.92
TNK2	Q07912	16	4.37
TRKA	P04629	24	6.56
TRKB	Q16620	26	7.10
TRKC	Q16288	26	7.10
TSSK1	Q9BXA7	21	5.74
TSSK2	Q96PF2	10	2.73
TTK	P33981	23	6.28
TXK	P42681	19	5.19
TYK2	P29597	17	4.64
TYRO3	Q06418	25	6.83
YES	P07947	28	7.65
ZAP70	P43403	15	4.10
Mean/Std.		20.58 / 7.84	5.62 / 2.14

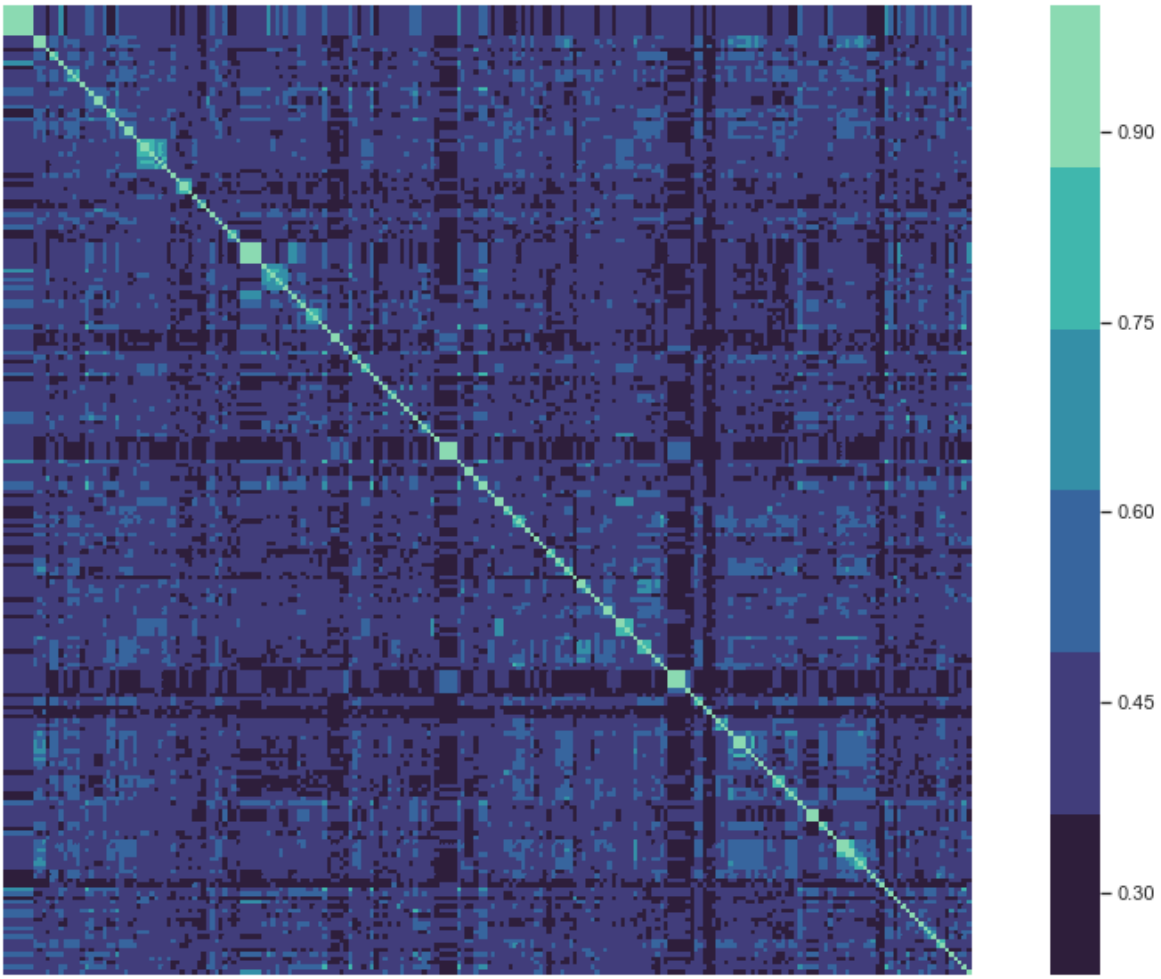


Figure 4.13: PKIS1's 224 targets local pairwise sequence alignment matrix visualized as a heatmap of values from 0 (no-alignment) to 1 (perfect-alignment). Sequences were downloaded from UniProt [12] and local pairwise alignment scores were computed using EMBOSS Water [13].

PKIS1 Parameter Tuning

In this part of the experiment, we fine-tune parameters for various informer set methods using 5 PKIS1 targets: ABL1, CK1a, ITK, PASK, and ZAP70. The main focus will be on the CISP parameters. First, we conducted manual exploration of the PKIS1 dataset to determine suitable fixed parameters for the sampling and clustering parameters. Table 4.3 shows the parameters defined for the number of sample runs n_{runs} and the three cluster sampling parameters. The sampling component of the sampling stage was tested individually to inspect if it behaved as desired. Mainly, we want a healthy representation of compounds from each unique cluster to ensure diversity in the

next cycle. Furthermore, we want to also reduce the number of compounds that are being sampled in the current cycle to ensure that we are moving towards the k informer size goal. An explanation of the sampling procedure is described in section 4.5. Table 4.4 shows the random forest model fixed parameters for all stages. These were selected based on past experience with random forest models and compound fingerprint features [63, 34].

Next, the tune-able CISP parameters are shown in Table 4.5 delineated by the parameter ID. The three parameters influence the analysis stage's behavior as discussed in section 4.5. We define six parameter configurations that were continually expanded based on results; i.e. we started with CISP_0 then defined other parameter IDs based on results. This process was stopped after a deadline time-frame had passed. Furthermore, we ran CISP on these six configurations supplemented with the pairwise sequence similarity matrix (denoted with _sim suffix). The pairwise sequence similarity matrix is used to weigh target indices in the merging function of the analysis stage. This gives a total of 12 CISP configurations.

Figure 4.14 shows the boxplots and mean performances of the 12 CISP configurations on the 5 PKIS1 tasks. The figure shows the four metrics used in Zhang et al. AUC[ROC], $\text{NEF}_{10\%}$, $\text{FASR}_{10\%}$, and hits in the top 10%. We are more interested in $\text{NEF}_{10\%}$ and $\text{FASR}_{10\%}$ since they reflect early hit retrieval and novelty, respectively. In general we notice a trend that supplementing CISP with the similarity matrix improves performance. However, keep in mind that for other datasets, the pairwise sequence similarity might not be available due to the nature of the targets. CISP_2 and CISP_3 are objectively the top 2 performers across all metrics. To discriminate between the two, we looked at the median performance. CISP_3 outperforms CISP_2 on median AUC[ROC], $\text{NEF}_{10\%}$ and $\text{FASR}_{10\%}$. Furthermore, CISP_3 improves when supplemented with a similarity matrix, whereas CISP_2 worsens. Thus, we select CISP_3 to promote to the next round of experiments.

For the other informer set methods that were described in section 4.3, we conducted manual exploratory runs on the same 5 PKIS1 targets. This was done in an iterative manner by starting at a fixed setting and expanding based on the results. Table 4.6 shows only those methods whose parameters were tuned. In general, methods that were tuned were given roughly the same oppor-

tunity; i.e. CISP and methods in Table 4.6 were tuned with six to eight parameter settings. This parameter search was by no means exhaustive.

Table 4.3: CISP sampling stage fixed parameters for the PKIS1 experiment.

Parameter	Value
n_{runs}	1000
min_sample_per_cluster	2
cluster_sample_factor	0.25
sample_threshold	0.40

Table 4.4: CISP random forest fixed parameters for the PKIS1 experiment.

Parameter	Value
n_estimators	100
max_features	log2
min_samples_leaf	1
class_weight	balanced
random_state	20192603
oob_score	False

Table 4.5: CISP parameter configurations evaluated on the 5 PKIS1 tuning targets: ABL1, CK1a, ITK, PASK, and ZAP70.

Param ID	τ_{GRF} Good Runs Threshold	W_{IIF} Normalize Important Index Frequency?	τ_{MF} Merge Function Threshold
CISP_0	0.50	True	0.50
CISP_1	0.75	True	0.50
CISP_2	0.75	False	0.50
CISP_3	0.50	False	0.50
CISP_4	0.75	True	0.75
CISP_5	0.50	False	0.75

Table 4.6: Parameter configuration of other informer set methods. These are not all the parameters involved, but just those that were tuned on the 5 PKIS1 tuning targets. Details of fixed parameters like the random forest model, optimizers, loss functions, etc. can be found at the project's GitHub: `https://github.com/Malnammi/informer-set-drug-discovery`.

Method	λ in eq. 4.16	# Iterations
CustomMC	0.1	100000

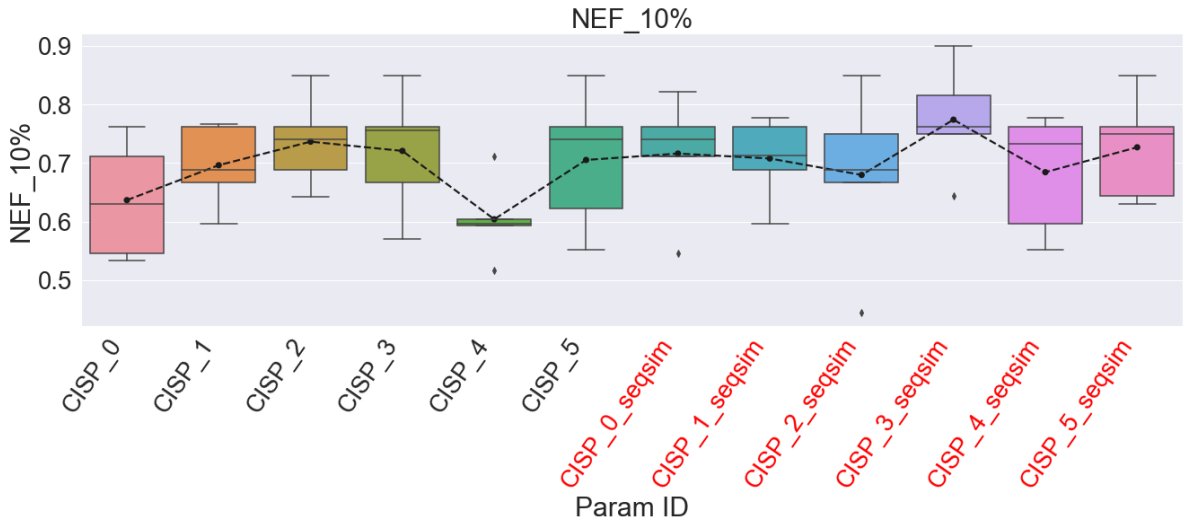
Method	Variance Threshold
SimpleMC	0.8

Method	Latent Dimension
FCIMatrixFactorization	10

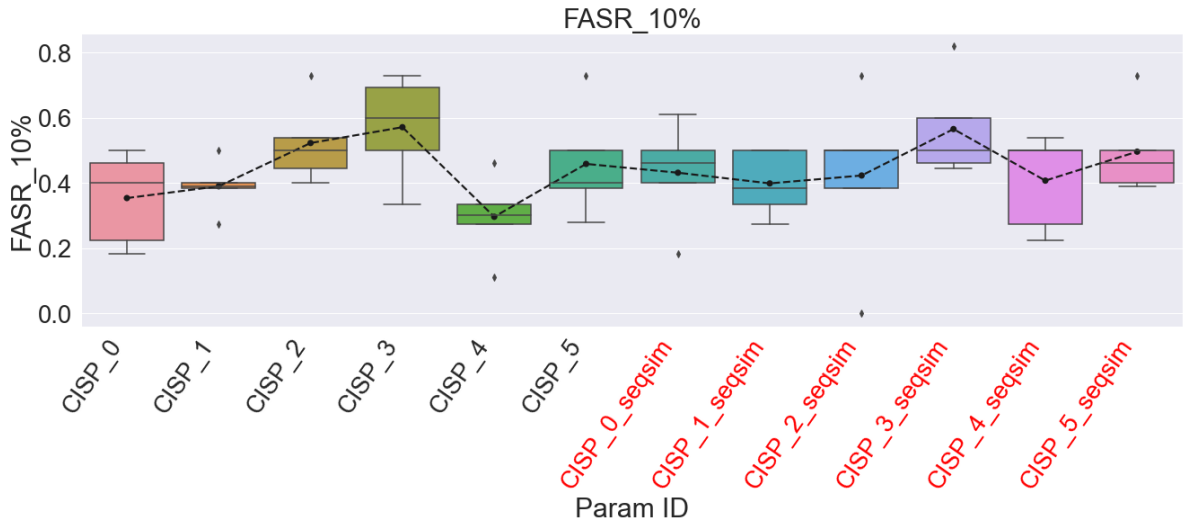
Method	# Clusters
TargetClusteringRF	10

Method	n_estimators	max_features	min_samples_leaf	class_weight
RFSupervised	1000	log2	1	balanced

Method	Hidden Layers	# Iterations
MTNNSupervised	[Dropout(rate=0.25), Dense(512, activation='relu'), Dropout(rate=0.25), Dense(256, activation='relu')]	50

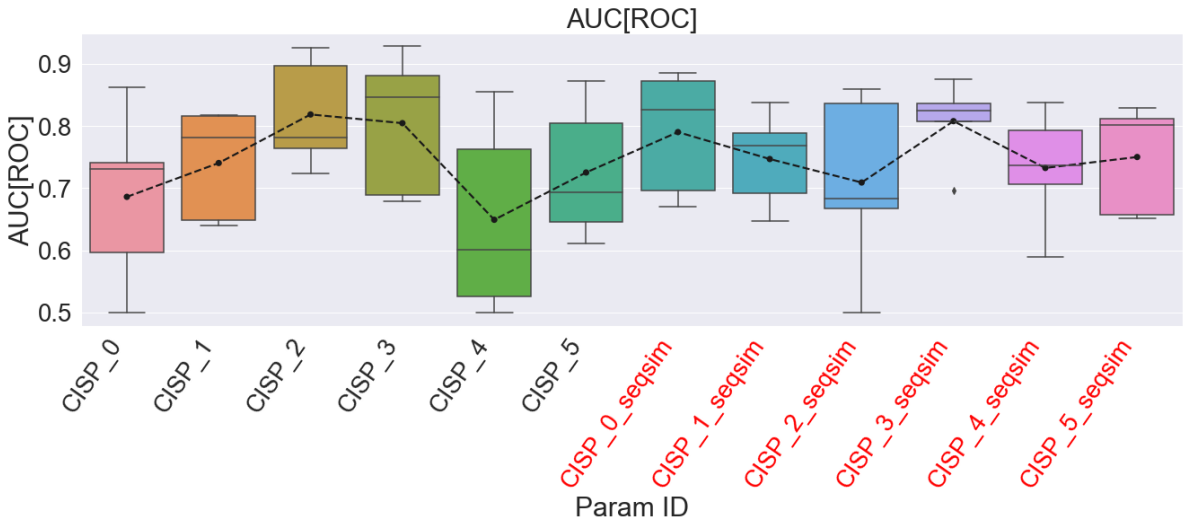


(a) $NEF_{10\%}$

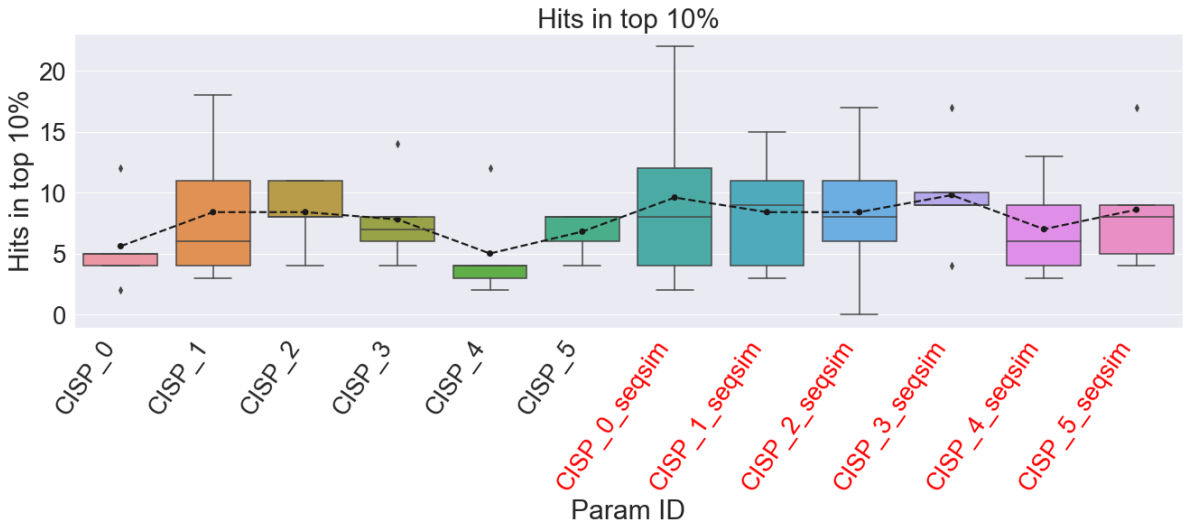


(b) $FASR_{10\%}$

Figure 4.14: CISP parameter configuration results on the 5 PKIS1 tuning targets: ABL1, CK1a, ITK, PASK, and ZAP70. The metrics are (a) $NEF_{10\%}$, (b) $FASR_{10\%}$, (c) AUC[ROC], and (d) hits in the top 10%. As used in Zhang et al. [2], hits in the top 10% counts the number of actives found in the top 10% of predicted non-informers, and also adds the number of informer actives. *(1 of 2 cont.)*



(c) AUC[ROC]



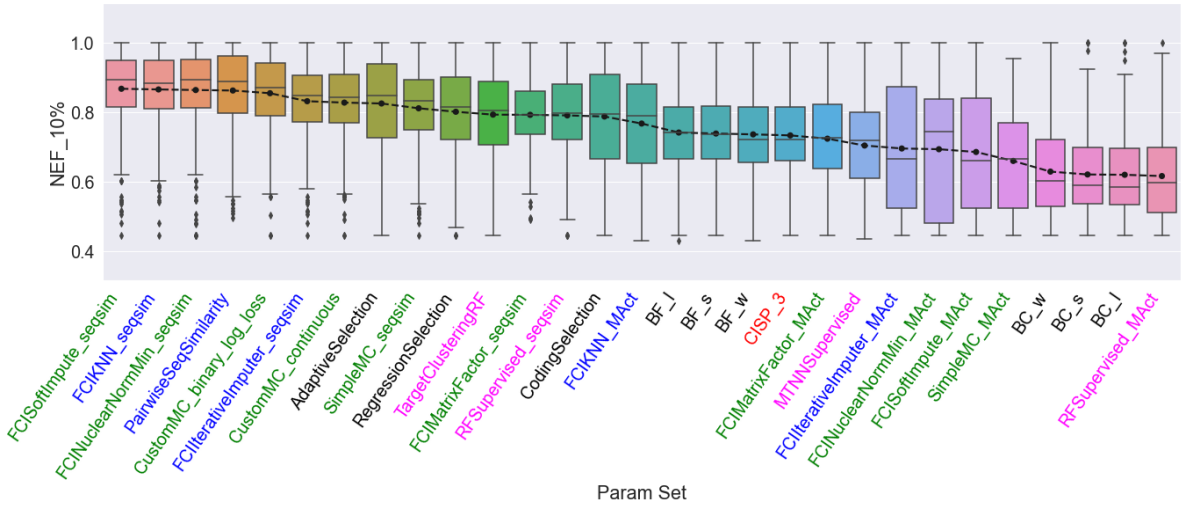
(d) Hits in top 10%

Figure 4.14: CISP parameter configuration results on the 5 PKIS1 tuning targets: ABL1, CK1a, ITK, PASK, and ZAP70. The metrics are (a) $NEF_{10\%}$, (b) $FASR_{10\%}$, (c) AUC[ROC], and (d) hits in the top 10%. As used in Zhang et al. [2], hits in the top 10% counts the number of actives found in the top 10% of predicted non-informers, and also adds the number of informer actives. *(2 of 2)*

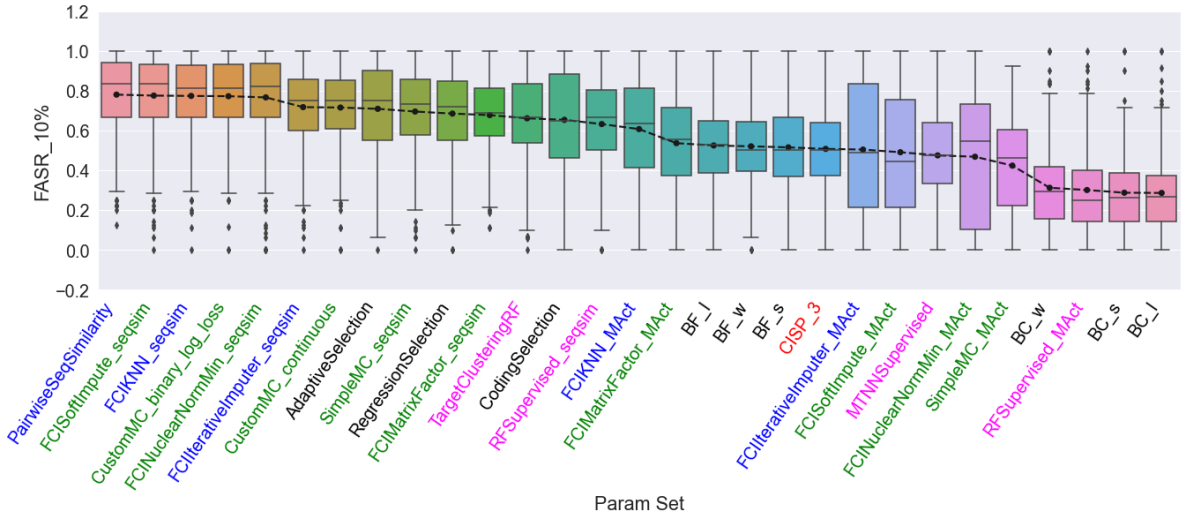
PKIS1 Leave-one-target-out Results

Now we run the informer set methods on each of the 224 PKIS1 target in a leave-one-target-out fashion; i.e. the held-out target is treated as the missing target. Figure 4.15 shows the boxplots of the methods across the 224 targets ordered from highest to lowest mean performance. Focusing on $\text{NEF}_{10\%}$, a clear trend is that methods that make use of the protein-protein sequence similarities (seqsim) in their first-step heuristic perform better than their most-active-across-targets (MAct) variant. In addition, the PairwiseSeqSimilarity method is among the top performers. This is more evident when looking at the direct comparison between the two first-step heuristics in Figure 4.16.

Disregarding methods that make use of the sequence similarities, the top performers are: CustomMC (both binary and continuous variants), Adaptive Selection, and Regression Selection. The CISP method does no better than the BF baselines from Zhang et al. [2]. Due to this low performance, and the fact that it takes considerable time and processing to run the CISP method for large datasets, we remove it from further experiments. The poor performance of CISP, and many of the adapted solutions with configurable parameters, may be in large part due to not performing an exhaustive enough search in the previous experiment. Although CustomMC has comparable performance to Adaptive Selection and Regression Selection, it is adaptable to handle missing values. The next experiment follows up on this adaptation and showcases performance on simulated missing at random values for the PKIS1 dataset.

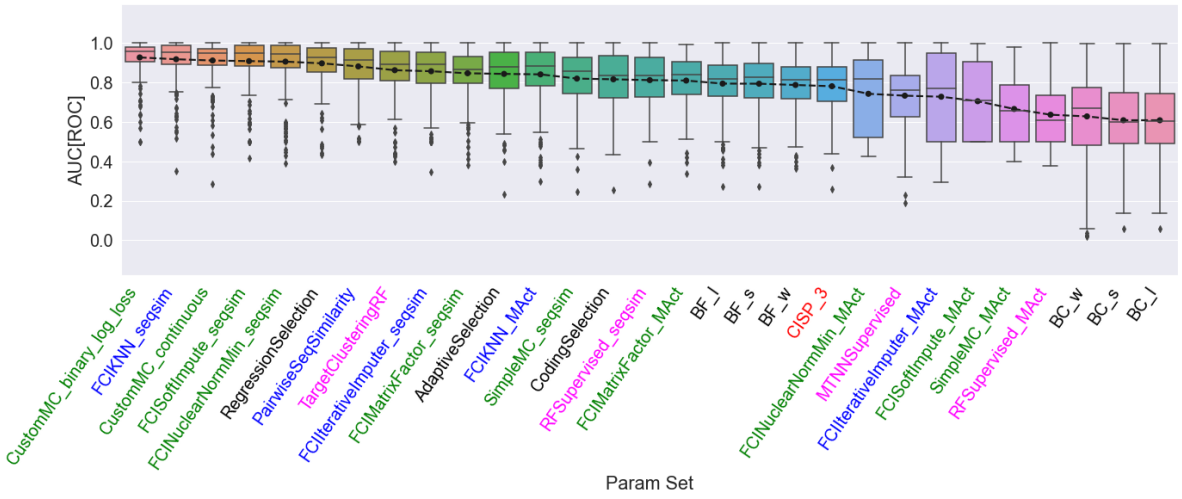


(a) $\text{NEF}_{10\%}$

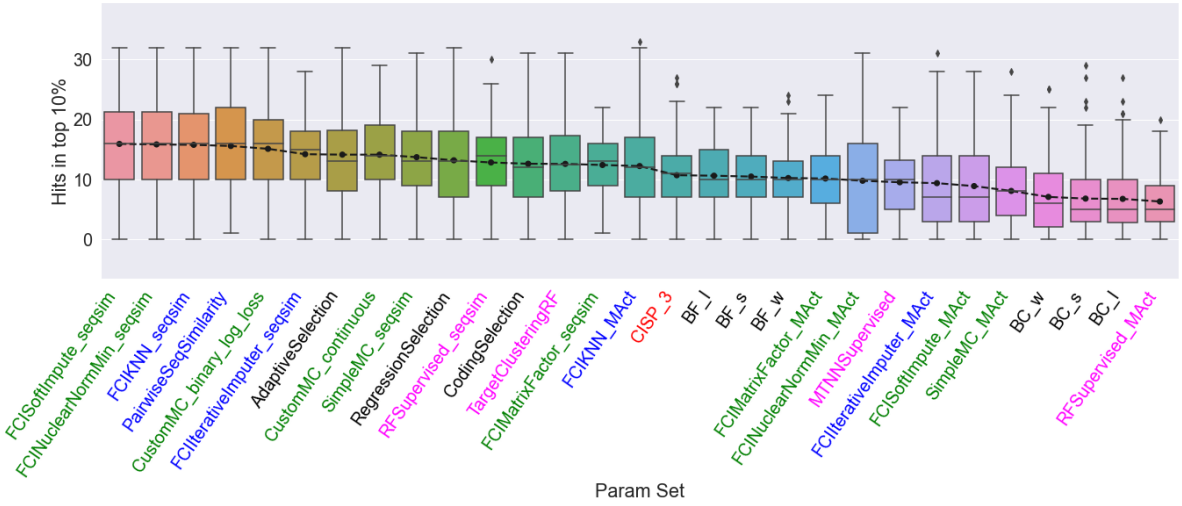


(b) $\text{FASR}_{10\%}$

Figure 4.15: Boxplots and means of the informer set methods on the 224 PKIS1 targets in a leave-one-target-out fashion. The metrics are (a) $\text{NEF}_{10\%}$, (b) $\text{FASR}_{10\%}$, (c) AUC[ROC], and (d) hits in the top 10%. As defined in Table 4.1, the text color denotes the solution type: matrix completion/factorization (green), supervised learning (magenta), CISP (red), nearest neighbor or imputation (blue), and Zhang et al. (black). *(1 of 2 cont.)*



(c) AUC[ROC]



(d) Hits in top 10%

Figure 4.15: Boxplots and means of the informer set methods on the 224 PKIS1 targets in a leave-one-target-out fashion. The metrics are (a) $NEF_{10\%}$, (b) $FASR_{10\%}$, (c) AUC[ROC], and (d) hits in the top 10%. As defined in Table 4.1, the text color denotes the solution type: matrix completion/factorization (green), supervised learning (magenta), CISP (red), nearest neighbor or imputation (blue), and Zhang et al. (black). *(2 of 2)*

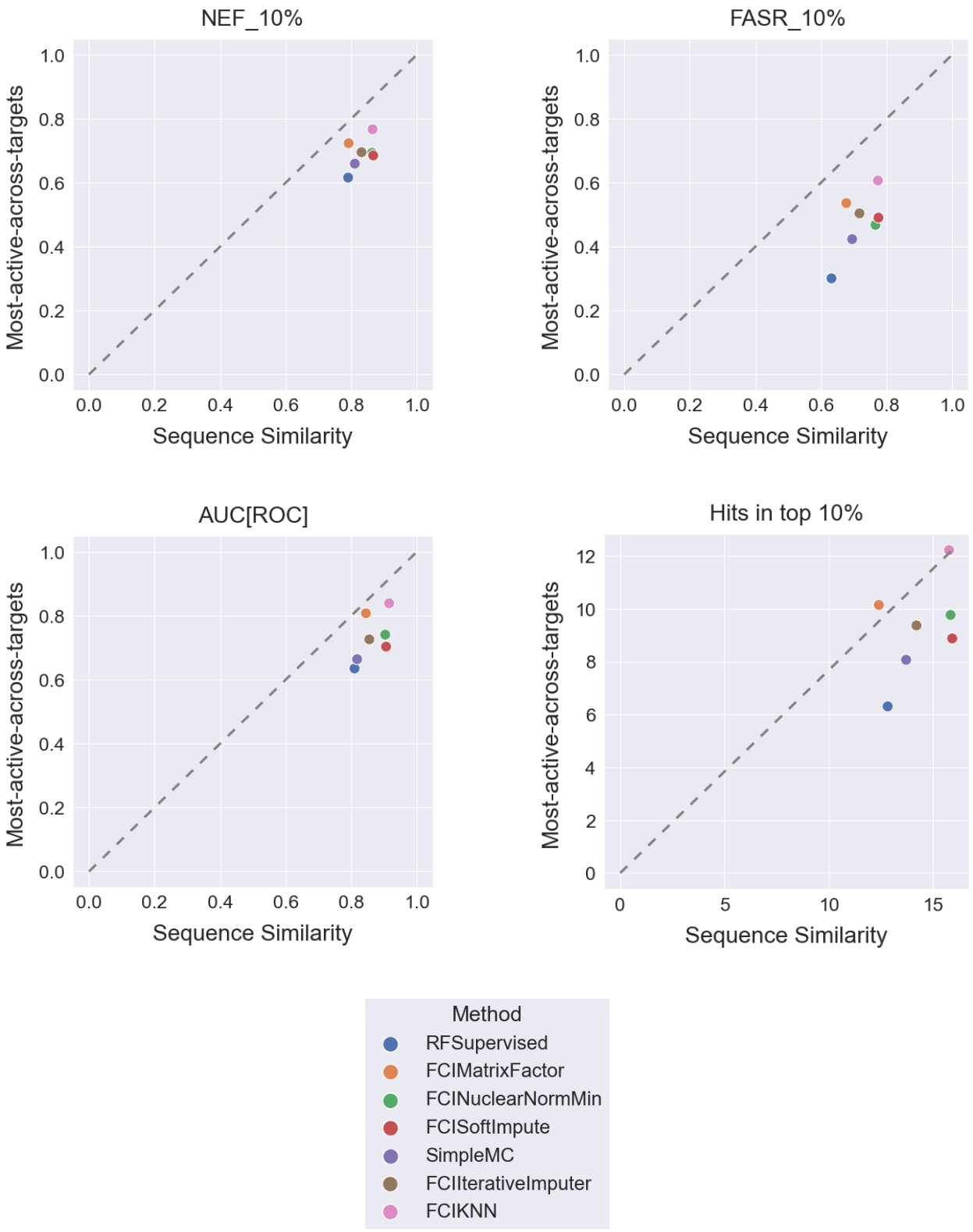


Figure 4.16: Mean performance comparison of informer set methods that use the two first-step heuristics: most-active-across-targets and sequence similarity (see Figure 4.3).

CustomMC with Missing Values

We modify the CustomMC optimization equation to the following:

$$\hat{\Omega}, \hat{W} = \underset{\Omega, W}{\arg\min} \frac{1}{mn} \left\| A \odot (Y_{known} - W f(\Omega) Y_{known}) \right\|_F^2 + \lambda \left\| \Omega \right\|_1 \tag{4.22}$$

$$f(\Omega) = \begin{bmatrix} \frac{1}{1+e^{-\Omega_1}} & 0 & \dots & 0 \\ 0 & \frac{1}{1+e^{-\Omega_2}} & \dots & 0 \\ 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & \frac{1}{1+e^{-\Omega_m}} \end{bmatrix}$$

A simple adjustment to handle missing values is to apply an element-wise multiplication $\odot$ with an observation matrix A of the same size as Y_{known} where 0 indicates missing and 1 indicates observed entries. This is to *zero-out* the loss contributions of the missing values when performing gradient descent. Thus, in an implementation, the missing values in Y_{known} can be replaced with any placeholder value (e.g. zero). The same treatment can be applied to the log-loss on binary activity labels:

$$\hat{\Omega}, \hat{W} = \underset{\Omega, W}{\arg\min} \frac{-1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij} \Big(Y_{ij} \log(f(W_{i\cdot} f(\Omega) Y_{\cdot j})) + (1 - Y_{ij}) \log(1 - f(W_{i\cdot} f(\Omega) Y_{\cdot j})) \Big) + \lambda \left\| \Omega \right\|_1 \tag{4.23}$$

To test that the implementation is working, and to see how far it can be pushed, we ran leave-one-target-out on PKIS1 with simulated missing-at-random at various missing percentages. We expect that performance drops as we increase the missing percentage. For a given percentage, say 10%, for each target column in Y_{known}, we uniformly sample 10% of the cells to be missing. Then the optimization is run to solve for the held-out target. Note that the held-out target has no missing values. This is done five times to see the effect of different samples on performance. Figure 4.17 shows the $\text{NEF}_{10\%}$ boxplot results for binary and continuous implementations at the missing percentages: 10%, 20%, 30%, 40%, 50%, 60%, 70%, and 80%. At 40% and above we start to

see decline in performance in both variants, but performance is still better than random (i.e. 0.5). At 70% we start to see the continuous variant overtaking the binary variant in performance. The results look promising as we can retain relatively good performance up to 40% missingness.

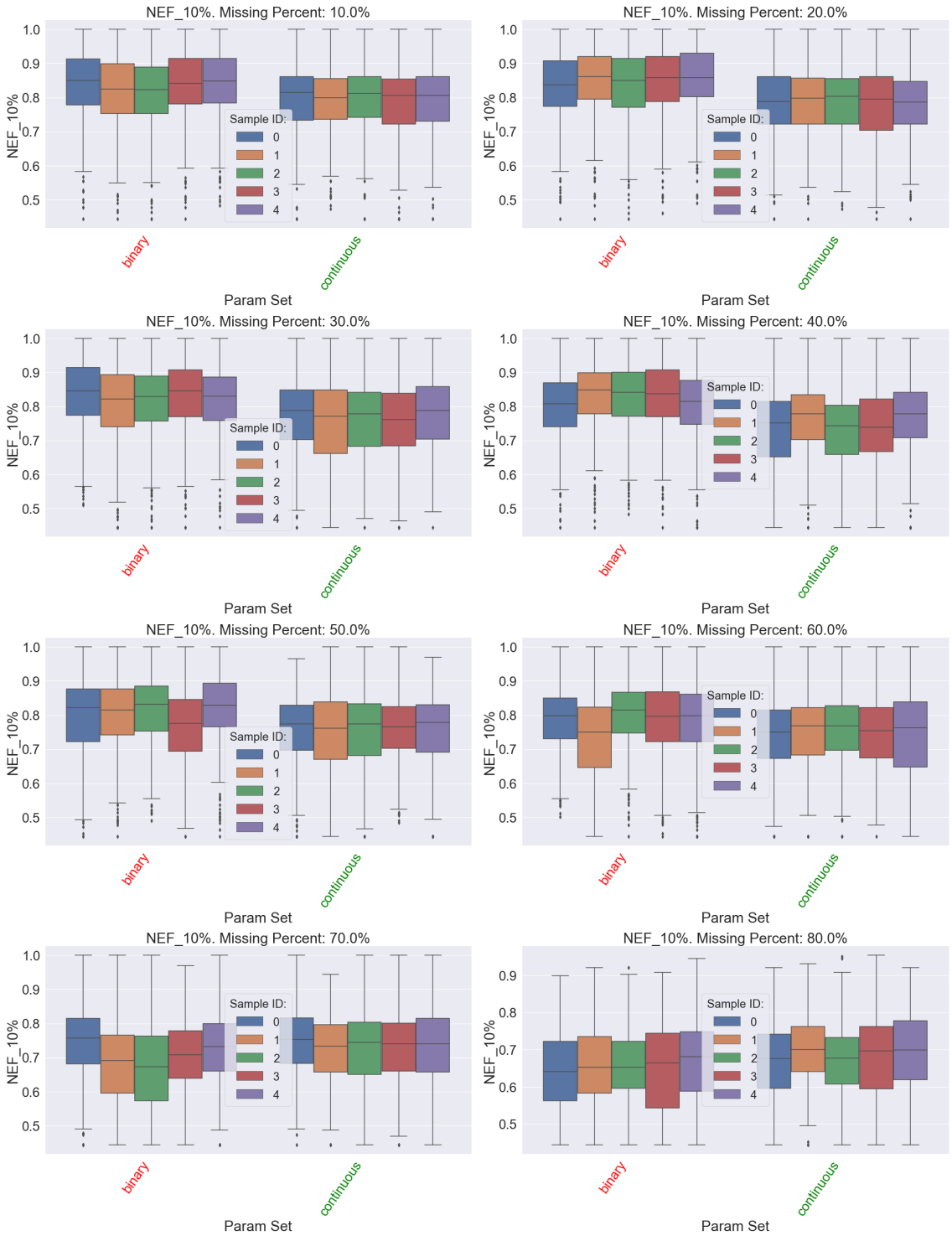
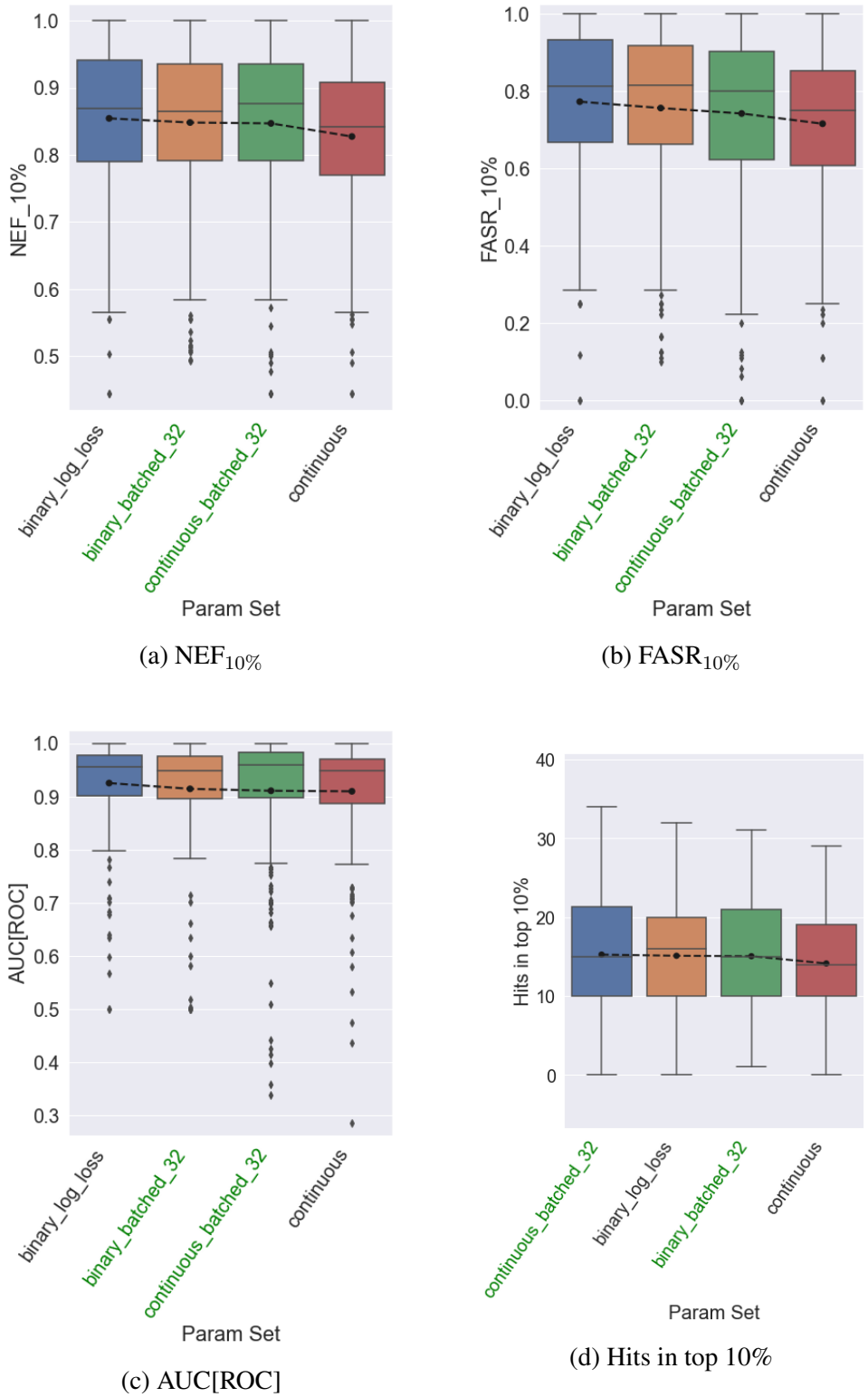


Figure 4.17: $\text{NEF}_{10\%}$ boxplots of CustomMC binary and continuous variants on the 224 PKIS1 targets in a leave-one-target-out fashion at various simulated missing-at-random percentages: 10%, 20%, 30%, 40%, 50%, 60%, 70%, and 80%. For a given percentage, the missing values are sampled at random for *each* of the known target columns; i.e. 80% of each target column is missing. The sample ID refers to the five different random sampling of missing value locations.

CustomMC with Missing Values and Batching

In preparation for larger datasets that can go up to hundreds of thousands of compounds, it is not feasible to perform gradient descent on the matrices involved in CustomMC due to CPU and GPU random-access-memory (RAM) constraints. As an example, if the known target matrix has 100,000 compounds and 128 targets, then W would require approximately 40 GB of memory. Fortunately, the optimization equation is simply a summation over the elements of the matrix. Borrowing from gradient descent concepts in machine learning that handle large datasets, we can perform stochastic gradient descent on batches of these elements. The matrix W can be stored on disk and batched into memory as needed. The index matrix Ω can be stored in vector format and implemented accordingly to save on memory.

Testing this implementation of the batched version on PKIS1 proved problematic at first with a batch size of 32. In later iterations, the loss would stagnate and reach a problematic local minima. To remedy this issue, the mini-batches were randomized during each iteration and the optimization was split into two steps. The first step would optimize both W and Ω as originally intended. After N iterations, the top k scoring Ω elements would be set as the informers. The second step would then reset and solve for W but with those k informers fixed. In other words, the first step finds the k informers, the second step finds the known targets' matrix weights using those informers. With this implementation the batched version of the optimization was smooth and yielded similar performance to the complete version. Figure 4.18 shows the boxplot and mean results for binary, continuous, batched-binary, and batched-continuous implementations. As can be seen, the mean results for these implementations are close.



(a) $NEF_{10\%}$

(b) $FASR_{10\%}$

(c) AUC[ROC]

(d) Hits in top 10%

Figure 4.18: Boxplots and mean performance of the binary and continuous, complete and batched implementations of CustomMC on PKIS1 leave-one-target-out evaluation. The metrics are (a) $NEF_{10\%}$, (b) $FASR_{10\%}$, (c) AUC[ROC], and (d) hits in the top 10%. The batched implementations use a batch size of 32.

4.7 Experiment PCBA

PCBA Overview

With the success of the missing and batched variant of CustomMC on PKIS1, the main goal of this project can be realized. Namely, the evaluation of performance on larger bioactivity matrices with missing values. To this end, we bring four informer set methods from the PKIS1 experiments: CustomMC (binary and continuous), RFSupervised, and MTNNSupervised. CustomMC achieved competitive performance on PKIS1 while not requiring sequence similarity information, and is adaptable to handle missing values. RFSupervised and MTNNSupervised are supervised learning methods that operate on a significantly different notion to the other informer set methods. In the case that the missing target has no relation or no pattern to be learned from the known targets, such a supervised learning method can be preferable as it is able to adapt its scoring of non-informers by learning the relationship between the compound features and the informer labels. Due to the diversity of the targets in the 128-PCBA dataset, we expected less target relatedness than in PKIS1. RFSupervised uses the same settings as in Table 4.6 and the most-active-across-targets first-step heuristic (denoted as MAct). MTNNSupervised uses the same architecture as in Table 4.6 but for 5000 iterations and batch size of 2056. CustomMC uses the batched version with batch size of 1024.

The dataset used in this experiment is the 128 assays from PubChem (128-PCBA) as introduced in Ramsundar et al. [5, 4]. This dataset consists of 128 assays, each treated as a target, and 439,877 compounds. These are 128 diverse targets consisting of proteins, protein-protein interactions, enzymes, GPCRs, etc. (see appendix in Ramsundar et al. [4]). Thus, it is expected that there is little target relatedness. Using the search criterion as detailed by Ramsundar et al., the assays were downloaded and processed to construct a compound-target bioactivity matrix. This dataset is not as clean as PKIS1 since the assays are run at varying concentrations and can have a large number of exclusive compounds. This makes constructing a continuous bioactivity matrix problematic as the targets are not at the same concentration. Most, if not all, of the assays perform a primary

screen at varying concentrations, and then follow up with a secondary screen on the promising compounds. The secondary screen along with filters like PAINS are used to assign an activity score to a compound from 0 to 100. This activity score is the PUBCHEM_ACTIVITY_SCORE column in the assay file. These scores are used for defining Active, Inconclusive, and Inactive compounds as denoted in the PUBCHEM_ACTIVITY_OUTCOME column. For many of the assays the activity scores are 0 for inactives, the same low value for inconclusive compounds, and 100 for actives. This 3 value column may not be as informative as a continuous raw signal. We process each target's assay file individually into a column vector, and then merge the target columns into one matrix. The binary bioactivity matrix can be easily constructed by treating Inconclusive compounds as Inactive. For the continuous bioactivity matrix, we use the PUBCHEM_ACTIVITY_SCORE directly due to no clear alternative. The constructed bioactivity matrix has a wide range of missing values per target column. Table 4.8 details the size, hit %, and % missing values for the 128 targets.

Thus, the **goal** of this experiment is to evaluate the performance of four informer set methods on 10 of the 128-PCBA targets. These 10 targets were selected based on their size, hit %, and % missing values as seen in Table 4.7. The size was restricted to targets with less than 75,000 compounds due to resource and time limitations. We set the informer set size to 5% of the missing target's size since this is similar to the informer set size used for PKIS1 (i.e. 16 out of 366 compounds). Note that assays aid884 and aid885 have the same target and protocol, but aid884 designates inhibitors as those with low measured signal, whereas aid885 designates activators as those with high measured signal. That is, a hit in aid884 are inhibitors (low signal), whereas a hit in aid885 are activators (high signal). The target class and targets were obtained from Ramsundar et al. [4], and showcase that these are diverse targets. Evaluation uses the same metrics introduced in section 4.6: AUC[ROC], $NEF_{10\%}$, and $FASR_{10\%}$. We do not use hits in the top 10% since the targets have different compound sizes. Similarly to PKIS1, RDKit's MurckoScaffold was used to define compound scaffolds [1, 114].

Finally, there is an important consideration when running these methods on a missing target with missing labels. Recall that the entire 128-PCBA matrix has 439,877 compounds and 128

targets. When running an informer set method on a missing target with, say 10,000 non-missing compound labels, we only run the method on the submatrix consisting of those 10,000 compounds and 128 targets. This is done due to not knowing the labels for the other missing compounds for the missing target, and consequently, if those missing compounds were selected as informers, then we cannot make use of them since we don't know the labels. In practice, one would restrict informer selection to the compounds that are available for screening.

Table 4.7: Summary of the 10 targets from 128-PCBA used in the evaluation of the four informer set methods: CustomMC, RFSupervised, and MTNNSupervised. Target and target class were obtained from Ramsundar et al. [4]. As mentioned in the main text, aid884 and aid885 have the same target and protocol but differ in hit definition. aid884 defines hits as those with low measured signal (inhibitors), whereas aid885 defines hits as those with high measured signal (activators).

Target Assay ID	Hits	Size	% Hits	% Missing	Informer Set Size	Target Class	Target
aid899	1693	9093	18.62	65.81	455	other enzyme	CYP2C19
aid884	3348	13075	25.61	62.72	654	other enzyme	CYP3A4
aid885	158	13075	1.21	62.72	654	other enzyme	CYP3A4
aid720532	887	15471	5.73	74.61	774	miscellaneous	Marburg virus
aid493208	342	43989	0.78	56.39	2200	protein kinase	mTOR
aid904	528	54509	0.97	48.23	2726	viability	H1299-neo
aid925	39	64405	0.06	21.91	3221	miscellaneous	EGFP-654
aid411	1554	71297	2.18	20.59	3565	other enzyme	luciferase
aid938	1765	72024	2.45	22.14	3602	ion channel	CNG
aid875	32	73910	0.04	22.87	3696	protein-protein interaction	brca1-bach1

Table 4.8: Summary of the 128-PCBA targets in terms of size, hits, and % missing values.

Target Assay ID	Hits	Size	Hit %	% Missing
aid883	1161	9093	12.77	65.81
aid891	1489	9093	16.38	65.81
aid899	1693	9093	18.62	65.81
aid915	415	10838	3.83	52.85
aid914	217	10838	2.00	52.85
aid884	3348	13075	25.61	62.72

Continued on next page

Table 4.8: Summary of the 128 PCBA targets size, hits, and % missing values.

Target Assay ID	Hits	Size	Hit %	% Missing
aid885	158	13075	1.21	62.72
aid720532	887	15471	5.73	74.61
aid493208	342	43989	0.78	56.39
aid904	528	54509	0.97	48.23
aid903	338	54513	0.62	48.14
aid927	59	59169	0.10	19.37
aid925	39	64405	0.06	21.91
aid912	433	68874	0.63	22.25
aid995	692	70896	0.98	23.04
aid411	1554	71297	2.18	20.59
aid926	344	72024	0.48	22.14
aid938	1765	72024	2.45	22.14
aid887	1008	73172	1.38	22.62
aid875	32	73910	0.04	22.87
aid881	584	107253	0.54	21.66
aid1454	511	121333	0.42	23.87
aid924	1134	124021	0.91	34.33
aid902	1847	125392	1.47	33.71
aid1452	175	151819	0.12	20.58
aid1379	558	197903	0.28	20.82
aid2242	715	199174	0.36	18.35
aid1457	719	205589	0.35	21.59
aid1458	5762	206982	2.78	24.00
aid1030	15856	216489	7.32	21.85
aid1688	2367	220576	1.07	19.45
aid1461	2299	220868	1.04	19.14
aid1471	278	223619	0.12	22.56
aid485360	1484	225315	0.66	24.04
aid2147	3381	226920	1.49	20.41
aid2549	1180	234925	0.50	22.17
aid1631	892	263668	0.34	21.55
aid1634	154	263668	0.06	21.55
aid1460	5578	265747	2.10	26.57

Continued on next page

Table 4.8: Summary of the 128 PCBA targets size, hits, and % missing values.

Target Assay ID	Hits	Size	Hit %	% Missing
aid2326	1064	269258	0.40	23.35
aid1468	1024	271412	0.38	26.59
aid1479	778	273693	0.28	27.45
aid1469	165	273693	0.06	27.45
aid540276	4316	274192	1.57	27.46
aid2675	97	279434	0.03	22.80
aid2451	1987	286481	0.69	24.13
aid1721	1086	292738	0.37	22.71
aid2662	110	294066	0.04	23.04
aid2100	1155	302306	0.38	24.96
aid2546	10512	304068	3.46	24.88
aid2551	16627	305447	5.44	24.82
aid720579	1883	305982	0.62	35.84
aid485297	9123	320610	2.85	24.52
aid485313	7565	320689	2.36	24.52
aid504706	201	321433	0.06	24.91
aid2101	279	321562	0.09	25.75
aid485349	618	322365	0.19	25.35
aid720580	1485	325897	0.46	34.60
aid485353	602	328647	0.18	25.13
aid652105	4069	328848	1.24	30.61
aid504842	101	329574	0.03	25.21
aid504467	7644	330113	2.32	25.14
aid504466	4166	330115	1.26	25.14
aid463254	41	330683	0.01	25.20
aid485367	556	330683	0.17	25.20
aid492947	79	330683	0.02	25.20
aid485341	1728	330683	0.52	25.20
aid485314	4467	334467	1.34	27.18
aid624287	423	334813	0.13	26.51
aid651965	6297	336348	1.87	33.59
aid624288	1354	337435	0.40	25.98
aid485281	249	341509	0.07	28.11

Continued on next page

Table 4.8: Summary of the 128 PCBA targets size, hits, and % missing values.

Target Assay ID	Hits	Size	Hit %	% Missing
aid720553	3259	342072	0.95	26.96
aid485290	925	342200	0.27	29.95
aid624297	6176	342266	1.80	26.35
aid2528	638	343093	0.19	28.98
aid624296	9792	343221	2.85	26.33
aid720551	1263	343654	0.37	27.00
aid2517	1106	343726	0.32	27.12
aid504332	30052	344607	8.72	29.33
aid624291	222	345843	0.06	27.00
aid485364	10677	356652	2.99	28.39
aid504333	15634	356837	4.38	28.30
aid588342	24994	360862	6.93	28.24
aid504444	7325	360868	2.03	29.48
aid651644	747	361865	0.21	29.29
aid485294	148	362206	0.04	28.29
aid2676	1069	362207	0.30	28.29
aid651768	1673	363999	0.46	29.26
aid720504	10158	364053	2.79	29.25
aid720542	733	364084	0.20	29.25
aid686970	5937	364452	1.63	29.31
aid720707	268	364602	0.07	29.31
aid720711	290	364602	0.08	29.31
aid720709	515	364602	0.14	29.31
aid720708	661	364602	0.18	29.31
aid652025	238	364605	0.07	29.31
aid624246	101	366962	0.03	29.82
aid652106	495	368606	0.13	29.96
aid624202	3959	376017	1.05	30.20
aid602233	165	381071	0.04	30.40
aid504327	733	381582	0.19	30.02
aid540317	2119	383355	0.55	29.74
aid504891	34	383688	0.01	29.96
aid602313	759	383840	0.20	30.39

Continued on next page

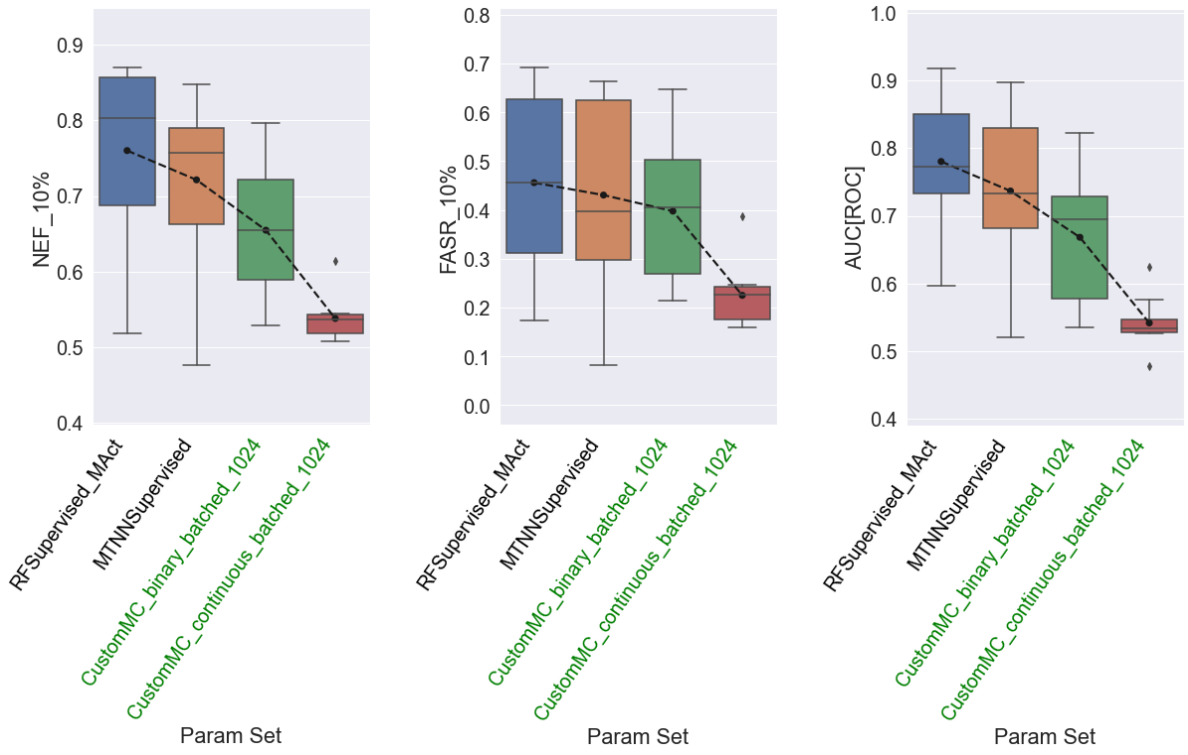
Table 4.8: Summary of the 128 PCBA targets size, hits, and % missing values.

Target Assay ID	Hits	Size	Hit %	% Missing
aid588453	3866	384115	1.01	30.21
aid504339	16808	384521	4.37	29.94
aid588590	3882	384784	1.01	30.11
aid588456	51	384990	0.01	30.13
aid588795	1292	384993	0.34	30.13
aid504845	92	385289	0.02	29.97
aid588591	4674	386369	1.21	30.32
aid743255	898	386843	0.23	30.39
aid602179	363	386990	0.09	30.19
aid588579	1946	391089	0.50	31.20
aid651635	3752	391566	0.96	31.51
aid504847	3435	392896	0.87	30.92
aid602310	309	402340	0.08	31.99
aid588855	4828	403336	1.20	32.07
aid652104	7071	403694	1.75	32.26
aid624171	1238	403862	0.31	32.05
aid624417	6366	405121	1.57	32.24
aid624170	826	405280	0.20	32.41
aid743266	303	405676	0.07	32.29
aid624173	485	405845	0.12	32.54
aid602332	68	410931	0.02	33.60
aid686979	48029	412836	11.63	33.75
aid686978	61848	412836	14.98	33.75
Mean/Std.	3665.02 / 8057.57	281434.76 / 121967.21	1.80 / 3.77	29.98 / 10.18

PCBA Leave-one-target-out Results

We ran the four informer set methods CustomMC (binary and continuous), RFSupervised, and MTNNSupervised on the 10 PCBA targets listed in Table 4.7. Figure 4.19 shows the boxplots of the four methods on the three metrics of interest. RFSupervised achieves the best mean and median performance on all three metrics. MTNNSupervised comes in at second place. The CustomMC

continuous variant had evidently worse performance which, as we alluded to earlier, is most likely due to the uninformative continuous scores for this dataset. Recall that CustomMC was among the top performers on PKIS1, while RFSupervised achieved the worst performance. This is not the case for these 10 targets.



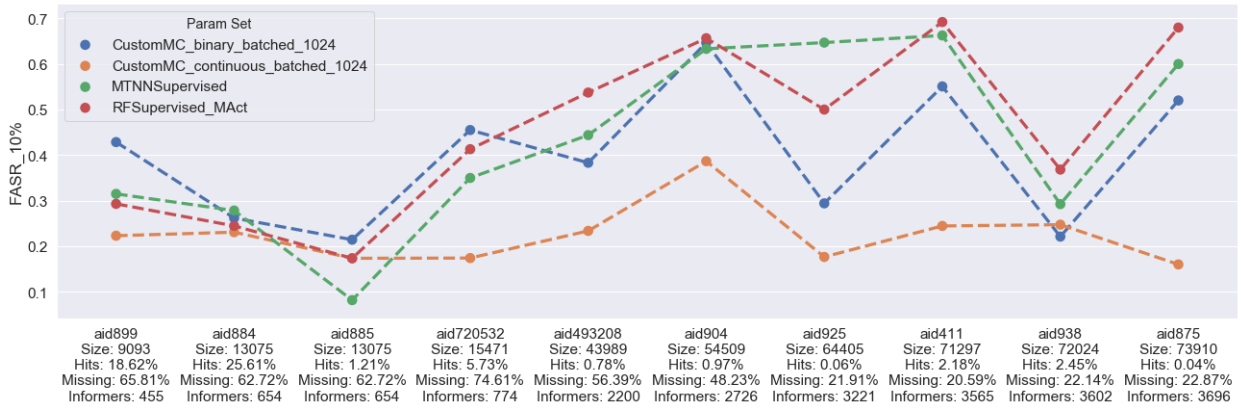
(a) $NEF_{10\%}$ (b) $FASR_{10\%}$ (c) AUC[ROC]

Figure 4.19: Boxplots and mean performance of CustomMC (binary and continuous), RFSupervised, and MTNNSupervised on the 10 PCBA targets in a leave-one-target-out fashion. The 10 targets are listed in Table 4.7. The metrics are (a) $NEF_{10\%}$, (b) $FASR_{10\%}$, and (c) AUC[ROC].

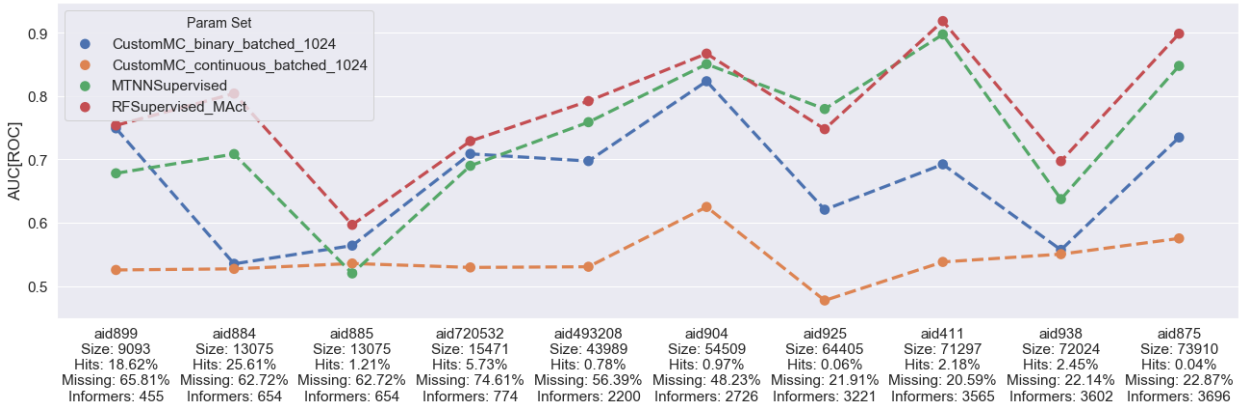
When the missing target has little relation to the known targets, we intuitively suspect that supervised methods that make use of compound features will do better than those that do not. If we look at the per target performance ranking in Figure 4.20, we can get more insight. The targets have been sorted on the x-axis in ascending order of compound size. Focusing on the $NEF_{10\%}$ metric, the RFSupervised achieves the best performance on 8 of the 10 targets. On target aid885, all methods struggle and barely achieve performance better than random (i.e. 0.5). This may be due to the combination of low hit rate and high missing percentage of 1.21% and 62.72%, respectively.

However, targets aid720532 and aid493208 also have low hit rates and high missing percentage but do not exhibit such low performance on all methods. Knowing the target-target similarity in some form like sequence similarity might be able to explain this. Unfortunately, we did not carry out this step as the targets are a diverse set of proteins, protein-protein interactions, enzymes, GPCRs, etc. Looking at the $FASR_{10\%}$, we see a different perspective. The binary CustomMC is competitive with RFSupervised, even outperforming it on 4 of the targets. Since CustomMC is based on the compound labels and makes no use of compound features, this may be the reason why it is able to achieve good hit diversity despite low hit counts [115]. In summary, the RFSupervised achieves the best performance on most of the targets. On targets aid899 ($FASR_{10\%}$) and aid925 ($NEF_{10\%}$), RFSupervised clearly does worse than the top performer.

(a) $\text{NEF}_{10\%}$



(b) $\text{FASR}_{10\%}$



(c) AUC[ROC]

Figure 4.20: **Per target** performance of CustomMC (binary and continuous), RFSupervised, and MTNNSupervised on the 10 PCBA targets in a leave-one-target-out fashion. The 10 targets are listed in Table 4.7. The metrics are (a) $\text{NEF}_{10\%}$, (b) $\text{FASR}_{10\%}$, and (c) AUC[ROC].

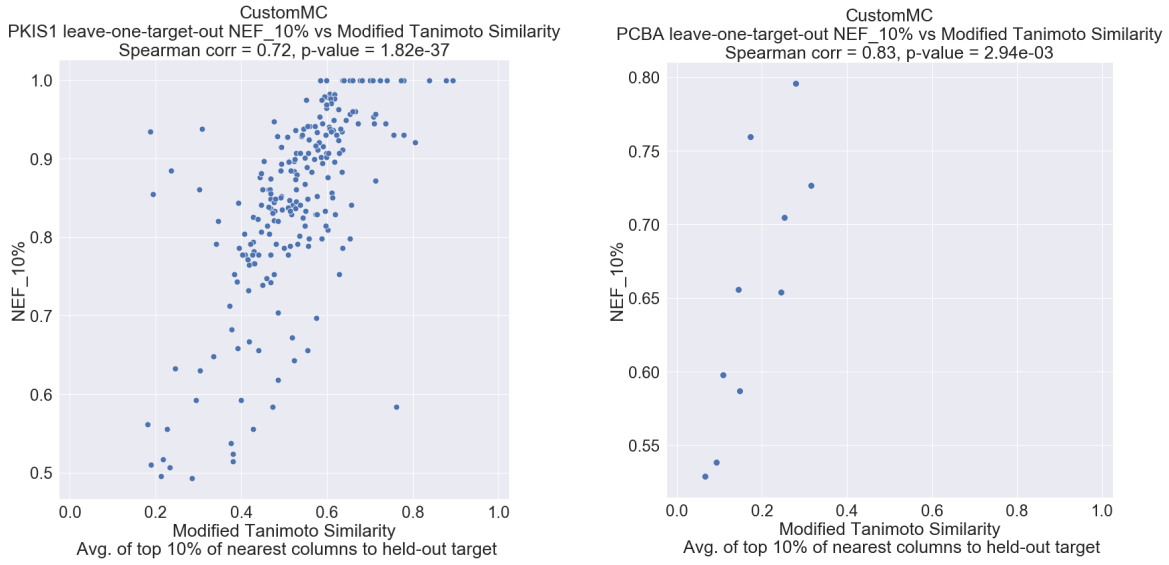
In a further attempt to explain the performance of CustomMC, we analyzed the relationship between $\text{NEF}_{10\%}$ and the column similarity of the missing target to the known targets. Of course in practice, the missing target is unknown, so this analysis is a post-hoc attempt at explaining why some tasks perform worse than others. Useful similarity measures between column bioactivity vectors should be done in the context and assumptions of the informer set method under analysis. Nonetheless, if we simplify the problem down to one missing target and one known target, then one useful measure is direct column Tanimoto similarity. That is, if the missing target has perfect Tanimoto similarity to the lone known target, then any reasonable method will achieve perfect performance. If there is more than one known target, then the information from every known target can help or hurt the informer set method depending on how similar these targets are to the missing target. In the context of CustomMC, it relies on a linear combination of the informers to predict non-informers. Thus, a helpful known target is one that has actives at the same index where the missing target has actives. As such, a useful similarity measure is a modified Tanimoto similarity that only takes into account agreement with the active indices of the missing target. Note that typical Tanimoto similarity takes into account agreement of *both* known and missing target active indices, whereas we exclusively look at missing target active indices. In mathematical terms, given bit vectors X and Y, the Tanimoto and modified Tanimoto computations are as follows (note the difference in the denominator):

$$Tanimoto(X, Y) = \frac{\sum_i (X_i \wedge Y_i)}{\sum_i (X_i \vee Y_i)}$$

$$ModifiedTanimito(X, Y) = \frac{\sum_i (X_i \wedge Y_i)}{\sum_i (X_i)}$$

Therefore, for a missing target, we compute this modified Tanimoto similarity to the known targets. Then we compute the average of the top 10% most similar known targets, to take into account the influence of more than one known target on CustomMC. Finally, we plot the relationship between $\text{NEF}_{10\%}$ and this average similarity. First, we revisit the PKIS1 results and look at this

relationship in Figure 4.21 (a). A noticeable trend is the positive relationship, indicating that increased similarity more than likely increases $\text{NEF}_{10\%}$ performance. The Spearman correlation for PKIS1 is a strong 0.72 with a significant p-value of 1.82×10^{-37}. Secondly, we showcase the plot for the 10 PCBA targets in Figure 4.21 (b). We see a similar trend of strong correlation between $\text{NEF}_{10\%}$ and similarity. However, note that the modified Tanimoto similarity range is much smaller for these 10 targets (maximum is 0.32). This is more than likely due to the dataset size differences between PKIS1 and PCBA targets (366 compounds versus $9,000 - 74,000$ compounds). As a result, larger bit vectors are more likely to have more disagreements. The Spearman correlation for the 10 PCBA targets is a strong 0.83 with a significant p-value of 2.94×10^{-3}. Again much of this is an attempt to explain the target performance of CustomMC because of the shift in its performance ranking in PKIS1 and PCBA. As a final note, this correlation was established using this modified similarity measure which in turn was constructed based on how CustomMC operates. Consequently, one can conceive of other measures that can strengthen or weaken this correlation.



(a) PKIS1 224 Targets

(b) PCBA 10 Targets

Figure 4.21: Plots showcasing the missing target relationship between CustomMC's $\text{NEF}_{10\%}$ and the average *modified* Tanimoto similarity of the top 10% nearest known targets. These two plots represent the leave-one-target-out results for (a) PKIS1 224 targets and (b) PCBA 10 targets. Note that this modified Tanimoto similarity is based on the bioactivity column vectors and only takes into account the active indices of the missing target.

4.8 Lessons Learned and Future Work

We defined the informer set problem in the same manner as that described in Zhang et al. [2]. That is, an informer set method performs two steps: selects k informers and obtains their bioactivity labels for the missing target, then predicts the bioactivity of the remaining non-informers towards the missing target. Defined in this way, we limit the set of viable solutions. That is, we don't consider three step or multi-step solutions. However, such limitations are justified in the context of drug screening due to scheduling and costs. Nonetheless, we continued with such a constrained definition and considered possible methods for solving the problem. Furthermore, this explicit constraint and definition provides formalism from which we can proceed.

In this chapter, we discussed 21 informer set methods that incorporate concepts from machine learning, active learning, core-set selection, matrix completion, matrix factorization, and recommender systems. One goal was to explore the efficacy of such concepts. Although the BOISE method proposed by Yu et al. [86] handles missing data, they have not applied it to larger datasets as it does not scale well. Thus, another goal was to apply scalable methods on a larger dataset with missing values like 128-PCBA [5].

The first experiment on PKIS1 was run on all 21 methods in a leave-one-target-out fashion. This dataset is small and serves as a toy dataset for informer set methods. An interesting result is that methods that incorporate target-target sequence similarities outperform the same method variants that do not. In fact, the direct pairwise sequence similarity method is among the top performers on PKIS1. This advocates the use of such similarities if there is prior knowledge that the targets come from the same family like in PKIS1. The CustomMC method is especially interesting since it is the best performing method that makes no use of sequence similarities, operates only on the bioactivity matrix, and can handle missing values. Methods that use supervised learning classifiers and compound features like CISP, MTNNSupervised, and RFSupervised did no better than the baselines from Zhang et al. [2]. However, intuitively, we suspected that there is value in such methods since they can adapt to a bad informer set selection by falling back on what the classifier learns from the informers. In contrast, methods like CustomMC make a bold assumption that the

missing target shares the same row weights with the known targets. These row weights are selected along with the informers, and so, there is no adjustment being made after selecting the informers.

To this end, we applied CustomMC, MTNNSupervised, and RFSupervised on ten targets from 128-PCBA. This dataset is much larger than PKIS1, and has a considerable percentage of missing values. The ten targets range from 9,000 to 74,000 compounds. CISP was not applied to this dataset due to the considerable computation costs in running it, and was replaced with MTNNSupervised since they are both based on concepts from core-set selection. The results on the ten PCBA targets was in direct opposition to that of PKIS1. RFSupervised had the best mean performance on all metrics of interest, followed by MTNNSupervised. Due the varying sizes of the targets, a better comparison would be to look at the per target performance of these methods. RFSupervised achieves the best $\text{NEF}_{10\%}$ performance on 8 of the 10 targets. Interestingly, on the $\text{FASR}_{10\%}$ which judges diversity, RFSupervised performs the best on 5 of the 10 targets. However, for two targets all three methods achieve relatively similar performance.

The PCBA experiment also revealed the computational and resource considerations that must be taken into account. The BOISE method proposed by Yu et al. [86] is based on a satisfying statistical framework, however it took 300 CPU hours to run for one PKIS1 target (366 compounds and 223 targets). CustomMC scales much better as we were able to run it in less than 18 hours for the ten PCBA targets (9,000 to 74,000 compounds and 127 targets). However, the memory costs are quadratic, as the weight matrix W is $m \times m$ where m is the number of compounds. In order to speed up computation during gradient descent, it is preferred to keep these weights in RAM, rather than on disk. In contrast, the supervised learning method RFSupervised which showed the best performance on the ten PCBA targets and scales well in terms of space and time. The base learner used is the random forest, whose memory usage will be typically dominated by the large dataset, rather than random forest parameters.

The PKIS1 dataset is helpful as a toy dataset for building informer set methods, but should not be extrapolated from to more complex datasets like PCBA. Informer methods that rely on just the bioactivity matrix are at a disadvantage because they have to make strict assumptions on how to

complete the missing column with very little room for adjustment. Once the informers are selected, methods like CustomMC use the informer labels to score the non-informers using fixed weights, i.e. there is no adjustment made from knowing the informer labels. If those assumptions are met, which in practice is not guaranteed, then they can perform very well. In contrast, supervised learning methods that rely on compound features and target-target similarities can make use of such extra information to select informers, then adjust their scoring of non-informers by learning the relationship between compound features and informer labels. In the experiments, CustomMC performed very well on the related PKIS1 human kinase targets, but performed worse than RFSupervised on the ten diverse PCBA targets. This is again most likely due to the heavy reliance of CustomMC on the relatedness of targets and the bioactivity matrix. Thus, PKIS1 and PCBA experiments represent contrasting settings in terms of target relatedness (i.e. high relatedness versus low relatedness). In practice, given a missing target of interest, it would be beneficial to *search* for available data on related targets via online resources like PubChem or ChEMBL [5, 85].

For future work, one avenue to explore is to consider the application of informer set methods in practice, and potential alternatives. In the context of our definition, an informer set method serves as a complete two-round screening campaign. That is, the first round selects k informers to screen which provides an initial dataset. The second round, utilizes the first round's dataset to select non-informers for a second screen. The goal is to maximize a measure of interest like hits or diversity. The methods developed in this chapter attempt to do this, but they make use of the bioactivity matrix of known *related* targets. We did not consider the alternative options that ignore this bioactivity matrix. For example, earlier in chapter 1 we mentioned that an initial dataset can be gathered using high-throughput screening (e.g. diversity sampling). Subsequently, we can train a supervised learning method on this initial dataset, then use it to select the remaining compounds. This is similar to what was done in the ORS case studies in chapters 2 and 3. We can extend this further by considering the effect of informer set size on method performance in such two-round screening scenarios.

— 5 —

Iterative Batched Screening Strategy – Cluster-Based Weighted-Selector (CBWS)

5.1 Iterative Batched Screening (IBS) Problem Overview

Recall our definition of iterative batched screening (IBS) in section 1.4. We reiterate the definition here with a focused goal in mind. We have a target of interest with an initial set of compound-target bioactivity data T_0. The plan is to conduct a series of in vitro screens guided by a **compound selection strategy**. Each in vitro screen consists of screening B compounds. In practice, compounds are screened in plates typically with $96, 384,$ or 1536 wells. The strategy selects compounds from a *large* pool of untested compounds: U. In practice, this pool typically consists of compounds available for purchase from a vendor or maintained in stock at the screening facility. Vendors can synthesize custom orders, but this is expensive, and it is cheaper to bulk-order a pre-plated collection of compounds. However, in this project, we ignore cost considerations for now and defer its incorporation for future work. We plan to publish the results of this chapter in the future[1].

First, we define the input and output involved in an IBS campaign. Let $T_0 = (T_0^X, T_0^Y)$ be the initial dataset of compound features T_0^X and compound-target bioactivity T_0^Y. Let $U_0 = (U_0^X)$ be the untested pool of available compounds with features U_0^X; labels U_0^Y are **unknown**. Further note that $U_0 \cap T_0 = \emptyset$. Let the batch size for each screening iteration be $B \in \{96, 384, 1536\}$ (this

[1] Anticipated authors: Moayad Alnammi, Spencer S. Ericksen, Scott A. Wildman, Nathan Wlodarchak, Hunter Reis, Troy King, Song Guo, Gene E. Ananiev, and Anthony Gitter.

corresponds to standard plate sizes). Let the maximum number of screening iterations be $\tau_{max} \in \mathbb{N}$.

Consider an **IBS compound selection strategy** $S(T_i, U_i, B)$ that takes as input the ith iteration's T_i, U_i, and B, then returns a set Z of B compounds selected from U_i. Note that the features Z^X are known, but the labels Z^Y are unknown. Thus, the compounds in Z are screened in vitro to produce the labels Z^Y. The next iteration's datasets are updated by adding Z to the training set and removing Z from the unlabelled pool.

$$T_{i+1} = T_i \cup Z = (T_i^X \cup Z^X, T_i^Y \cup Z^Y)$$
$$U_{i+1} = (U_i^X \setminus Z^X)$$

With the needed terms defined, we now define the goal of this project:

Definition 5.1: Iterative Batched Screening Strategy Development Goal

The **goal** is to **develop** a selection strategy that **maximizes the number of hits and hit diversity** over τ_{max} iterations. Denote STRATS as the set of all IBS strategies that take in T as a training set, U as an unlabelled pool, and B as the batch size and selects the set Z. Thus, in set-notation this is:

$$STRATS = \{S \quad | \quad S : T \times U \times B \to Z \text{ and } |Z| = B\}$$

Then, the optimization goal is to find the strategy that maximizes hits and diversity over τ_{max} iterations:

$$S^* = \underset{S \in \text{STRATS}}{\arg\max} \; \alpha\text{HITS}(\bigcup_{i=0}^{\tau_{max}} S(T_i, U_i, B)) + (1-\alpha)\text{DIVERSITY}(\bigcup_{i=0}^{\tau_{max}} S(T_i, U_i, B)) \quad (5.1)$$

Here HITS$(\cdot)$ is the number of hits in the input compound set, DIVERSITY$(\cdot)$ measures the chemical diversity of the hits in the input compound set, $\bigcup_{i=0}^{\tau_{max}} S(T_i, U_i, B)$ are all the compounds selected by the selection strategy S over τ_{max} iterations, and $\alpha \in [0, 1]$ is the trade-off between maximizing hits and diversity. As we will discuss later, we measure diversity by clustering the compounds using fingerprint Tanimoto similarities. To summarize, the steps taken by any strategy S is as follows:

1. Starting from T_0, U_0, and $i = 0$.

2. Select compounds $Z = S(T_i, U_i, B)$.

3. Set $i := i + 1$ and update datasets: T_{i+1} and U_{i+1}.

4. Repeat steps 2 and 3 while $i < \tau_{max}$. At the end, return $T_{\tau_{max}}$.

5.2 Solving the IBS Goal

With the way the problem is set up in definition 5.1, the maximum number of hits **retrievable** is determined by U_0^Y, B (batch size), and τ_{max} (max iterations). Although the untested pool labels U_0^Y are unknown, an optimal strategy S^* will retrieve the minimum between the number of hits in U_0^Y and $B\tau_{max}$. That is, the optimal strategy cannot retrieve more hits than what is in the pool. For example, if the number of hits in the unlabelled pool is 100, $B = 10$, and $\tau_{max} = 5$, then an optimal strategy can only retrieve at most 50 hits since it can only select 50 compounds in 5 iterations.

$$\text{HITS}(\bigcup_{i=0}^{\tau_{max}} S^*(T_i, U_i, B)) = \min(\text{HITS}(U_0^Y), B\tau_{max})$$

Furthermore, the set of strategies STRATS is on the order of $O(\tau_{max}\binom{|U_0|}{B})$, so brute force exploration is not feasible. Thus, we opt towards a data-driven strategy that combines exploitation-exploration concepts from active and reinforcement learning. This is not guaranteed to find an optimum solution, but is hoped to find a *reasonably good* solution. In this chapter, we dedicate the early sections to discussing the proposed iterative strategy: cluster-based weighted-selector (CBWS). The strategy was developed with the strengths of ligand-based VS in mind and incorporates exploitation-exploration concepts from machine learning. Later on, we conduct experiments where we introduce other strategies and benchmarks that were adapted from reinforcement learning.

5.3 Related Work

In 2015, Reker et al. [36] provided an overview of active learning strategies applied to virtual screening. In it they explained the active learning pipeline in the context of screening and discussed eight studies in this domain. Furthermore, they also put forth the idea of exploitation vs. exploration as measured by a model's hit predictions and uncertainty, respectively. The main arguments for exploration is that it enriches the model with novel/diverse compounds that it has never seen before. One of the first studies discussing active learning in drug discovery was done in 2002 by Warmuth et. al [116]. The batch size was set to be 5% of the total dataset (training and unlabeled pool), and if there are no actives in the initial training set, initial batches were selected at random until an active is found. The goal in that study was also to maximize the number of hits over the tested compounds. Two datasets were used that are relatively small compared to today's vendor catalogs: $Round0$ with $1,316$ compounds (39 actives) and $Round1$ with 634 compounds (150 actives). This corresponds to batch sizes of around 66 and 32, respectively. Typical plates are able to accommodate 96, 384, and 1536 compounds, so this batch size is smaller. They compared fully exploitative vs. explorative strategies using an ensemble of perceptrons and a support vector machine (SVM). Their conclusion was that exploitative strategies were suitable for maximizing hits, but explorative strategies are helpful for model generalization.

A study in 2008 by Fujiwara et al. [117] used their adapted version of a query-by-committee (QBC) algorithm for measuring uncertainty. Three datasets on three targets were used: 299642 (1515 actives), 299751 (182 actives), and 299297 (46 actives). Two batch sizes of 200 and 2000 were used. They compared their two QBC methods against a random-selector and 1-nearest neighbor baseline. The QBC methods outperformed the baseline in hit-finding. One of the QBC methods was applied to a prospective dataset with a training set of 20,000 compounds and an unlabeled pool of 52,657 compounds. Another study in 2008 by De Grave et al. [118] used a Gaussian process model with a number of exploitative strategies along with random-selection and an explorative strategy. Their goal was to determine the hit-finding capability of these strategies. As expected, exploitative strategies showcased better hit-retrieval against non-exploitative strategies, but among themselves,

there was no clear significance. A study in 2013 by Ahmadi et al. [119] compared their exploitative strategy based on global optimization and a Gaussian process against other simpler exploitative strategies (like maximum prediction) to get similar results. In 2013, Desai et al. [120] proposed an automated pipeline of virtual selection of compounds followed by a robotic-controlled physical screening. The results of the physical screens are then fed back into the virtual selection component for the next iteration. Using a batch size of 1 they tested three strategies: full-exploitation based on maximum prediction, full-exploration based on undersampled compounds, and a combined strategy of alternating between rounds of exploration and rounds of exploitation.

In 2016, Paricharak et al. [115] using batch sizes of 50, 100, and 200 proposed an iterative screening selection strategy based on structural similarity and biological activity similarity. Structural similarity was measured using ECFP4-like fingerprints and Tanimoto distance, whereas biological similarity was measured in terms of multi-target bioactivity fingerprints of the compounds. Compound selection using biological activity similarity yielded more diverse active compounds than structural similarity. In 2017, Svensson et al. [121] applied an iterative screening strategy to 41 datasets using conformal predictors [122]. The interesting addition, along with conformal predictors, was the use of docking to generate the initial dataset for the iterative screening plan. The strategy is exploitive and the conformal predictor is able to determine confidence of each compound allowing varying batch sizes for each iteration (i.e. only screening highly active, highly confident compounds).

In 2018, Cortés-Ciriano et al. [123] proposed an iterative strategy with the goal of identifying a single, highly active compound from a pool using an initial training set of only lowly active compounds. The setting and goal are different than what this project is aiming for, but their method may help in the case when no actives are present in the initial training set. In 2019, Miyao et al. [124] performed an iterative screening comparison study on four methods: a genetic algorithm based partial least-squares regressor [125, 126], Bayesian-Optimization with a Gaussian process [127, 128], batch-based version of Bayesian-Optimization with a Gaussian process [129], and Support Vector Regression (SVR) [130]. Two comparison studies were performed based on two

goals: to find compounds with maximum/minimum values, or to find compounds with values in an interval. For both goals, a batch size of 10 and max iteration of 40 was used. The selection of compounds was determined by the distance of the predictions of the models (some of the models like GP perform implicit exploration via their prediction range for a single compounds).

In 2019, Buendia et al. [131] had the goal of estimating the number of hits for the next iteration in an iterative screening campaign using Venn-ABERS predictors [132]. This knowledge can then be used to decide when to end a campaign; i.e. the max number of iterations. They used a fixed batch size of $50,000$ compounds for each iteration for a maximum of 10 iterations (this same batch size was used in a 2015 study by Maciejewski et. al [133]). Such a batch size is relatively large for an academic lab with a limited budget. The algorithm was performed on six large datasets ranging from 1,691,877 to 2,017,739 compounds (with hit percentages in the range of 2.83% to 0.34%). The selection strategy was the highest $50,000$ predictions from the Venn-ABERS predictor which uses a Support vector machine (SVM) learner. The method showed promising results in predicting the number of hits in the next iteration.

In 2019, Jansen et al. [134] proposed a workflow for selecting diverse compounds from a pool with a focus on promoting quality of hits. The workflow is called Biased Complement Diversity (BCD) as it can be used to select a secondary diverse set that complements a primary set of compounds. These primary compounds are typically compounds of interest that are predicted to be active by some method like docking. The workflow takes in a prospective pool of compounds, and if available, a set of primary compounds of interest, and perform three main steps. First, it clusters all the compounds and classifies them into four quality tiers (A, B, C, and D). The quality is determined by user defined criteria like molecular weight and lipophilicity ranges. Second, it begins the selection process by looping on all clusters. For each cluster, it determines the number of compounds to select based on a user defined variable, and then selects compounds from tiers A, then B, and then C until the cluster budget is exhausted. The selection of compounds in a single tier is done in a diverse manner based on fingerprints. If a primary set is given, then only compounds in this set are of tier A, and thus, are automatically selected. In the end, at least one compound is

selected from each cluster and each tier. The workflow was tested on four targets with prospective pools of 832K, 1.157M, 736K, and 306K. Three of the four targets had a primary set. The BCD workflow was given the prospective pool and primary set, and returned a selected set of compounds (which included the primary set as well). These selected set of compounds were screened and hits were confirmed. The primary sets yielded higher hit rates than their complement sets, but interestingly, the scaffold overlap between the two sets was small. Furthermore, the BCD workflow enriched selection of hits with higher quality (A, B, C, and D). This was more evident when BCD was compared with two benchmarks: random sampling and diversity sampling. However, it is also interesting to note that although BCD found higher quality A hits than the two benchmarks, it found less total hits; i.e. most of the hits found by the benchmarks were of D quality. As a final note, it is possible to run the BCD workflow in an iterative manner as an exploration helper. For example, the primary set would be selected by an external exploitive strategy, and the diverse set would be selected by BCD. However, the batch size is dependent on the clustering and is significantly larger in their experiments than what we use in this chapter (i.e. batch sizes of 96, 384, and 1536).

5.4 Significance of Work

A completely active learning approach based on uncertainty focuses on improving the quality of the model's generalization. This may or may not maximize the number of hits and diversity on the untested pool over iterations. This is because uncertainty will favor selection of compounds that the model is unsure about which is akin to exploration, and so, likely will not be hits. An iterative screening campaign might desire a certain number of hits over a fixed number of subsequent iterations. This becomes more probable if we expend some of the budget towards exploitation. Thus, a strategy combining exploration and exploitation is suitable for the iterative screening scenario.

The exploration-exploitation dilemma is a staple discussion in reinforcement learning (RL). However, for the IBS case, the formulation into the RL framework is somewhat challenging. This is particularly evident when we consider what represents states and actions. A state corresponds

to the current iteration's training set, untested pool, and any other additional information based on these two datasets like uncertainty and activity as predicted by a model. An action corresponds to the selected compounds from the current iteration's untested pool (there is an additional constraint that the number of selected compounds must be equal to the budget B). The reward is given at the end of τ_{max} iterations in the form of the number of hits and diversity (i.e. the problem is episodic). Due to the sheer number of states and actions determined by the training set, untested pool, and budget, this may require condensed representations to be applicable to a wide range of target datasets. Furthermore, even with condensed representations of state and actions, it would require a large infrastructure commitment to train the RL models (via value-based approximation or policy gradient methods). For this reason, we opt towards a strategy that is guided by virtual screening successes and standards.

In agreement with Jansen et al. [134], we incorporate clustering in our selection process. Clustering can provide a high level view of the chemical landscape, and furthermore, help localize search within clusters. Clustering is used heavily in drug discovery to measure diversity, thus we incorporate it in the decision process. However, we explicitly attach value attributes to each cluster that provide summary statistics of the compounds within the cluster. These attributes influence whether a cluster is selected for exploitation, exploration, or neither. While Jansen et al. selects at least one compound from each cluster, our selection process will select from the few informative clusters.

The proposed strategy is highly customizable. The software framework allows to easily modify: the supervised learning algorithm, uncertainty measure, the clustering properties, the weight equations, compound distance functions, and selection strategy between extremes of exploitation and exploration. As an example, the CBWS can be easily changed to an instance-based weighted-selector by treating each instance as belonging to its own unique cluster. Another example, the CBWS algorithm can be set to a fully exploitative cluster selector as a special case. As a byproduct of the project's implementation, the software framework can easily be extended to implement new user-defined strategies. That is, users can easily create their own selection strategies via inheritance

and apply their own logic. The framework can run simulations of an iterative screen using the user's implemented selection strategy and benchmark datasets (or user defined dataset). With such a framework, one can develop a strategy and benchmark across a wide range of target datasets to measure robustness. The benchmarks and comparison strategies introduced in experiments were implemented under this framework.

The **significance of the proposed** CBWS iterative screening strategy is as follows:

- To our knowledge, no iterative screening strategy has tried splitting the batch size budget between exploitative and explorative compounds *within* the same iteration.
- The way clustering is incorporated is unique as it is used to provide a high-level overview of the chemical space and the model's learning. The cluster properties are updated every iteration and are used to decide which parts of the chemical space to exploit or explore.
- The strategy exposes a large number of parameters for easy customization and rapid development. These parameters were defined based on experience in ligand-based virtual screening.
- For a lab with limited means and funds, the batch size in previous studies are either very small (1 compound per iteration) or very large (>50k compounds). To this end, we explore batch sizes corresponding to the typical plate sizes of 96, 384, and 1536 wells. Furthermore, this fixed batch size alters the problem significantly, as strategies must carefully measure exploiting and exploring certain clusters over others due to the limited budget space.
- We run CBWS under various parameter settings through a rigorous set of experiments. We also compare CBWS against benchmarks adapted from standard reinforcement learning algorithms. In total, 109 target datasets are tested in an iterative setting, with one target being conducted in a prospective manner.
- The implementation framework allows for easy addition of custom strategies. This is exemplified by our addition of benchmarks to the framework that differ from CBWS.

5.5 Description of CBWS

The cluster-based weighted-selector (CBWS) algorithm clusters the compounds and assigns exploration and exploitation weights to each cluster. Compounds are clustered based on the feature similarity measures, most common being fingerprint features and Tanimoto/Jaccard similarity. Based on the budget and weights, a decision is made on which compounds are selected as exploitation and exploration. There are two reasons for clustering the compounds. First, due to the **similar property principle** [52], which roughly states that structurally similar compounds are likely to have similar properties, clusters for which we have good **coverage** can be exploited with more confidence than less covered clusters. Second, clustering compounds allows us to measure the **diversity** or **novelty** within and among clusters. Any clustering algorithm can be used with a distance measure based on similarity features. In this project, we use 1024-bit Morgan fingerprint [55, 1] features with Tanimoto distances and clustering is done using a custom implementation of the Taylor-Butina algorithm [80, 81].

Taylor-Butina clustering was developed for compound clustering, particularly with fingerprint features, but it can be used with any distance measure. It tries its best to assign compounds that are within a cutoff distance of each other to the same cluster. The clustering is performed on **all** compounds in the training and unlabeled pools: T_0^X and U_0^X. Unfortunately, available open source implementations of Taylor-Butina are done completely in RAM. This significantly limits the size of the datasets that can be clustered. Since we would like to scale our CBWS algorithm to datasets with millions of compounds and we favor the use of Taylor-Butina, we developed a custom implementation that stores intermediate computations on disk. Correctness of the implementation was verified by comparing clustering results with RDKit's `rdkit.ML.Cluster.Butina` module on small datasets [1].

The cluster is the main entity in CBWS, and we define the following properties for each cluster C_i:

- N_i: Number of compounds in cluster C_i.

- $N_i^{labeled}$: Number of labeled compounds in cluster C_i.

- $Density_i$: The fraction of compounds in cluster C_i.

$$Density_i = \frac{N_i}{|T_0^X| + |U_0^X|}$$

- $Coverage_i$: The fraction of labeled compounds in cluster C_i.

$$Coverage_i = \frac{N_i^{labeled}}{N_i}$$

- $Uncertainty_i$: The mean uncertainty of the compounds in cluster C_i. The uncertainty of a compound will be discussed later, but for now the uncertainty of a compound is estimated by a supervised learning model.

- $Activity_i$: The mean of the predicted hit probabilities of compounds within cluster C_i. This only considers compounds with a hit probability greater than a threshold probability $\tau_{activity}$. This is because we want to exploit only those compounds which exhibit high hit probabilities, and so, clusters with a few exploitative compounds do not get down-weighted by non-exploitative compounds. Compound hit probabilities are estimated by a supervised learning model.

Using these cluster properties we define **cluster exploitation and exploration weights** as $W_{i\_exploit}$ and $W_{i\_explore}$, respectively. These weights are used to impose qualification and priority over the clusters. The cluster weights change as we gather more data over the screening iterations. We want to favor exploiting clusters with high activity and high coverage of labelled data. Conversely, we want to favor exploring clusters with high uncertainty and low coverage. This is illustrated in the contrived example in Figure 5.1. We express these desires via the following convex

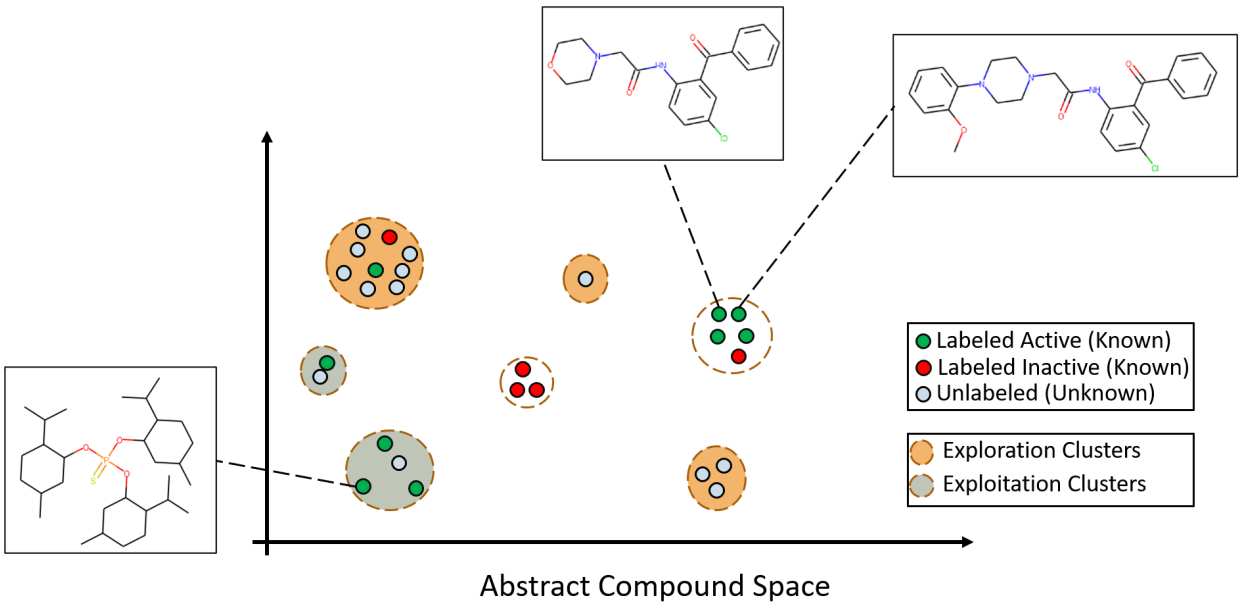


Figure 5.1: Contrived example illustrating an abstract 2D chemical space with exploitation and exploration clusters for the current iteration. The molecular graph of three compounds are depicted to illustrate that structurally similar compounds are closer in distance. Green and red circles denote active and inactive compounds that are known, respectively; these are used for training the machine learning model. Unknown unlabeled compounds are denoted as light blue circles. Finally, exploitation and exploration clusters are highlighted for the current iteration based on the machine learning model's predictions.

combinations:

$$W_{i\_exploit} = \alpha Activity_i + (1-\alpha)(Coverage_i Density_i) \tag{5.2}$$

$$W_{i\_explore} = \beta Uncertainty_i + (1-\beta)(1-Coverage_i) \tag{5.3}$$

Qualifying clusters are determined by a threshold for exploitation and exploration: $\tau_{exploit}$ and $\tau_{explore}$, respectively. Thus, we define qualifying exploration and exploitation clusters as:

$$E_{exploitation} = \{C_i : W_{i\_exploit} \geq \tau_{exploit}\} \tag{5.4}$$

$$E_{exploration} = \{C_i : W_{i\_explore} \geq \tau_{explore}\} \tag{5.5}$$

The next step is to **split the batch size** B between exploitation and exploration. We use a

preliminary estimate to be the ratio between the total compounds in $E_{exploitation}$ and $E_{exploration}$. For example, if the batch size is 96 and the total number of compounds in all the exploration and exploitation sets are 20 and 80, respectively, then the exploitation and exploration budgets are: 19 and 77, respectively. Denote the estimated exploitation and exploration budgets as: $B_{exploit}$ and $B_{explore}$, respectively (note that $B = B_{exploit} + B_{explore}$).

Now we can start **selecting from exploitation clusters** as follows:

1. Sort the clusters in $E_{exploitation}$ by the exploitation weight $W_{i\_exploit}$.

2. Select the first cluster to be the cluster with the highest exploitation weight in $E_{exploitation}$.

$$C_*^1 = \underset{C_i \in E_{exploitation}}{\arg\max} \; W_{C_i\_exploit}$$

3. Sample k compounds from C_*^1 according to the **Cluster Sampling Function** (see details in section 5.6). Update the remaining exploitation budget $B_{exploit} := B_{exploit} - k$.

4. Denote $S_{exploitation}$ as the set of selected exploitation clusters. Set $S_{exploitation} = \{C_*^1\}$. Update $E_{exploitation} = E_{exploitation} \setminus \{C_*^1\}$. Note that $E_{exploitation} \cap S_{exploitation} = \emptyset$.

5. For each cluster $C_i \in E_{exploitation}$ compute the average cluster dissimilarity against all the selected clusters in $S_{exploitation}$. Cluster-cluster dissimilarity is defined as the average distance between inter-cluster compounds. As a result, cluster dissimilarity is the average of cluster-cluster dissimilarities. Thus, for each cluster $C_i \in E_{exploitation}$, we compute cluster-cluster dissimilarity to each cluster $C_j \in S_{exploitation}$. Following this, for each $C_i \in E_{exploitation}$, we set $Dissimilarity_i$ to be the average of C_i's cluster-cluster dissimilarities.

6. Update exploitation weights to incorporate a diversity measure in the form of cluster dissimilarity.

$$W_{i\_exploit} := \lambda W_{i\_exploit} + (1 - \lambda) Dissimilarity_i \qquad \text{for all } C_i \in E_{exploitation}$$

7. Sort the clusters in $E_{exploitation}$ by the exploitation weight $W_{i\_exploit}$. Select the cluster with the highest exploitation weight C_*^j, sample compounds from this cluster according to the **Cluster Sampling Function**, and update exploitation cluster sets and budget.

$$S_{exploitation} := S_{exploitation} \cup \{C_*^j\}$$
$$E_{exploitation} := E_{exploitation} \setminus \{C_*^j\}$$
$$B_{exploit} := B_{exploit} - k \text{ where } k \text{ is number of compounds sampled from } C_*^j$$

8. Repeat steps 5, 6, and 7 until the exploitation budget is exhausted. Finally, return the sampled compounds.

We proceed similarly for exploration clusters:

1. Update exploration budget and exploration clusters to not include *selected* exploitation clusters.

$$B_{explore} := B - B_{exploit}$$
$$E_{exploration} := E_{exploration} \setminus S_{exploitation}$$

2. Sort the clusters in $E_{exploration}$ by $W_{i\_explore}$. Select the first cluster with the highest exploration weight.

$$C_*^1 = \underset{C_i \in E_{exploration}}{\arg\max} \; W_{C_i\_explore}$$

   Sample k compounds from C_*^1 using the **Cluster Sampling Function**. Update the remaining exploration budget $B_{explore} := B_{explore} - k$.

3. Set $S_{exploration} = \{C_*^1\}$. Update $E_{exploration} = E_{exploration} \setminus \{C_*^1\}$.

4. For each cluster $C_i \in E_{exploration}$ compute $Dissimilarity_i$ with respect to clusters $C_j \in S_{exploration}$.

5. Update exploration weights.

$$W_{i\_explore} := \lambda W_{i\_explore} + (1 - \lambda) Dissimilarity_i \qquad \text{for all } C_i \in E_{exploration}$$

6. Select the cluster with the highest exploration weight C_*^j, sample compounds using the **Cluster Sampling Function**, and update exploration cluster sets and budget.

$$S_{exploration} := S_{exploration} \cup \{C_*^j\}$$
$$E_{exploration} := E_{exploration} \setminus \{C_*^j\}$$
$$B_{explore} := B_{explore} - k \text{ where } k \text{ is number of compounds sampled from } C_*^j$$

7. Repeat steps 5, 6, and 7 until the exploration budget is exhausted. Finally, return the sampled compounds.

The sampled compounds are now designated as exploitation and exploration compounds. These compounds are passed to the in vitro screening team for scheduling, procurement, screening, and bioactivity data acquisition. The data is then incorporated in the training dataset for the next screening iteration's selection of compounds.

The **main takeaway summary of CBWS** is as follows:

- We use clustering to impose diversity over the set of compounds. Compounds within the same cluster are considered similar, i.e. close distance to each other. This allows us to get a high level view of the chemical space and make higher level decisions about which region of the chemical space to sample from.

- Cluster exploitation weight is used as a proxy to favor clusters with a large fraction of labelled data and strongly predicted active compounds. In other words, we want to exploit clusters that

have good coverage in terms of labelled compounds and the model finds promising activity for unlabelled compounds. Exploitation is important for retrieving more hit compounds.

- Cluster exploration weight is used as a proxy to favor clusters with high uncertainty and low local coverage. In other words, we want to explore clusters for which we have few labelled compounds or the model is uncertain about the compounds there. Exploration is important to ensure that the strategy is able to find other pockets of promising hit regions that are also diverse (from the known hit regions).

- To further ensure diversity, we re-weight clusters to include a diversity term after each selected cluster. The diversity term here is the cluster dissimilarity measure.

- Within cluster compound selection is done by a cluster selection function that selects dissimilar compounds within clusters as illustrated in Figure 5.2.

Finally, note that we have introduced a number of constant parameters like τ_{max}, α, β, $\tau_{exploit}$, $\tau_{explore}$, etc. One main goal of this project is to explore varying these parameters to find good parameter values across a wide range of target datasets.

5.6 Cluster Sampling Function

Each selected cluster is supplied with a cluster budget B_j that is proportional to the entire exploitation (or exploration) budget $B_{exploit}$ (or $B_{explore}$). In CBWS, instead of splitting the budget equally among qualifying exploitation clusters (or exploration), the budget is split **proportionally** according to the number of unlabeled compounds in each cluster. The desire is to sample more from highly-dense,

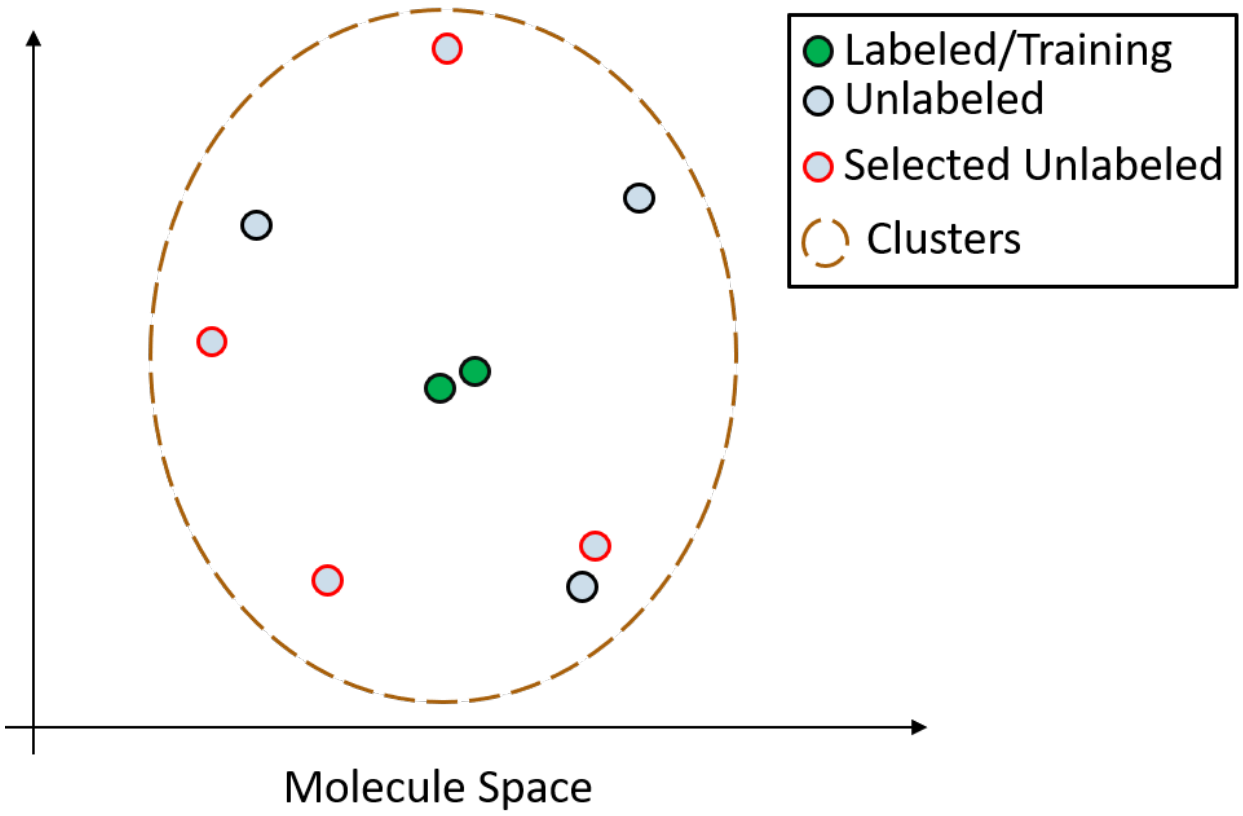


Figure 5.2: Contrived example illustrating an abstract 2D chemical space with selected cluster compounds that are not near each other. The idea is that, within a cluster, we want to select a diverse set of unlabelled compounds in order to explore as much of the cluster space as possible. Diversity here is measured via distance of compounds from each other.

low-coverage clusters.

$$\text{Exploitation cluster budget: } B_j = \left\lceil \frac{|C_j| B_{exploit}}{\sum_{C_i \in E_{exploitation}} |C_i|} \right\rceil \qquad \text{for all } C_j \in E_{exploitation}$$

$$\text{Exploration cluster budget: } B_j = \left\lceil \frac{|C_j| B_{explore}}{\sum_{C_i \in E_{exploration}} |C_i|} \right\rceil \qquad \text{for all } C_j \in E_{exploration}$$

Suppose that an exploitation (or exploration) cluster C_j is selected with cluster budget B_j. Now we want to select B_j unlabeled compounds from C_j. There are two ways to do this: random-selection or diversity-selection. The random-selection as the name implies randomly samples B_j unlabeled compounds. Diversity-selection samples a set of compounds from C_j that are diverse as

measured by their Tanimoto distances. The way this is done is as follows:

1. Denote the set of unlabeled compounds in C_j as U_{C_j}.

2. Compute the pairwise Tanimoto distance matrix for the unlabeled compounds in C_j. If C_j is an exploitation cluster, then we only consider unlabeled compounds for which the hit-probability exceeds $\tau_{activity}$.

3. Using the pairwise distances matrix, select the compound with the largest mean distance: x_1^*. Add this compound to the set of selected compounds: $S_{C_j} := \{x_1^*\}$. Update the cluster budget $B_j := B_j - 1$. Update the unlabeled cluster set: $U_{C_j} := U_{C_j} \setminus \{x_1^*\}$.

4. Now consider the compounds in U_{C_j} whose distances to ALL the compounds in S_{C_j} exceeds a cutoff τ_Δ. Denote this set of compounds as Z.

$$Z = \{x : x \in U_{C_j} \text{ and } \forall y \in S_{C_j}.\Delta(x, y) \geq \tau_\Delta\}$$

   This is to ensure that we don't sample compounds that are very close to each other; i.e. we want to cover more chemical space to gain more information (see Figure 5.2).

   Select a compound $x_i^* \in Z$ with the largest mean distance to the compounds in S_{C_j}. Update sets and budget as follows:

$$S_{C_j} := S_{C_j} \cup \{x_i^*\}$$
$$U_{C_j} := U_{C_j} \setminus \{x_i^*\}$$
$$B_j := B_j - 1$$

5. Repeat step 4 until $B_j = 0$ or $|Z| = 0$ (i.e. no qualifying compounds exceeding the cutoff distance). Return the set of selected compounds: S_{C_j}.

5.7 Design Decisions

There is debate on whether or not we should allow the same cluster to be both exploitable and explorable within the same iteration. This can happen if a cluster has a large number of compounds, many of which are unlabeled, and has a few compounds with high hit probabilities. In CBWS, it treats such a cluster as exploitable, and not explorable. We argue that treating a cluster as strictly exploitable or explorable, exclusively, allows more coverage of the chemical space by providing more budget for another cluster. Suppose there is a cluster with a healthy number of high hit probability compounds, but also a healthy number of uncertain compounds. In the current iteration, the cluster is treated as exploitable, and the compounds with high hit probabilities are sampled. One possible outcome is that as more compounds with high hit probabilities are sampled from this exploitation cluster, the lower its average cluster activity will become in subsequent iterations. As the cluster's average activity deteriorates and the cluster's average uncertainty rises, then it will no longer be qualified for exploitation. Thus, if it is indeed a suitable exploration cluster, then it will be selected for exploration at later iterations. Another possible outcome is that with each iteration, the cluster's average uncertainty decreases due the model becoming more confident as it trains on more of the cluster's data. In this case, due to the low average uncertainty, the cluster will likely not qualify as an exploration cluster.

The **reason for incorporating dissimilarity** is to ensure that we cover more of the chemical space. Figure 5.1 illustrates a 2D abstract representation of the chemical space where each point represents a compound. The contrived example showcases exploitation and exploration clusters, as well as clusters that are considered neither. It is important that the strategy explores adequately in order to update its beliefs about the coverage and hit probabilities of compounds.

5.8 Activity and Uncertainty

To measure activity and uncertainty for the unlabeled compounds U_i in iteration i, we use a supervised learning model trained on the training dataset T_i. Denote the model that trains on T_i and

predicts hit probabilities as L, then activity and uncertainty for unlabeled compounds are denoted as $L(U_i, Activity; T_i)$ and $L(U_i, Uncertainty; T_i)$, respectively. **Activity** of a compound is defined as the model's predicted hit probabilities given the compound features.

Uncertainty of a compound is defined as in the active learning domain in machine learning. Settles [110] provides a concise literature survey on active learning covering the scenarios and uncertainty calculation methods. In this chapter, the active learning scenario resembles the **pool-based sampling** scenario: a pool of unlabeled instances to select from. While the goal of active learning is, in general, to improve the model's **expected performance** in as few queried instances as possible, the goal of this project is to maximize the number of hits and diversity over the pool in as few iterations as possible. The **difference** is that maximizing hits over the pool may not coincide with improving the expected performance, although certainly improving expected performance may help in the long run of an infinitely large pool. Nonetheless, we use active learning's measure of uncertainty methods in CBWS as an aid in exploration. These uncertain compounds correspond to regions of the chemical space for which the model is unsure, and so, selecting them for exploration can help in expanding the model's knowledge. As we discussed, we also allocate budget for exploitation because we also want to return as many hits as possible. The exploration is important in this regard, as the model may have built a **bad** hypothesis with the training dataset it currently has, and so exploration (or even exploitation) of compounds will help adjust its bad hypothesis to a better one. As a final note, standard active learning methods use a batch size of 1 instance per iteration, whereas in this project we use batch sizes of 96, 384, and 1536 [110]. These batch sizes correspond to standard well plate sizes, so it is not cost efficient to screen 1 compound individually. We discuss three uncertainty measures that can be used by CBWS: least-confidence uncertainty sampling, query-by-committee (QBC), and density-weighted uncertainty.

Uncertainty sampling sets the uncertainty of a compound to be relative to its distance from the middle point of the hit probability, i.e. highest uncertainty when hit probability is 0.5. This is also known as uncertainty sampling by least-confidence. Given a compound x, training set T_i, and learning model L, the compound's uncertainty is given in terms of its hit probability

$L(x, Activity; T_i)$ by the following formula (also see Figure 5.3):

$$Uncertainty := 1 - |2L(x, Activity; T_i) - 1| \quad \text{Least-confidence formula} \tag{5.6}$$

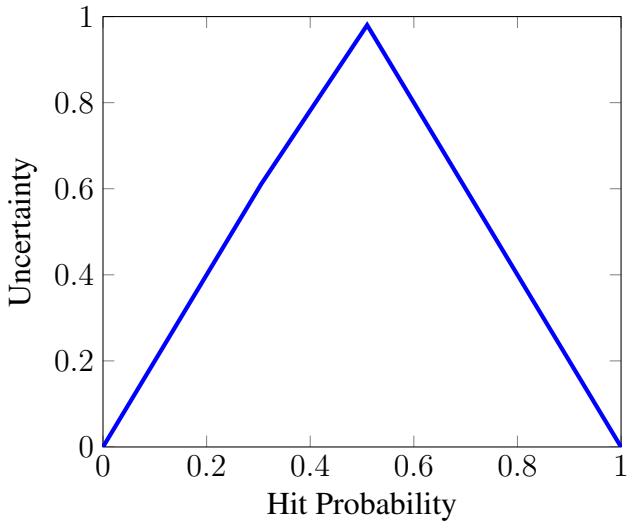


Figure 5.3: Uncertainty sampling by least-confidence function for the binary classification setting. The uncertainty is highest at the 0.5 hit probability and decreases as we move away from 0.5.

Query-by-committee (QBC) makes use of a committee of learners (e.g. an ensemble of decision trees) and sets the uncertainty of a compound to be a measure of disagreement among the learners. Because we use models that output hit probabilities, the disagreement measure among committee members is the average Kullback-Liebler divergence between the average committee distribution (consensus distribution) and individual learner distributions. This is given by the following formula:

$$Uncertainty = \frac{1}{N_{committee}} \sum_{i=1}^{N_{committee}} KL(P_i \| P_{committee}) \quad \text{QBC formula} \tag{5.7}$$

$$KL(P_i \| P_{committee}) = \sum_{c=0}^{1} P_i(y = c|x) \log \frac{P_i(y = c|x)}{P_{committee}(y = c|x)} \tag{5.8}$$

$$P_{committee}(y = c|x) = \frac{1}{N_{committee}} \sum_{i=1}^{N_{committee}} P_i(y = c|x) \tag{5.9}$$

Finally, one more appropriate uncertainty measure is an information density method. The

uncertainty of a compound in this case is a combination of an uncertainty measure and similarity measure to the underlying unlabeled pool U. This is done to help fight against querying instances that are outliers from the pool distribution. The formula is given by:

$$Uncertainty = \phi(x) \times \left(\frac{1}{|U|} \sum_{x_u \in U} sim(x, x_u) \right)^{\beta} \qquad \text{Density-weighted formula} \qquad (5.10)$$

Where $\phi(\cdot)$ is some uncertainty measure function like least-confidence or QBC, and β gives more or less weight to the similarity measure.

5.9 Summary of CBWS in Practice

After introducing the main components of the CBWS algorithm, we now summarize the steps of the algorithm:

1. Given an initial dataset T_0, unlabeled pool U_0, batch size B, maximum number of iterations τ_{max} , and supervised learning model L. It is assumed that the compounds in T_0 and U_0 are clustered using a clustering algorithm into clusters $\mathcal{C}$. Set $i := 0$.

2. Train L on T_i, then for each cluster in $\mathcal{C}$, compute cluster properties: $Density_i$, $Coverage_i$, $Activity_i$, and $Uncertainty_i$. For $Uncertainty_i$ we use one of the methods: least-confidence, QBC, or density-weighted.

3. Compute cluster exploitation and exploration weights. Identify qualifying exploitation and exploration clusters according to threshold: $\tau_{exploit}$ and $\tau_{explore}$, respectively. Compute rough estimates of the budget split among exploitation and exploration.

4. Go through the qualifying exploitation clusters in a greedy, most-exploitable-and-diverse clusters first, and sample compounds according to the **Cluster Sampling Function**. When finished adjust the remaining budget for the exploration clusters. Similarly select qualifying exploration clusters.

5. Return the selected compounds Z for in vitro screening. After screening, gather the data Z and update sets as follows:

$$T_{i+1} := T_i \cup Z$$
$$U_{i+1} := U_i \setminus Z$$
$$i := i + 1$$

6. Repeat steps 2-5 until $i = \tau_{max}$. Finally, if desired, evaluate performance on T_{final}.

5.10 Benchmarks

Random and Dissimilar Strategies

We introduce benchmark strategies that are based on random sampling and compound dissimilarity. These strategies were defined intuitively as benchmarks that other informative strategies should aim to beat. Note that these strategies make no use of a training set in their selection process. We list the strategies and describe the selection used in each iteration:

- **ClusterBasedRandom:** Randomly samples a cluster, then randomly samples compounds from that cluster in proportion to the size of the cluster; i.e. clusters with more compounds have a bigger sampling budget. The strategy continues randomly sampling clusters until it exhausts the total budget.

- **InstanceBasedRandom:** This strategy randomly samples compounds until the total budget is exhausted.

- **ClusterBasedDissimilar:** This strategy selects the first cluster randomly, then selects remaining clusters so that they are most dissimilar to clusters already selected until the total budget is exhausted. When a cluster is selected, it is given a cluster budget that is proportional to its size. Compound sampling within a selected cluster also proceeds via dissimilarity.

- **InstanceBasedDissimilar:** This strategy selects the first compound randomly, then selects remaining compounds so that they are most dissimilar to compounds already selected until the total budget is exhausted.

Multi-Armed Bandit (MAB) Strategies

These benchmark strategies are inspired by multi-armed bandit methods that utilize upper confidence bound (UCB) solutions [135, 136]. As a summary, in the MAB problem there are k arms that have k continuous distributions whose values correspond to rewards. The distributions are assumed to be parametric (e.g. normal distribution), but the parameters are unknown (e.g. the mean is unknown). The interaction with an arm is a random sampling of a reward from its underlying distribution. The goal is to select arms at each step/iteration so as to maximize the cumulative reward after n iterations. If the mean reward of each arm was known, then one would simply sample from the arm with the maximum mean reward. UCB style strategies select arms with an optimistic estimate of its mean reward; e.g. the upper-tail of a 95% confidence interval of the estimated mean reward based on current samples. Variants of MAB algorithms that make use of a *context* of the setting in regards to which arm to select are called contextual MABs. For example, in the case of iterative batched screening, the context can be the model predictions and dissimilarity of the unlabelled pool compounds. This compound information can be used to decide which cluster to select for sampling in a contextual MAB.

The iterative batched screening setting shares similarities with the MAB setting by treating compound clusters as arms, whereby we want to sample from clusters that have higher expected hits. However, in our setting the arms/clusters diminish in their hits with every hit discovered. This change of reward distribution of an arm violates the stationary environment assumption of standard MAB settings. Furthermore, most MAB solutions assume that only one arm is sampled each iteration, and although there are variants of batched MAB solutions, they do not deal with the non-stationary environment case [137]. Nonetheless, we craft a MAB-UCB style solution fitted to the iterative batched screening setting which we call **MABSelector**. The solution shares similarities

with what Madhawa et al. [138] proposed for maximizing discovery of a network graph.

In a typical UCB style solution, the estimated mean reward of an arm is a linear combination of an exploitation and exploration term, with a parameter α to control the trade-off. In our setting, we treat each cluster as an arm and make use of a machine learning model to produce compound predictions as context. At each iteration, we train the machine learning model on the training set and produce predictions on the unlabelled pool compounds. Subsequently, for each cluster, we compute its exploitation term as the mean of the cluster's unlabelled compound hit probabilities ($\mu_{exploit}$), and its exploration term as the mean of the cluster's unlabelled compound uncertainty ($\mu_{explore}$). Thus, a cluster's mean reward is estimated by the following formula:

$$\hat{r}^i = \mu^i_{exploit} + \alpha\mu^i_{explore} \qquad \text{for each cluster } i \tag{5.11}$$

At each iteration, the MABSelector strategy proceeds as follows:

1. Train the machine learning model (i.e. random forest) on the current iteration's training set. Compute hit predictions for all unlabelled pool compounds. Let C be the set of clusters in the unlabelled pool.

2. Compute the estimated reward for each cluster in $\mathcal{C}$ as shown above in Equation 5.11. Select the cluster A in $\mathcal{C}$ with the largest estimated reward.

3. Within the selected cluster A, compute estimated rewards for each unlabelled compound as shown above in equation 5.11; i.e. linear combination of prediction and uncertainty scores. Select the unlabelled compound x in A with the largest estimated reward.

4. Remove x from the cluster A. Repeat steps 2 and 3 until the total budget is exhausted.

As seen in equation 5.11, there is only one parameter α to be tuned. Since the exploration and exploitation terms are in the range from 0 to 1, we define four MABSelector settings that reflect the trade-off spectrum from no-exploration to heavy-exploration. Table 5.1 show the MABSelec-

tor names and their corresponding α value used in the experiments. The name of MABSelector_exploitive is due to the full reliance on the exploitation term in equation 5.11.

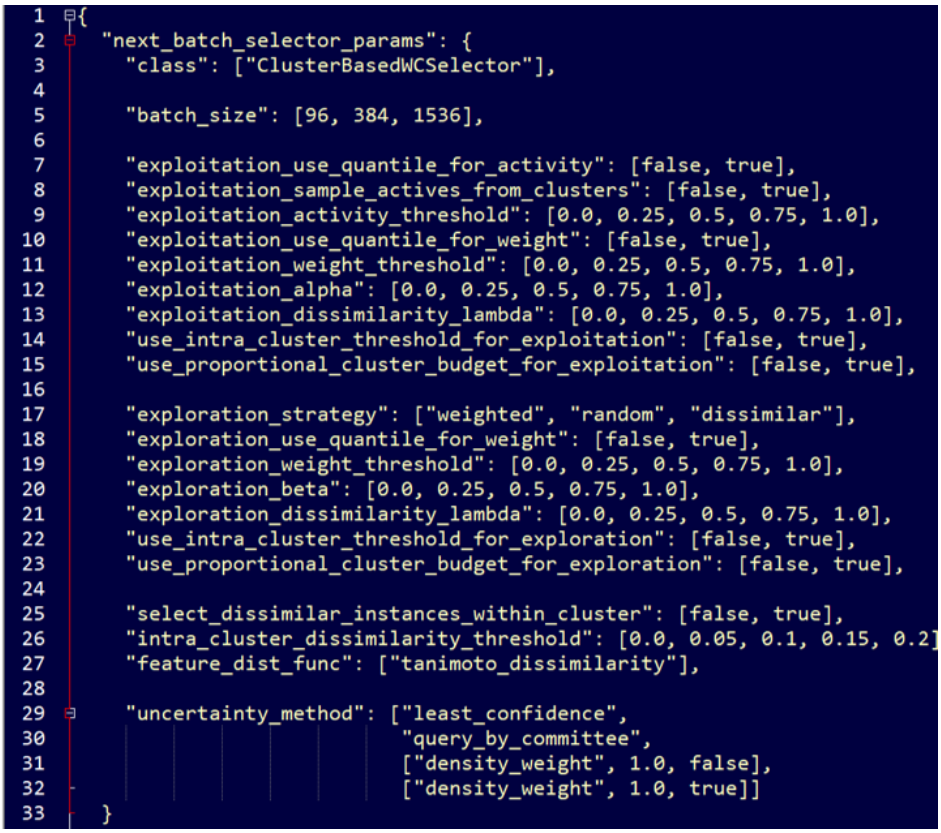
Table 5.1: MABSelector names and α configurations used in experiments.

Name	α
MABSelector_exploitive	0.0
MABSelector_1	0.2
MABSelector_2	0.5
MABSelector_3	0.8

5.11 Experiments Overview

The goal of these experiments is to explore the performance of CBWS and benchmarks in a variety of screening settings. The experiments serve to prune different parameter configurations of CBWS based on performance metrics. Figure 5.4 showcases the hyperparameters in the CBWS algorithm. Continuous parameters were discretized as shown. Consequently, with this discretization, there are 600 million possible parameter settings. For searching this space, random-search or prior-distribution search can be done. By prior-distribution we mean placing a distribution over each of the parameters, and sampling the parameter values from this distribution. Random-search is a uniform prior over each of the parameters. Ultimately, due the large number of possible settings, we cannot hope to do the search justice. Thus, we compromise according to our computational resources by opting towards a 4:1 ratio of prior-distribution and random-searching. In particular, we sample 800 settings from the prior-distribution and 200 settings from the random-search. The prior-distribution was set manually based on discussions with chemist team members. We refer to these sampled CBWS strategies as **Sampled CBWS**.

In addition, we manually select intuitive settings for the CBWS parameters. We refer to these manually defined CBWS strategies as **Custom CBWS**. For comparison purposes, we add popular benchmark strategies like random-search and Multi-armed bandit (MAB) strategies (see section 5.10). Table 5.2 shows an overview of the strategy groups and counts for each experiment. There are five experiments in total, and they are conducted in order, pruning and adding strategies along



```
{
  "next_batch_selector_params": {
    "class": ["ClusterBasedWCSelector"],

    "batch_size": [96, 384, 1536],

    "exploitation_use_quantile_for_activity": [false, true],
    "exploitation_sample_actives_from_clusters": [false, true],
    "exploitation_activity_threshold": [0.0, 0.25, 0.5, 0.75, 1.0],
    "exploitation_use_quantile_for_weight": [false, true],
    "exploitation_weight_threshold": [0.0, 0.25, 0.5, 0.75, 1.0],
    "exploitation_alpha": [0.0, 0.25, 0.5, 0.75, 1.0],
    "exploitation_dissimilarity_lambda": [0.0, 0.25, 0.5, 0.75, 1.0],
    "use_intra_cluster_threshold_for_exploitation": [false, true],
    "use_proportional_cluster_budget_for_exploitation": [false, true],

    "exploration_strategy": ["weighted", "random", "dissimilar"],
    "exploration_use_quantile_for_weight": [false, true],
    "exploration_weight_threshold": [0.0, 0.25, 0.5, 0.75, 1.0],
    "exploration_beta": [0.0, 0.25, 0.5, 0.75, 1.0],
    "exploration_dissimilarity_lambda": [0.0, 0.25, 0.5, 0.75, 1.0],
    "use_intra_cluster_threshold_for_exploration": [false, true],
    "use_proportional_cluster_budget_for_exploration": [false, true],

    "select_dissimilar_instances_within_cluster": [false, true],
    "intra_cluster_dissimilarity_threshold": [0.0, 0.05, 0.1, 0.15, 0.2]
    "feature_dist_func": ["tanimoto_dissimilarity"],

    "uncertainty_method": ["least_confidence",
                           "query_by_committee",
                           ["density_weight", 1.0, false],
                           ["density_weight", 1.0, true]]
  }
```

Figure 5.4: Hyperparameters for the CBWS algorithm.

the way. Using hit-based performance metrics, the goal is to select one final strategy to run on the prospective target in experiment 4 (see section 5.18).

Table 5.2: Summary of strategy groups and counts used in this project's experiments. Experiments are run sequentially, pruning (and sometimes adding) strategies. The final experiment uses a single promoted strategy.

Strategy Group	Experiment 0	Experiment 1	Experiment 2	Experiment 3	Experiment 4
Sampled CBWS	800 prior-distribution 200 uniform-random	70	12	3	1
Custom CBWS	-	4	3	1	-
Benchmarks	-	4	7	4	-
Total	1000	78	22	8	1

For many of the experiments we provide the strategies with an initial plate of 96 labelled compounds to serve as the initial training set. We use the term initial training plate and initial seeding plate interchangeably. The manner in which these initial seeding plates are sampled from

the datasets are described within each experiment's section. Experiment 0 is an initial search whereby we run 800 prior-distribution and 200 random-search settings for each batch size: 96, 384, and 1536. To make the computational demands manageable, we focus on a single target dataset (PriA-SSB), one initial seeding plate, and 10 iterations. We promote strategies from top, middle, and worst performers to experiment 1 to analyze if they maintain these delineations when settings are changed. In experiment 1, a larger dataset is used with 10 different initial seeding plates to judge the robustness of strategies. Custom and benchmark strategies are also introduced. Here we only promote top performers from each of the three groups. Experiment 2 assess the long run performance at 50 iterations under the same settings of experiment 1. At this point, we solely use batch size 96 due to computational and time demands. In experiment 3, we conduct runs on 107 diverse target datasets with 10 initial seeding plates and for 50 iterations. Experiment 3 serves as the decider for promoting a single strategy to the prospective dataset in experiment 4. This final strategy is used for experiment 4 on a prospective target with unknown labels/hits for 50 iterations and no initial seeding plate. Details for each experiment can be found in the corresponding experiment section.

5.12 Tools for Comparing Strategies

The primary metrics for comparing strategies are total hits and total unique hits. Total hits is the number of compound hits a strategy has acquired over a defined number of iterations. Total unique hits are used as a proxy for the novelty of hit compounds. In this chapter, it is the the number of *unique* clusters that contain at least one hit. Clusters are defined by the Taylor-Butina [80, 81] clustering algorithm.

In all experiments, each strategy performs a number of runs using a combination of different: batch sizes, initial starting plates, and target datasets. As a result, there are different ways to group/aggregate the results along these dimensions. Strategies can be compared using single-point estimates like mean and standard deviation of hit metrics by aggregating along the strategy

dimension. This may be appropriate for experiments 1 and 2 since we have a single target dataset, but for experiment 3, we have 107 diverse targets. These 107 target datasets can have significantly varying hit rates and learning tasks. Thus, it makes more sense to compare strategies per task in experiment 3.

Use cases and issues with statistical hypothesis testing for comparing algorithms in the context of machine learning have been discussed in many papers, most notably by Demšar et al. and García et al. [139, 140, 141, 142]. The main setting in these papers is the comparison of n machine learning models on k datasets. Furthermore, each model and dataset pairing has a single metric. In experiment 1 and 2, we have a single target dataset and multiple initial starting plates. Thus, the strategies and initial starting plates are analogous to the models and datasets, respectively. The tests suggested are the non-parametric Friedmann test followed by an appropriate post-hoc correction. However, in experiment 3, we have strategies, tasks, and initial seeding plates. Demšar et al. point out that they do not know of any test to incorporate a setting where every model and dataset have multiple readings without violating required assumptions.

In any case, we do not make use of significance hypothesis tests since the goal of our experiments are clear: promote a single final strategy for the prospective target in experiment 4. That is, regardless of significance, we are to promote a final strategy. In 2010, García et al. [141] introduced a non-parametric comparison among methods called contrast estimation based on medians (CEM). CEM does not use hypothesis testing, but rather computes the median of differences between each pair of algorithms across datasets. These median of differences are then averaged for each algorithm to compute a single metric per algorithm μ_i that reflects the average difference in performance against other algorithms. Finally, we compute algorithm-vs-algorithm differences $\mu_i - \mu_j$ for every pair of algorithms i and j. The results are stored in the CEM matrix. Each cell in the CEM matrix is read as row-algorithm vs column-algorithm. A positive value indicates that the row-algorithm outperforms the column-algorithm. Thus, a row-algorithm that exhibits positive performance in each of its cells is deemed as the best. It is important to note, as stated by García et al. [141], that the intent of CEM is to find performance difference based on the results. Thus, CEM does

not test for significance based on probability. Several recent papers have used CEM for comparing machine learning algorithms [143, 144, 145, 146]. In experiment 1, 2, and 3, we use CEM matrices by treating strategies and initial plates as the algorithms and datasets, respectively. In experiment 3, we construct CEM matrices per task. Details for experiment strategy promotions based on CEM matrices can be found in the corresponding experiment section.

5.13 Datasets Overview

In this section we describe the datasets used in this chapter's experiments. The PriA-SSB dataset consists of compound-target % inhibition interaction of compounds with a protein-protein target. The target description and inhibitor interaction is introduced in Voter et al. [57]. The processed/binarized version of the dataset was introduced in Liu et al. [34]. PriA-SSB has 94,857 compounds with 133 hits (see Table 5.3), and was used in experiments 0 and 1. This is the same dataset that was used in chapter 2 of this thesis.

The 128-PCBA dataset consists of 128 bioactivity compound-target datasets with a wide range of compound counts and hit percentages that can be downloaded from PubChem BioAssay [5]. This dataset first appeared in Ramsundar et al. massive multitask neural networks paper [4]. The 128-PCBA was also used in the informer set experiments in chapter 4. As aforementioned, these are 128 diverse targets consisting of proteins, protein-protein interactions, enzymes, GPCRs, etc. (see appendix in Ramsundar et al. [4]). The 128 assays were downloaded using the search criterion as detailed by Ramsundar et al. Each target dataset was processed to construct compound canonical SMILES using RDKit [1] and hits were set via the 'Active' label in the datasets ('Inactive' and 'Inconclusive' compounds were set as inactive). Compounds that produced errors in processing were dropped. We restrict our experiments to target datasets with more than 100,000 compounds; only 107 of the 128 targets qualify. Table 5.4 summarizes the size and hits of these 107 target datasets. Note that the sizes and hits of 128-PCBA defined here in Table 5.4 are slightly different than the 128-PCBA counts that appear in the informer set experiments in chapter 4. This is because

the 128-PCBA dataset in this chapter was constructed in 2016 using an earlier version of RDKit, whereas the dataset in chapter 4 was constructed in 2020 using a recent version of RDKit. The different versions of RDKit can result in different compound processing errors which may explain the difference in counts.

Recall from chapter 2 that the PriA-SSB dataset consists of compounds obtained from Life Chemicals, Inc. (LC). 94,044 of these LC compounds were screened in vitro on a target of interest: phosphoserine/threonine phosphatase (PstP), a protein phosphatase in the Mycobacterium tuberculosis cell membrane. This target is used as the prospective target in experiment 4 that is run with the final promoted strategy. All 94,044 compounds have already been screened in vitro by a collaborative lab by the time experiment 4 was run, but the details of the hits were kept unknown to the strategy. Only compound information was made available like SMILES and structure, which were used to generate features and to cluster the compounds. Details on how experiment 4 was conducted can be found in section 5.18. For completeness and organization, we show the summary of the PstP dataset in Table 5.3.

All datasets were clustered using the Taylor-Butina algorithm [80, 81] with a threshold of 0.4. Consequently, unique hits are defined as the total number of unique clusters with at least one hit. Finally, experiments 0 to 3 employ initial seeding plates that serve as the initial training set for the first iteration of every run. For a given target dataset, these initial seeding plates are randomly sampled 96-compound plates with exactly 1 active (unless otherwise stated). If there are more than one initial seeding plates like in experiment 3, then they are exclusive; i.e. different seeding plates do not share the same compounds.

Table 5.3: Summary of PriA-SSB and PstP datasets used in experiment 0 and experiment 4, respectively. Note that PstP hit labels were not known until after experiment 4 was performed.

Target ID	# Hits	# Unique Hits	Count	Hit %
PriA-SSB	133	99	94857	0.14
PstP	143	127	94044	0.15

Table 5.4: Summary of 128-PCBA [5, 4], a diverse set of 128 target datasets used in experiments 2 and 3 to promote strategies. We omit targets with less than 100,000 compounds, yielding 107 valid targets.

Target Assay ID	# Hits	# Unique Hits	Count	Hit %
aid588456	51	36	384401	0.01
aid504891	34	29	361253	0.01
aid463254	41	32	329208	0.01
aid602332	69	66	408382	0.02
aid492947	80	66	329377	0.02
aid504845	100	91	372364	0.03
aid624246	101	84	364607	0.03
aid504842	101	85	324667	0.03
aid2675	99	84	248888	0.04
aid2662	110	89	285346	0.04
aid602233	165	140	379215	0.04
aid485294	148	106	309795	0.05
aid1469	170	134	272696	0.06
aid1634	154	120	262142	0.06
aid720707	268	216	363520	0.07
aid624291	222	219	332023	0.07
aid504706	201	195	302745	0.07
aid652025	238	191	364400	0.07
aid743266	306	301	399029	0.08
aid720711	290	221	363530	0.08
aid602310	310	267	394124	0.08
aid485281	253	229	314600	0.08
aid602179	364	283	385215	0.09
aid2101	288	194	310187	0.09
aid1452	178	146	149537	0.12
aid1471	293	277	218543	0.13
aid624287	423	362	302649	0.14
aid652106	497	415	362825	0.14
aid720709	516	419	353361	0.15
aid485367	557	451	326151	0.17
aid720708	661	541	357399	0.18
aid485349	618	558	320080	0.19
aid485353	603	526	323054	0.19

Continued on next page

Table 5.4: Summary for 128-PCBA [5, 4], a diverse set of 128 target datasets used in experiments 2 and 3 to promote strategies. We omit targets with less than 100,000 compounds, yielding 107 valid targets.

Target ID	# Hits	# Unique Hits	Count	Hit %
aid2528	652	514	341585	0.19
aid602313	762	612	373030	0.20
aid624170	838	693	398586	0.21
aid651644	748	526	354725	0.21
aid504327	766	634	371753	0.21
aid720542	733	562	356932	0.21
aid743255	901	628	367811	0.24
aid1379	561	417	196927	0.28
aid485290	938	752	336790	0.28
aid1479	793	671	270316	0.29
aid2676	1081	669	358419	0.30
aid624171	1239	1006	395908	0.31
aid2517	1138	849	333254	0.34
aid1631	892	687	259922	0.34
aid1457	720	571	202822	0.35
aid588795	1307	911	377549	0.35
aid1721	1087	701	290738	0.37
aid720551	1265	1056	342920	0.37
aid2242	715	417	184089	0.39
aid2100	1157	892	293012	0.39
aid1468	1038	733	252185	0.41
aid2326	1065	969	260751	0.41
aid624288	1356	1161	324402	0.42
aid1454	513	418	115847	0.44
aid651768	1677	1192	357664	0.47
aid720580	1508	1102	305957	0.49
aid588579	1980	1383	386184	0.51
aid2549	1211	972	231654	0.52
aid485341	1729	1480	327428	0.53
aid881	590	454	104390	0.57
aid540317	2129	1632	370041	0.58
aid720579	1908	1469	282894	0.67
aid485360	1485	1126	218478	0.68

Continued on next page

Table 5.4: Summary for 128-PCBA [5, 4], a diverse set of 128 target datasets used in experiments 2 and 3 to promote strategies. We omit targets with less than 100,000 compounds, yielding 107 valid targets.

Target ID	# Hits	# Unique Hits	Count	Hit %
aid2451	2005	1530	273722	0.73
aid504847	3509	2470	380031	0.92
aid924	1144	805	119950	0.95
aid720553	3259	2493	339283	0.96
aid588453	3904	2558	369760	1.06
aid624202	3968	2904	366506	1.08
aid651635	3784	2431	346938	1.09
aid588590	3931	2845	356872	1.10
aid1461	2305	1833	208320	1.11
aid1688	2375	1940	204281	1.16
aid588591	4700	2996	372675	1.26
aid652105	4072	2757	322432	1.26
aid504466	4169	2729	310916	1.34
aid588855	4897	3378	352448	1.39
aid485314	4493	3062	317082	1.42
aid902	1865	1290	118929	1.57
aid686970	5948	4126	337003	1.76
aid2147	3473	2660	192236	1.81
aid652104	7126	4962	375678	1.90
aid624417	6389	4856	325673	1.96
aid651965	6346	4288	324379	1.96
aid624297	6213	4517	308156	2.02
aid540276	4393	3403	197140	2.23
aid485313	7569	3982	311759	2.43
aid504444	7388	4985	290376	2.54
aid1460	5650	3485	222659	2.54
aid720504	10170	7325	350522	2.90
aid485297	9128	4701	310418	2.94
aid1458	5778	3549	194622	2.97
aid485364	10698	6190	342165	3.13
aid504467	7648	5061	243252	3.14
aid624296	9840	6910	292261	3.37
aid2546	10556	5913	278436	3.79

Continued on next page

Table 5.4: Summary for 128-PCBA [5, 4], a diverse set of 128 target datasets used in experiments 2 and 3 to promote strategies. We omit targets with less than 100,000 compounds, yielding 107 valid targets.

Target ID	# Hits	# Unique Hits	Count	Hit %
aid504339	16858	9481	355613	4.74
aid504333	15673	8587	325784	4.81
aid2551	16671	8147	270318	6.17
aid588342	25036	10947	326779	7.66
aid1030	15932	10158	161295	9.88
aid504332	30264	16664	294012	10.29
aid686979	48532	21675	305803	15.87
aid686978	62375	25705	298995	20.86

5.14 Experiment 0: Initial CBWS Hyperparameter Sweep

Overview and Goal

This initial hyperparameter sweep of CBWS consisted of 1000 hyperparameter configurations. 800 were sampled from a prior distribution and 200 were sampled from a uniform random distribution. The prior distribution was manually set based on intuitive discussions with chemist group members. In this experiment we ran each of the 1000 hyperparameter settings on the PriA-SSB dataset for 10 iterations and for each batch size {96, 384, 1536}; that is a total of $1000 \times 3 = 3000$ jobs. From the dataset, we randomly selected a 96-compound plate with exactly 1 active to be the initial starting plate for all runs; i.e. the initial training set. The **primary goal** was to select top hyperparameter configurations that maximize cumulative total hits at the end of the 10 iterations. As a **secondary goal**, we wanted to analyze the effect of batch size; i.e. are the top hyperparameter configurations the same across batch sizes?

After scheduling these jobs and running until a defined time deadline was reached, only 1828 out of the 3000 jobs finished completely (all 10 iterations). The default deadline for each job was 72 hours, but some strategies and larger batch sizes (384 and 1536) were extended to roughly 336 hours (2 weeks). Details of hyperparameter counts by batch size are shown in Table 5.5. These

1828 jobs included a total of 775 hyperparameters: 618 (out of 800) from the prior-distribution and 157 (out of 200) from the random distribution.

Table 5.5: Experiment 0 summary of successfully completed hyperparameters; success denotes that the full 10 iterations were run to completion.

Batch size	Total Hyperparameters	Successfully Finished Hyperparameters
96	1000	764
384	1000	763
1536	1000	301
Intersection	-	298
Union	-	775

Results and Discussion

Each job is a combination of hyperparameter configuration and batch size. Thus, if we group results by batch size, we can look at the per-iteration hit aggregates across hyperparameter configurations. Due to the difference in successful hyperparameters by batch size (see Table 5.5), we only consider the 298 overlapping hyperparameters. Figure 5.5 showcases the mean ratio of total hits to batch size per iteration. The normalization by ratio was done to allow comparison among the batch sizes; i.e. as a proxy for efficiency. Since we are interested in cumulative hits, Figure 5.6 showcases a similar plot as the ratio of cumulative hits to batch size per iteration. Intuitively, one would expect smaller batch sizes to allow for better cumulative hit retrievals, since that would allow a strategy to take less risks and adjust its decision. At the same time, a larger batch size gives the strategy more information about the learning task. Both figures showcase that the middle batch size of 384 is the best in terms of cumulative hit retrievals; however batch size of 96 is also comparable and outperforms in the early iterations.

For the primary goal, we look at the cumulative total hits for each job. As seen in the results earlier, due to the effect of batch size on performance, we look at each batch size results individually. Tables 5.6, 5.7, and 5.8 show the top 15 hyperparameter performers from each batch size. There is very little overlap across these batch sizes; only CBWS_678, CBWS_28, CBWS_219, and

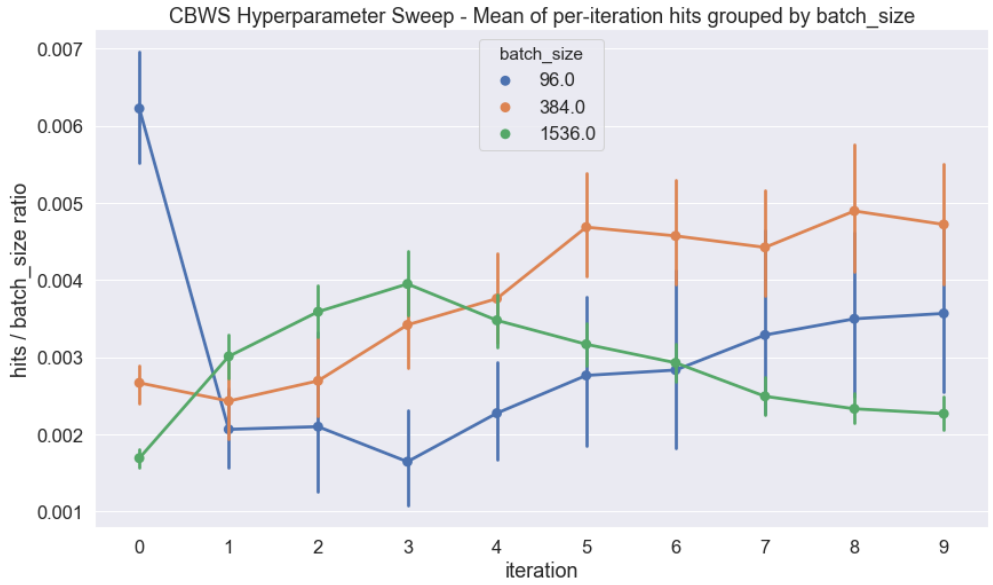


Figure 5.5: Plot of the mean ratio of hits to batch size for experiment 0 hyperparameter results along iterations.

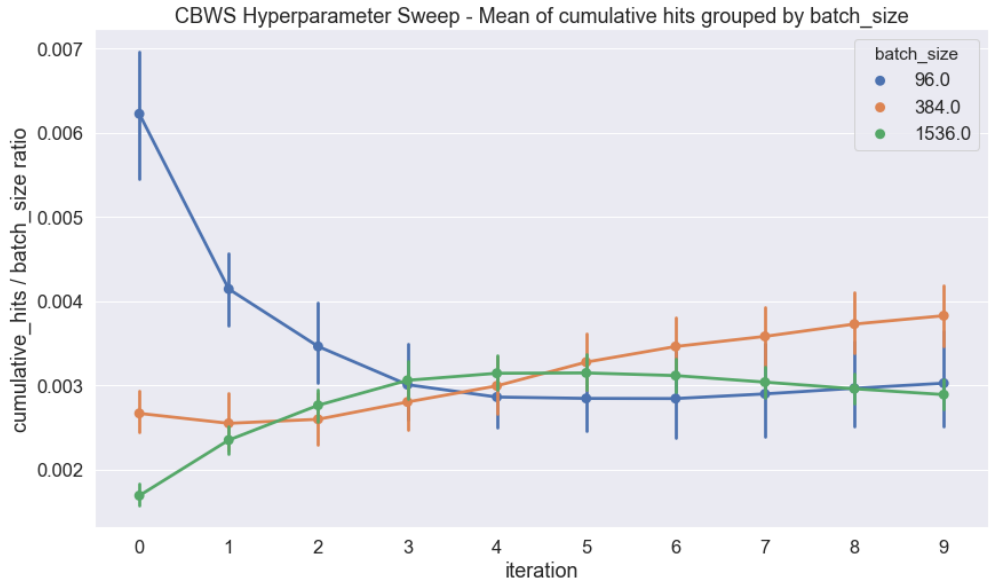


Figure 5.6: Plot of the mean ratio of cumulative hits to batch size for experiment 0 hyperparameter results along iterations.

CBWS_411 overlap. This could be mainly due to many jobs not finishing for higher batch sizes due to time limits on the compute nodes. Nonetheless, we promote all these configurations to the next experiment. Furthermore, we can see a healthy representation of full exploration strategies which was not expected. An exploration heavy strategy makes use of the machine learning model's uncer-

tainty and dissimilarity of already selected compounds, so it should not be prioritizing compounds that the model predicts to be highly active.

Finally, based on the cumulative total hits over the 10 iterations, we promoted the top 15, middle 5, and bottom 5 hyperparameters from *each* batch size to the next experimental stage. We select top, middle, and bottom in order to confirm their performance ordering is consistent in the next experiment. This was a total of $15 + 5 + 5 = 70$ hyperparameters due to overlap in hyperparameter performance across batch size.

Table 5.6: Experiment 0 results of top 15 hyperparameters for batch size 96.

Strategy ID	Exploitation Compounds	Exploration Compounds	Exploitation Hits	Exploration Hits	Total Hits
CBWS_456	0	960	0	17	17
CBWS_592	226	734	10	8	18
CBWS_38	957	3	18	0	18
CBWS_666	393	567	17	1	18
CBWS_427	437	523	17	1	18
CBWS_467	0	960	0	19	19
CBWS_678	0	960	0	19	19
CBWS_28	960	0	19	0	19
CBWS_27	0	960	0	20	20
CBWS_46	38	922	15	7	22
CBWS_54	17	943	9	19	28
CBWS_341	0	960	0	31	31
CBWS_787	480	480	30	2	32
CBWS_368	12	948	7	26	33
CBWS_248	0	960	0	40	40

Table 5.7: Experiment 0 results of top 15 hyperparameters for batch size 384.

Strategy ID	Exploitation Compounds	Exploration Compounds	Exploitation Hits	Exploration Hits	Total Hits
CBWS_288	0	3840	0	44	44
CBWS_678	0	3840	0	44	44
CBWS_670	0	3840	0	45	45
CBWS_411	0	3840	0	45	45
CBWS_55	0	3840	0	46	46
CBWS_219	0	3840	0	47	47
CBWS_389	0	3840	0	49	49
CBWS_84	800	3040	4	46	50
CBWS_15	0	3840	0	50	50
CBWS_228	1828	2012	47	3	50
CBWS_627	1940	1900	41	10	51
CBWS_676	1782	2058	50	1	51
CBWS_428	0	3840	0	63	63
CBWS_207	0	3840	0	64	64
CBWS_75	0	3840	0	69	69

Table 5.8: Experiment 0 results of top 15 hyperparameters for batch size 1536.

Strategy ID	Exploitation Compounds	Exploration Compounds	Exploitation Hits	Exploration Hits	Total Hits
CBWS_28	15360	0	87	0	87
CBWS_638	0	15360	0	87	87
CBWS_411	0	15360	0	87	87
CBWS_344	8044	7316	78	11	89
CBWS_124	6402	8958	80	10	90
CBWS_201	0	15360	0	91	91
CBWS_491	0	15360	0	91	91
CBWS_533	0	15360	0	92	92
CBWS_262	0	15360	0	93	93
CBWS_86	0	15360	0	93	93
CBWS_572	0	15360	0	94	94
CBWS_208	0	15360	0	96	96
CBWS_490	0	15360	0	96	96
CBWS_219	0	15360	0	97	97
CBWS_558	0	15360	0	104	104

5.15 Experiment 1: Larger Dataset, 10 Initial Starting Plates, 3 batch sizes, and 10 iterations

Overview and Goal

From the previous experiment, 70 sampled CBWS hyperparameters were promoted, and we want to prune them further. In this experiment, we used PubChem target aid624173 with 400,122 compounds and 487 actives [5]. From the dataset, 10 different 96-compound plates with exactly 1 active per plate were generated as initial seeding plates. Each hyperparameter was run for each batch size of {96, 384, 1536} and 10 different seeding plates for a total of 10 iterations. The 10 seeding plates were exclusive 96-compound plates randomly selected from the dataset. In total, these are $70 \times 3 \times 10 = 2100$ jobs.

Furthermore, a number of custom and benchmark strategies were introduced in this stage. Three custom defined hyperparameter settings for CBWS based on intuition, one custom instance-based CBWS implementation where each compound is treated as its own cluster, and an adapted multi-armed bandit upper-confidence-bound (MAB-UCB) style algorithm with 4 settings. We call the MAB-UCB strategies MABSelector as introduced in section 5.10. That is a total of $4 + 4 = 8$ custom and benchmark strategies; a total of $8 \times 3 \times 10 = 240$ jobs.

This is a total of 78 strategies for a total of 2340 jobs. Recall that each job is a combination of strategy, batch size, and starting plate. Thus, for each strategy and batch size, we have 10 starting plate runs. We denote a strategy and batch size combination as successfully run if more than 5 of the 10 plate runs successfully finish all 10 iterations. The **goal** in this experiment is to compare strategy robustness on a larger dataset with different starting initial plates.

Again, similar to experiment 0, here the default deadline for each job was 72 hours, but some strategies and larger batch sizes (384 and 1536) were extended to roughly 336 hours (2 weeks). Only 802 of the 2100 sampled CBWS jobs finished completely. These 802 jobs included 62 of the 70 sampled CBWS hyperparameters. The 4 MABSelector strategies and 2 of the 4 custom CBWS strategies finished successfully for batch sizes 96 and 384. Analysis for batch size 1536

was removed because only two strategies of the 78 total strategies finished successfully. Table 5.9 summarizes the successful strategy and batch size combination counts.

Table 5.9: Experiment 1 summary of successfully completed strategy and batch size combinations. Success denotes that more than 5 of the 10 initial starting plate runs finished 10 iterations.

Group	Total	Success Batch Size 96	Success Batch Size 384	Success Batch Size 1536
Sampled CBWS	70	56	25	2
Custom CBWS	4	4	4	0
Benchmark	4	2	2	0
Total	78	62	31	2

Results and Discussion

First, we look at the boxplots for each batch size in Figures 5.7 and 5.8. The x-axis strategy labels are color coded to indicate which performance groupings they belong to from experiment 0 (top, middle, and bottom). Recall these were promoted in order to confirm they adhere to the same ordering; i.e. the worst strategies in experiment 0 are generally the worst in experiment 1. Overall, the strategies' total hits performances adhere to their experiment 0 groupings. Tables 5.10 and 5.11 showcase the metric means and standard deviations for all strategies, along with exploitation and exploration hits. Of interest is that many of the CBWS models that excel are exploration heavy.

Next, we used contrast estimation based on medians (CEM) total wins to promote strategies. Recall from section 5.12 that a strategy wins against another strategy if its CEM difference is positive [141]. For each batch size of 96 and 384, we compute the CEM matrix and tally the total wins. Figures 5.9 and 5.10 showcase the heatmaps for each batch size and metric along with the total wins for each strategy. This gives four criteria for promotion: top 15 strategies based on total hits CEM wins for batch size 96, top 15 strategies based on total unique hits CEM wins for batch size 96, top 15 strategies based on total hits CEM wins for batch size 384, and top 15 strategies based on total unique hits CEM wins for batch size 384. There are many overlapping strategies in these lists, and its easy to define a top set of strategies just by manual inspection. Note that although some strategies did not complete their 10 runs (we set the threshold to 5 runs), and thus we have

unequal sample sizes, we can still technically compute contrast estimation based on medians by ignoring incomplete paired samples (see algorithm computation in Garcia et al. [141]).

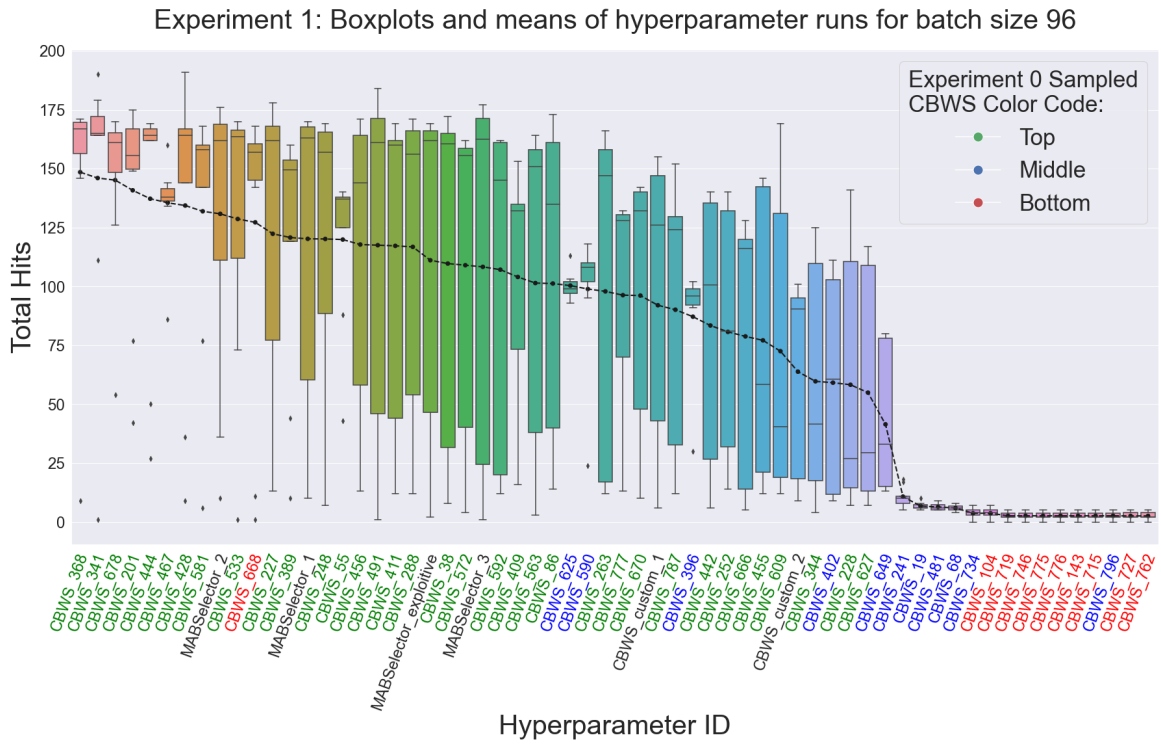
To that end, we promote strategies from each of the strategy groups independently (Sampled CBWS, Custom CBWS, and Benchmarks). The promoted list is as follows:

- 12 sampled CBWS strategies with the following IDs: 201, 341, 368, 396, 411, 467, 55, 581, 590, 609, 625, and 678.

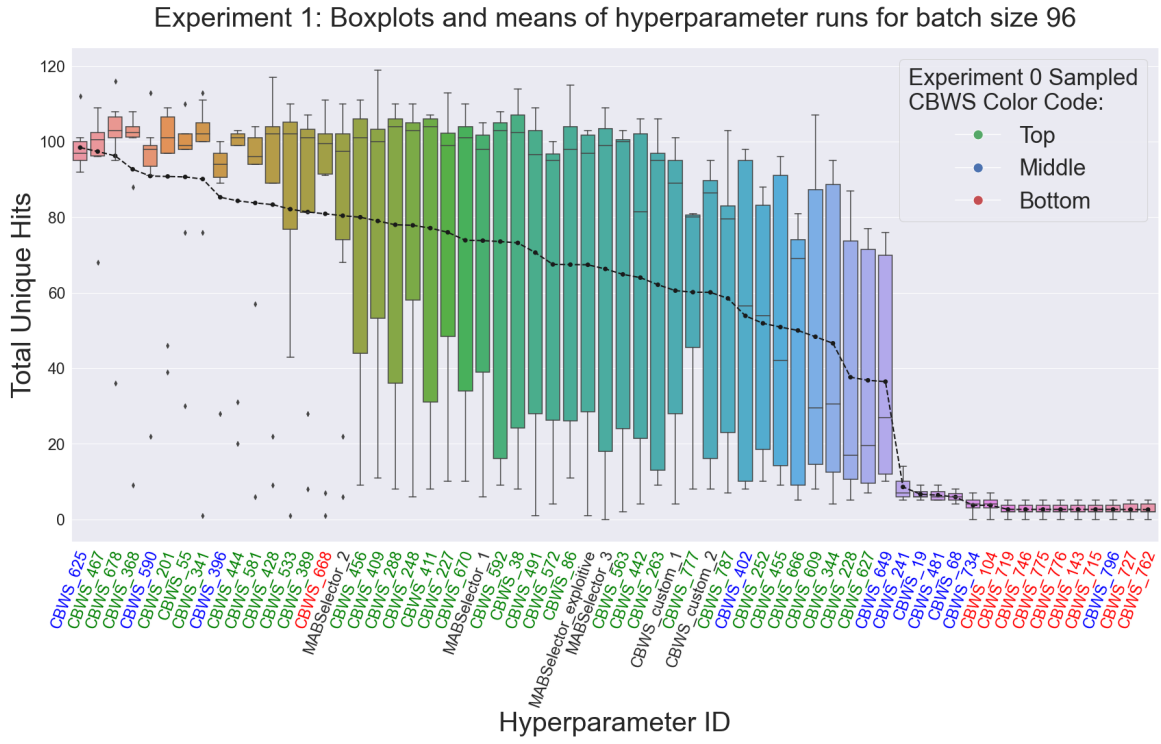
  These are mostly exploration heavy strategies, but we also add some high performing exploitation strategies like CBWS_581.

- 3 benchmarks: MABSelector_exploitive, MABSelector_2, and MABSelector_3. MABSelector_2 is the best MABSelector setting on batch size 96 based on CEM wins. MABSelector_exploitive and MABSelector_3 are on the fringe settings of exploiting heavily and exploring heavily, and they are both the top MABSelector settings on batch size 384 based on CEM wins.

- 3 custom CBWS strategies: CBWS_custom_1, CBWS_custom_2, InstanceBWS_custom_0. Note that InstanceBWS_custom_0 did not finish successfully in experiment 1 but we still include it for the next experiment.

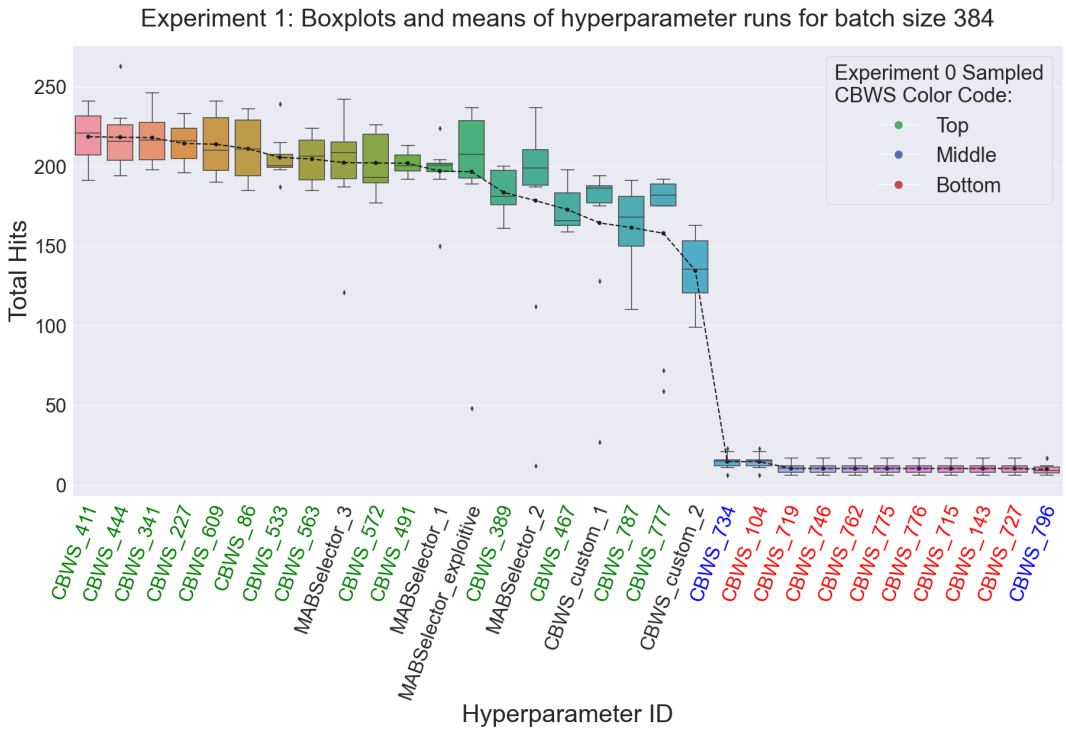


(a) Total hits

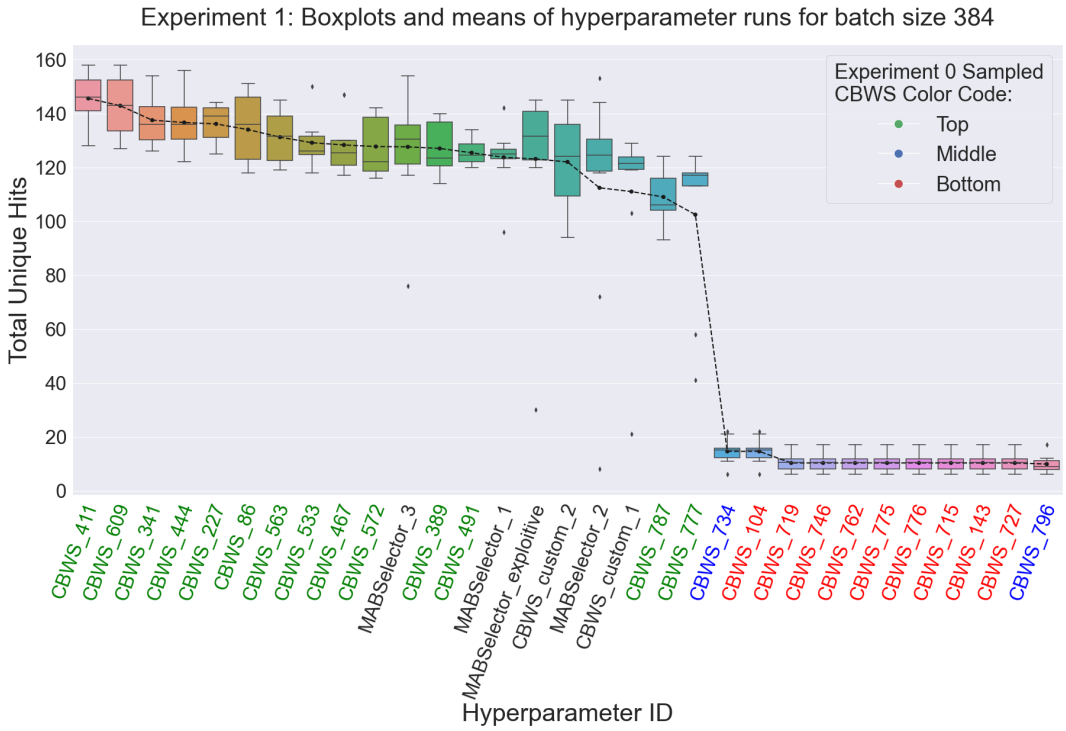


(b) Total unique hits

Figure 5.7: Experiment 1 boxplot and means of hyperparameter runs for batch size **96** for metrics (a) total hits and (b) total unique hits. Colored x-axis labels indicate Sampled CBWS performance groupings from experiment 0: top (green), middle (blue), and bottom (red).

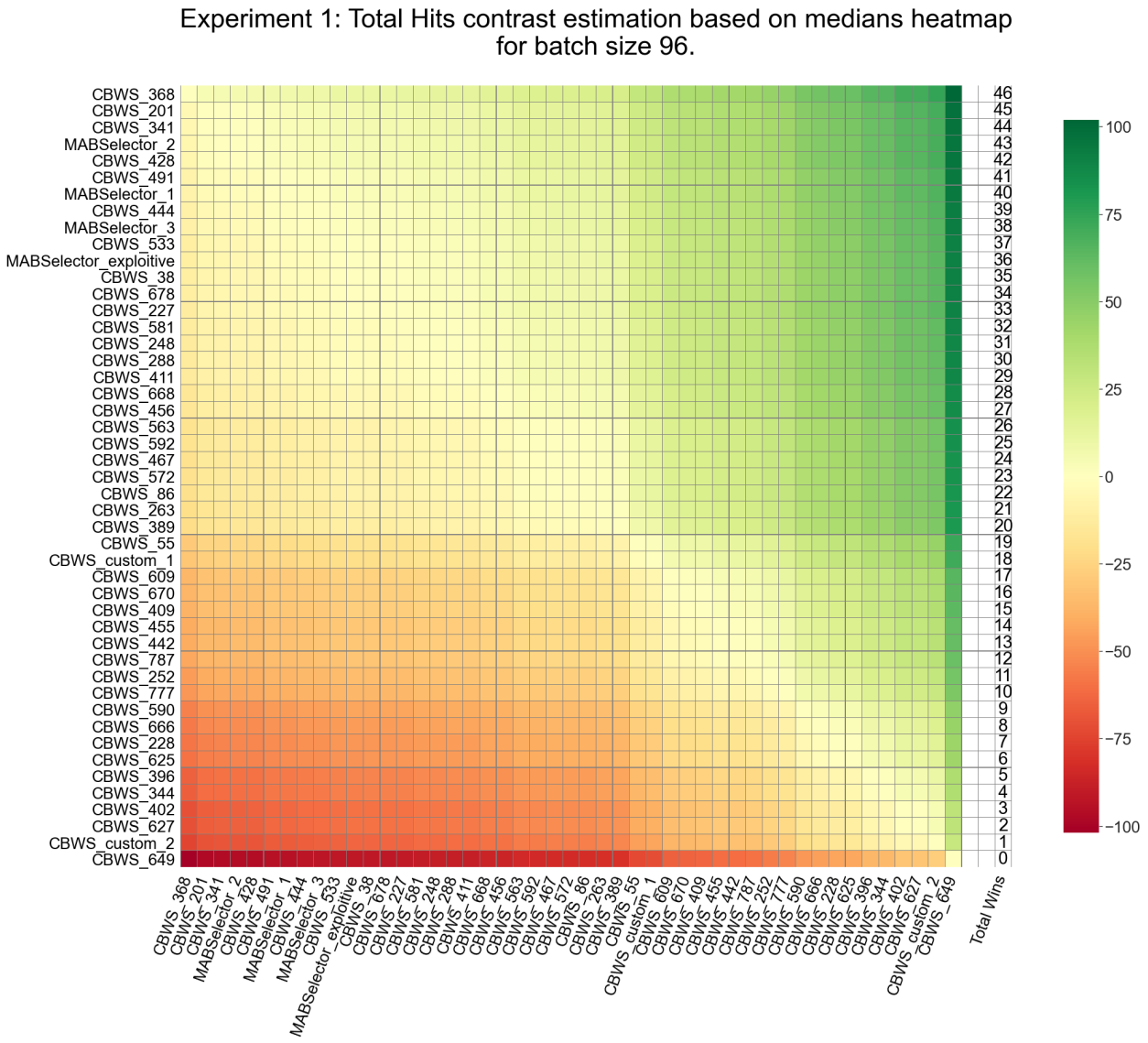


(a) Total hits



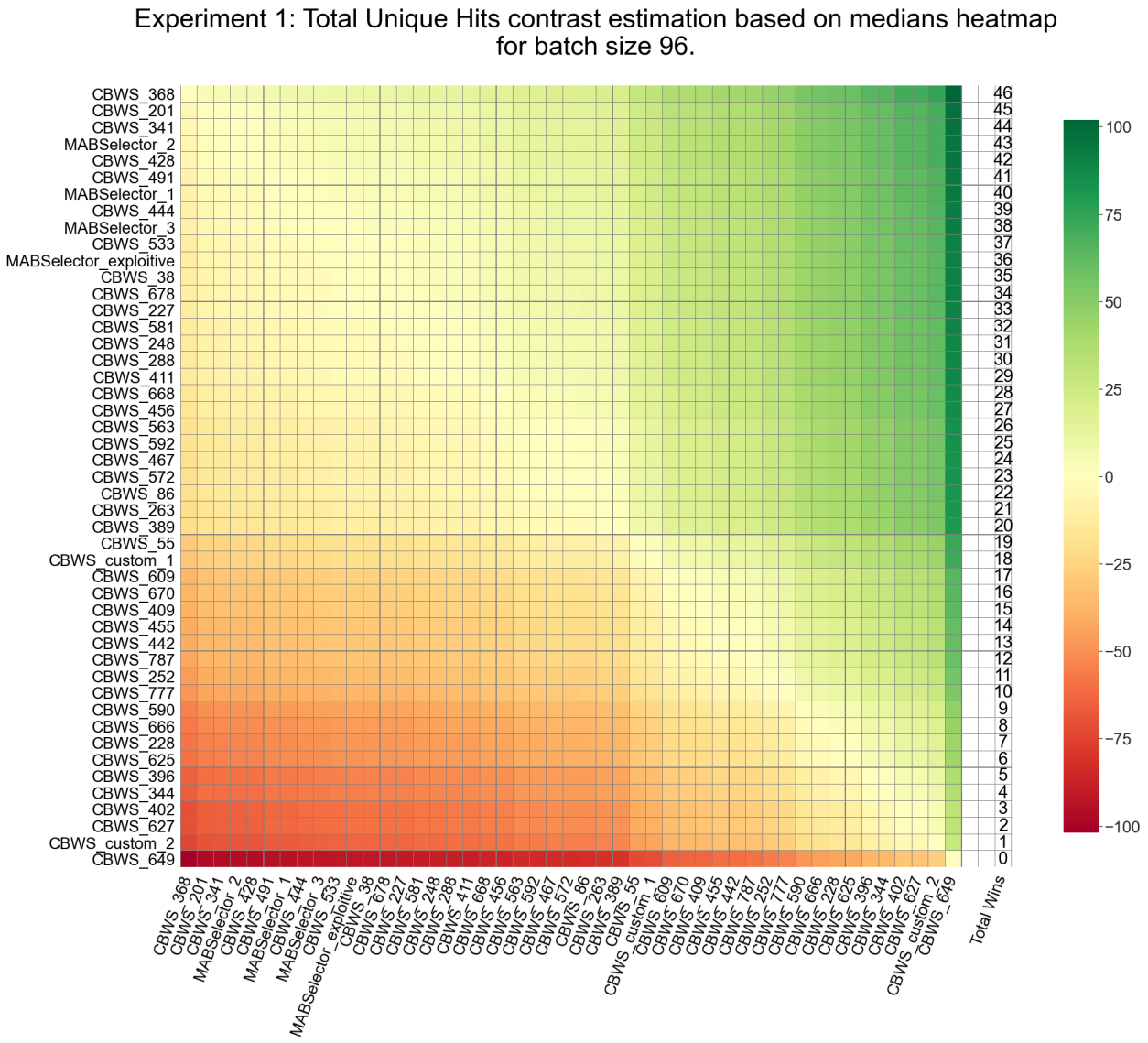
(b) Total unique hits

Figure 5.8: Experiment 1 boxplot and means of hyperparameter runs for batch size **384** for metrics (a) total hits and (b) total unique hits. Colored x-axis labels indicate Sampled CBWS performance groupings from experiment 0: top (green), middle (blue), and bottom (red).



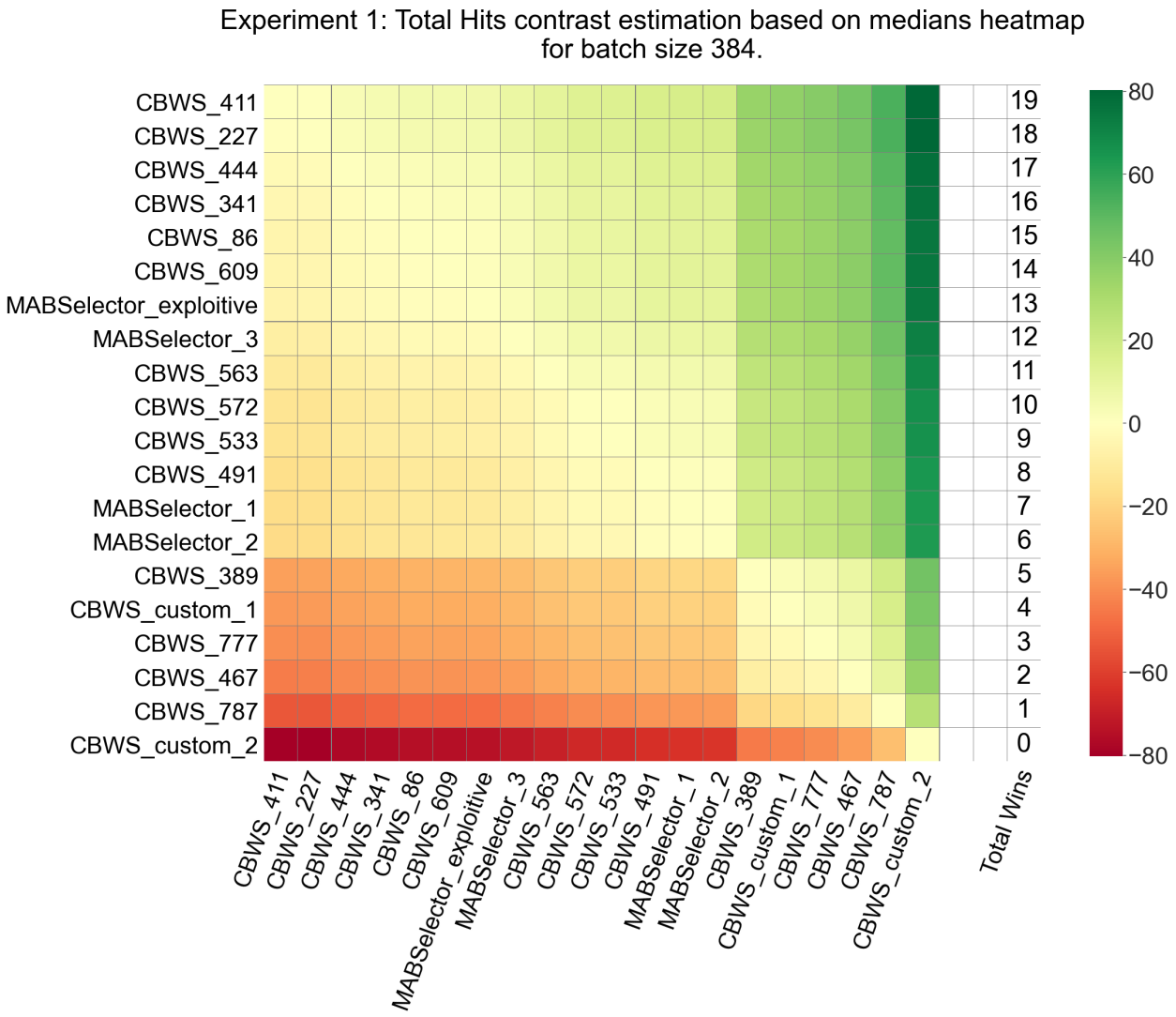
(a) Total hits

Figure 5.9: Experiment 1 contrast estimation based on medians (CEM) heatmaps for batch size **96** for metrics (a) total hits and (b) total unique hits. Total wins column indicates the number of times a strategy (row) outperforms another strategy (column) by having a positive difference. *(subfigure (b) on next page)*



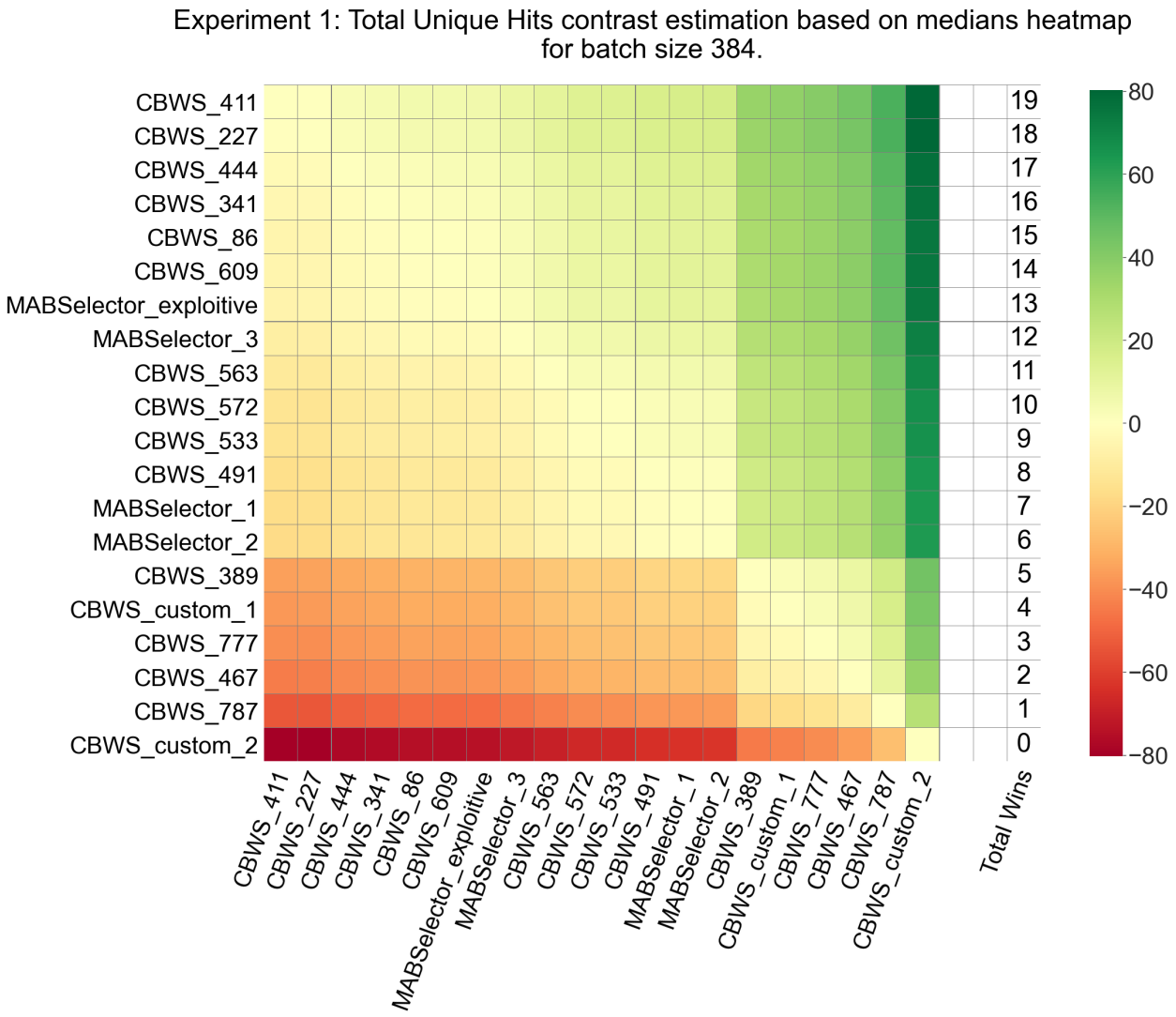
(b) Total unique hits

Figure 5.9: Experiment 1 contrast estimation based on medians (CEM) heatmaps for batch size **96** for metrics (a) total hits and (b) total unique hits. Total wins column indicates the number of times a strategy (row) outperforms another strategy (column) by having a positive difference.



(a) Total hits

Figure 5.10: Experiment 1 contrast estimation based on medians (CEM) heatmaps for batch size **384** for metrics (a) total hits and (b) total unique hits. Total wins column indicates the number of times a strategy (row) outperforms another strategy (column) by having a positive difference. *(subfigure (b) on next page)*



(b) Total unique hits

Figure 5.10: Experiment 1 contrast estimation based on medians (CEM) heatmaps for batch size **384** for metrics (a) total hits and (b) total unique hits. Total wins column indicates the number of times a strategy (row) outperforms another strategy (column) by having a positive difference.

Table 5.10: Experiment 1 metric means and standard deviations of strategies for batch size **96**. Hits by exploitation and exploration are denoted.

HS ID	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Hits std	Total Unique Hits Mean	Total Unique Hits std	Run Counts	Exp 0 Group
CBWS_368	45.60	102.90	148.50	49.67	92.70	29.89	10	top_96
CBWS_341	17.44	128.56	146.00	58.57	90.11	35.04	9	top_96
CBWS_678	0.00	145.12	145.12	39.35	96.25	25.05	8	top_96
CBWS_201	0.40	140.40	140.80	44.42	90.80	25.87	10	top_1536
CBWS_444	11.89	125.33	137.22	56.31	84.33	33.50	9	top_1536
CBWS_467	22.90	112.60	135.50	18.87	97.40	11.05	10	top_96
CBWS_428	0.00	134.33	134.33	64.86	83.33	39.27	9	top_384
CBWS_581	127.11	4.78	131.89	54.81	83.78	32.45	9	top_384
MABSel_2	130.80	0.00	130.80	61.43	80.40	37.04	10	benchmark
CBWS_533	1.38	127.25	128.62	61.43	82.12	39.31	8	top_1536
CBWS_668	32.80	94.40	127.20	64.27	80.90	40.92	10	worst_384
CBWS_227	0.00	122.33	122.33	71.59	76.00	43.47	6	top_1536
CBWS_389	0.00	120.75	120.75	58.78	81.38	39.55	8	top_384
MABSel_1	120.20	0.00	120.20	64.26	73.80	38.74	10	benchmark
CBWS_248	0.00	120.14	120.14	72.77	77.86	46.18	7	top_96
CBWS_55	0.00	119.89	119.89	33.19	90.67	24.52	9	top_384
CBWS_456	0.00	117.78	117.78	61.30	80.00	38.33	9	top_96
CBWS_491	0.00	117.50	117.50	73.90	70.62	44.26	8	top_1536
CBWS_411	0.00	117.22	117.22	65.66	77.11	41.62	9	top_384
CBWS_288	0.00	116.78	116.78	65.69	78.00	41.96	9	top_384
MABSel_exploit	111.10	0.00	111.10	72.25	67.40	43.94	10	benchmark
CBWS_38	109.10	0.60	109.70	73.61	73.20	46.46	10	top_96
CBWS_572	12.00	97.00	109.00	68.79	67.50	41.30	8	top_1536
MABSel_3	108.30	0.00	108.30	79.24	66.30	47.87	10	benchmark
CBWS_592	42.78	64.33	107.11	69.83	73.56	46.48	9	top_96
CBWS_409	0.00	104.00	104.00	53.39	79.00	41.27	10	top_96
CBWS_563	11.71	89.71	101.43	71.35	64.86	45.69	7	top_1536
CBWS_86	0.00	101.22	101.22	69.10	67.44	45.01	9	top_1536
CBWS_625	0.11	100.22	100.33	5.68	98.44	5.81	9	middle_384
CBWS_590	0.00	98.90	98.90	27.01	90.90	24.92	10	middle_96
CBWS_263	89.56	8.33	97.89	71.57	62.11	44.05	9	top_96
CBWS_777	94.71	1.57	96.29	51.87	60.14	32.45	7	top_1536
CBWS_670	0.00	96.11	96.11	56.19	73.89	42.90	9	top_384
CBWS_cstm1	43.78	48.22	92.00	64.05	60.56	41.16	9	custom_cbws
CBWS_787	88.80	1.30	90.10	56.21	58.50	36.67	10	top_96

Continued on next page

Table 5.10: Experiment 1 metric means and standard deviations of strategies for batch size **96**. Hits by exploitation and exploration are denoted.

HS ID	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Hits std	Total Unique Hits Mean	Total Unique Hits std	Run Counts	Exp 0 Group
CBWS_396	0.00	87.14	87.14	25.48	85.29	25.52	7	middle_384
CBWS_442	0.00	83.40	83.40	58.27	64.00	44.06	10	top_96
CBWS_252	67.10	13.60	80.70	53.75	51.90	33.81	10	top_384
CBWS_666	74.22	4.56	78.78	53.53	50.00	32.71	9	top_96
CBWS_455	57.40	19.70	77.10	59.81	50.90	37.83	10	top_96
CBWS_609	0.00	72.50	72.50	72.45	48.33	45.99	6	top_1536
CBWS_cstm2	0.00	63.80	63.80	40.90	60.10	39.12	10	custom_cbws
CBWS_344	21.80	37.90	59.70	48.55	46.60	38.81	10	top_1536
CBWS_402	0.00	59.12	59.12	47.43	53.88	43.45	8	middle_384
CBWS_228	57.38	0.88	58.25	57.49	37.62	36.34	8	top_384
CBWS_627	45.80	9.10	54.90	49.36	36.80	32.13	10	top_384
CBWS_649	10.11	31.33	41.44	29.25	36.44	28.22	9	middle_384
CBWS_241	10.44	0.33	10.78	4.24	8.56	3.50	9	middle_96
CBWS_19	0.00	6.75	6.75	1.58	6.62	1.30	8	middle_1536
CBWS_481	0.00	6.38	6.38	1.51	6.38	1.51	8	middle_1536
CBWS_68	0.00	5.90	5.90	1.20	5.90	1.20	10	middle_1536
CBWS_734	0.00	3.70	3.70	2.06	3.70	2.06	10	middle_1536
CBWS_104	0.00	3.70	3.70	2.06	3.70	2.06	10	worst_96
CBWS_719	0.00	2.60	2.60	1.65	2.60	1.65	10	worst_384
CBWS_746	0.00	2.60	2.60	1.65	2.60	1.65	10	worst_1536
CBWS_775	0.00	2.60	2.60	1.65	2.60	1.65	10	worst_1536
CBWS_776	0.00	2.60	2.60	1.65	2.60	1.65	10	worst_1536
CBWS_143	0.00	2.60	2.60	1.65	2.60	1.65	10	worst_1536
CBWS_715	0.00	2.60	2.60	1.65	2.60	1.65	10	worst_384
CBWS_796	0.00	2.60	2.60	1.65	2.60	1.65	10	middle_1536
CBWS_727	0.00	2.56	2.56	1.74	2.56	1.74	9	worst_1536
CBWS_762	0.00	2.56	2.56	1.74	2.56	1.74	9	worst_384

Table 5.11: Experiment 1 metric means and standard deviations of strategies for batch size **384**. Hits by exploitation and exploration are denoted.

HS ID	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Hits std	Total Unique Hits Mean	Total Unique Hits std	Run Counts	Exp 0 Group
CBWS_411	0.00	218.57	218.57	18.41	145.57	10.56	7	top_384
CBWS_444	13.38	204.88	218.25	22.47	136.62	10.86	8	top_1536
CBWS_341	17.83	200.17	218.00	18.41	137.50	10.45	6	top_96
CBWS_227	0.00	214.33	214.33	11.78	136.11	7.51	9	top_1536
CBWS_609	0.00	213.86	213.86	21.11	142.86	12.67	7	top_1536
CBWS_86	0.00	211.00	211.00	18.89	134.00	12.28	9	top_1536
CBWS_533	3.25	202.38	205.62	15.55	129.12	9.57	8	top_1536
CBWS_563	9.70	194.90	204.60	14.76	131.20	9.55	10	top_1536
MABSel_3	202.30	0.00	202.30	33.79	127.60	21.79	10	benchmark
CBWS_572	20.57	181.57	202.14	19.37	127.71	11.50	7	top_1536
CBWS_491	0.00	201.90	201.90	6.64	125.40	4.50	10	top_1536
MABSel_1	197.00	0.00	197.00	18.53	123.70	11.37	10	benchmark
MABSel_exploit	196.60	0.00	196.60	55.02	123.10	33.92	10	benchmark
CBWS_389	0.00	183.62	183.62	14.06	127.00	9.91	8	top_384
MABSel_2	178.50	0.00	178.50	67.66	112.40	42.40	10	benchmark
CBWS_467	23.30	149.50	172.80	14.90	128.30	10.81	10	top_96
CBWS_cstm1	100.30	64.20	164.50	51.90	111.00	32.42	10	custom_cbws
CBWS_787	156.57	5.00	161.57	28.52	109.00	10.72	7	top_96
CBWS_777	152.33	5.67	158.00	52.90	102.44	30.48	9	top_1536
CBWS_cstm2	0.00	134.60	134.60	23.30	122.00	18.41	10	custom_cbws
CBWS_734	0.00	14.70	14.70	4.83	14.60	4.65	10	middle_1536
CBWS_104	0.00	14.70	14.70	4.83	14.60	4.65	10	worst_96
CBWS_719	0.00	10.40	10.40	3.31	10.30	3.27	10	worst_384
CBWS_746	0.00	10.40	10.40	3.31	10.30	3.27	10	worst_1536
CBWS_762	0.00	10.40	10.40	3.31	10.30	3.27	10	worst_384
CBWS_775	0.00	10.40	10.40	3.31	10.30	3.27	10	worst_1536
CBWS_776	0.00	10.40	10.40	3.31	10.30	3.27	10	worst_1536
CBWS_715	0.00	10.40	10.40	3.31	10.30	3.27	10	worst_384
CBWS_143	0.00	10.40	10.40	3.31	10.30	3.27	10	worst_1536
CBWS_727	0.00	10.40	10.40	3.31	10.30	3.27	10	worst_1536
CBWS_796	0.00	9.88	9.88	3.52	9.88	3.52	8	middle_1536

5.16 Experiment 2: Large Dataset, 10 Initial Starting Plates, 96-plate batch size, and 50 iterations

Overview and Goal

Experiment 1 promoted 12 sampled CBWS, 3 custom CBWS, and 3 benchmark strategies. We also introduce 4 more benchmark strategies: a cluster-based random selector (ClusterBasedRandom), an instance-based random selector (InstanceBasedRandom), a dissimilar cluster-based selector (ClusterBasedDissimilar), and a dissimilar instance-based selector (InstanceBasedDissimilar). These strategies were discussed in section 5.10. In total that is $12 + 3 + 7 = 22$ strategies.

We use the same PCBA target from experiment 1. This is target aid624173 with 400,122 compounds and 487 actives. For each strategy, we perform a run with a batch size of 96 for 50 iterations for each of the 10 initial starting plates. That is a total of $22 \times 10 = 220$ jobs. We decided to use a batch size of 96 from now on due to resource and scheduling constraints (compute nodes, time, and memory). Recall from experiment 1 that the 10 initial seeding plates are exclusive 96 compounds with 1 active per plate that were randomly sampled from the dataset.

The 220 jobs were run until a deadline was reached. The default deadline for each job was set to roughly 336 hours (2 weeks). The ClusterBasedDissimilar and InstanceBasedDissimilar strategies took too long to complete for all 10 runs, and so moving forward, we remove them from the analysis. The remaining strategies finished successfully to completion for all runs. Table 5.12 summarizes the strategy groups and success counts. The **goal** is to compare and prune strategies per hyperparameter group when run for more iterations (i.e. from 10 to 50 iterations).

Table 5.12: Experiment 2 summary of successfully completed strategies. Success denotes that all 10 initial plate runs completed 50 iterations.

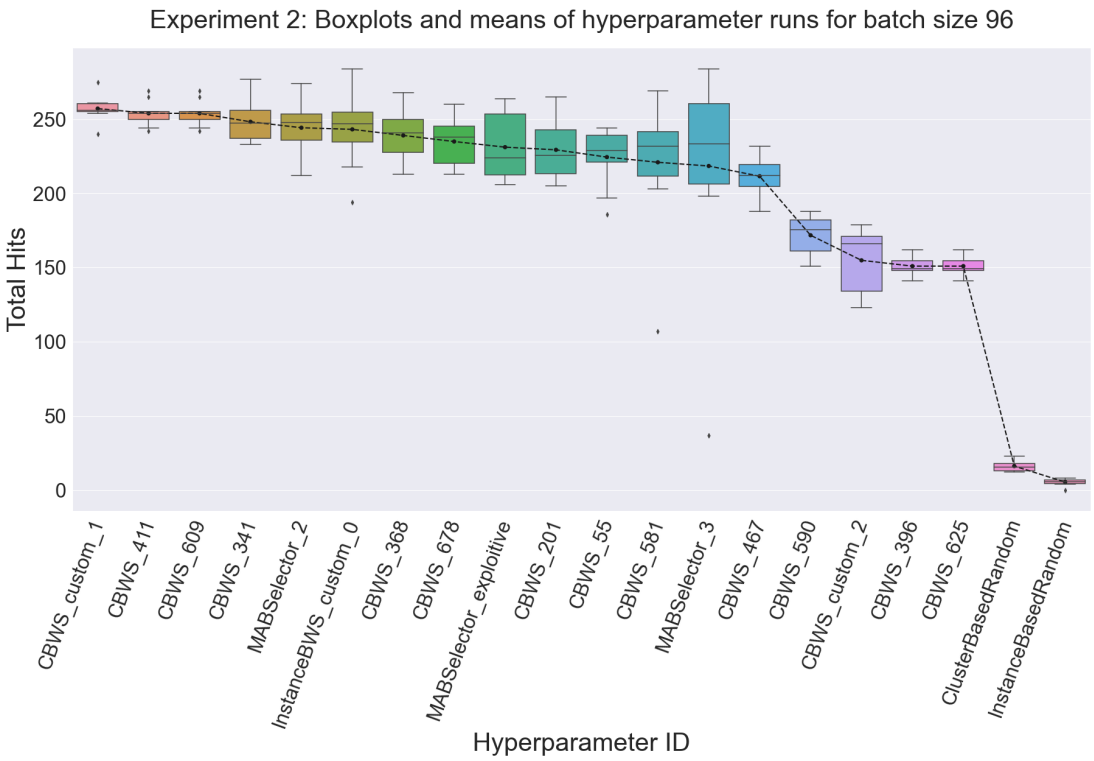
Group	Total	Success Batch Size 96
Sampled CBWS	12	12
Custom CBWS	3	3
Benchmark	7	5
Total	22	20

Results and Discussion

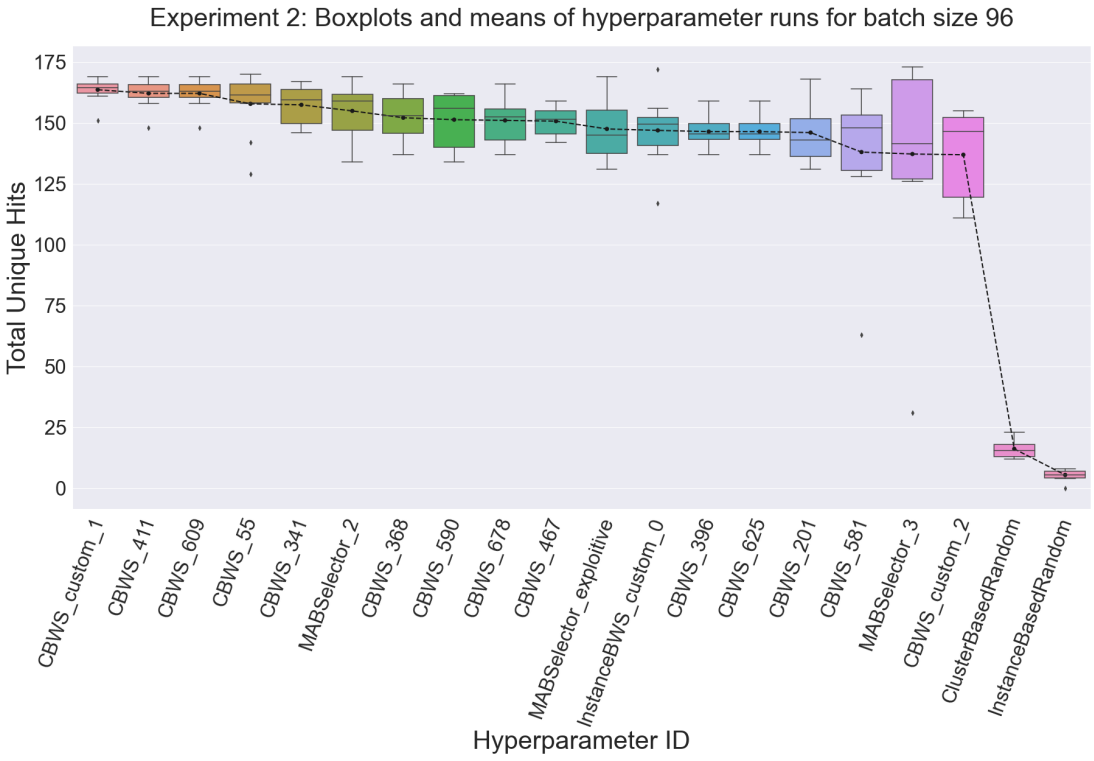
The results are summarized in the boxplots in Figure 5.11. Summary of exploitation vs exploration hits are shown in Table 5.13. CBWS_custom_1 is the best performing in mean total hits and mean total unique hits. From the Sampled CBWS strategies, the exploration heavy settings are high performing. As expected, the random-sampling benchmarks are clearly the worst performers. Figure 5.12 shows the CEM heatmaps and total wins for the strategies, the latter of which we use for promotion. We again focus on pruning strategies from each group: Sampled CBWS, Custom CBWS, and Benchmarks. The promotion list to experiment 3 is as follows:

- **Custom CBWS**: CBWS_custom_1 since it has highest CEM wins rank on both total hits and unique hit.
- **Sampled CBWS**: CBWS_609 high CEM wins rank, CBWS_55 second best Sampled CBWS for CEM total unique hits, and CBWS_341 second best Sampled CBWS for CEM total hits and third best Sampled CBWS for CEM total unique hits. We removed CBWS_411 due to large similarity in parameter configuration with CBWS_609; inspection confirmed almost complete overlap in compound selection.
- **Benchmarks**: ClusterBasedRandom and InstanceBasedRandom for benchmark comparisons and they are fast to run. MABSelector_2 since it is high performing on both metrics, and MABSelector_exploitive for comparison to a simple exploitive strategy.

Finally, Figure 5.13 shows the exploitation and exploration cumulative hits progression for each strategy for all 50 iterations. These are 95% confidence interval (CI) point plots that estimate the cumulative mean of each iteration over the 10 initial plate runs and provides error bar uncertainty for the mean. Many of the top performing strategies exhibit rapid hit accumulation in the first couple of iterations, followed by a linear accumulation of hits up until around iteration 30, after which they exhibit a plateau. This is expected since there is a finite number of hits in any given pool.

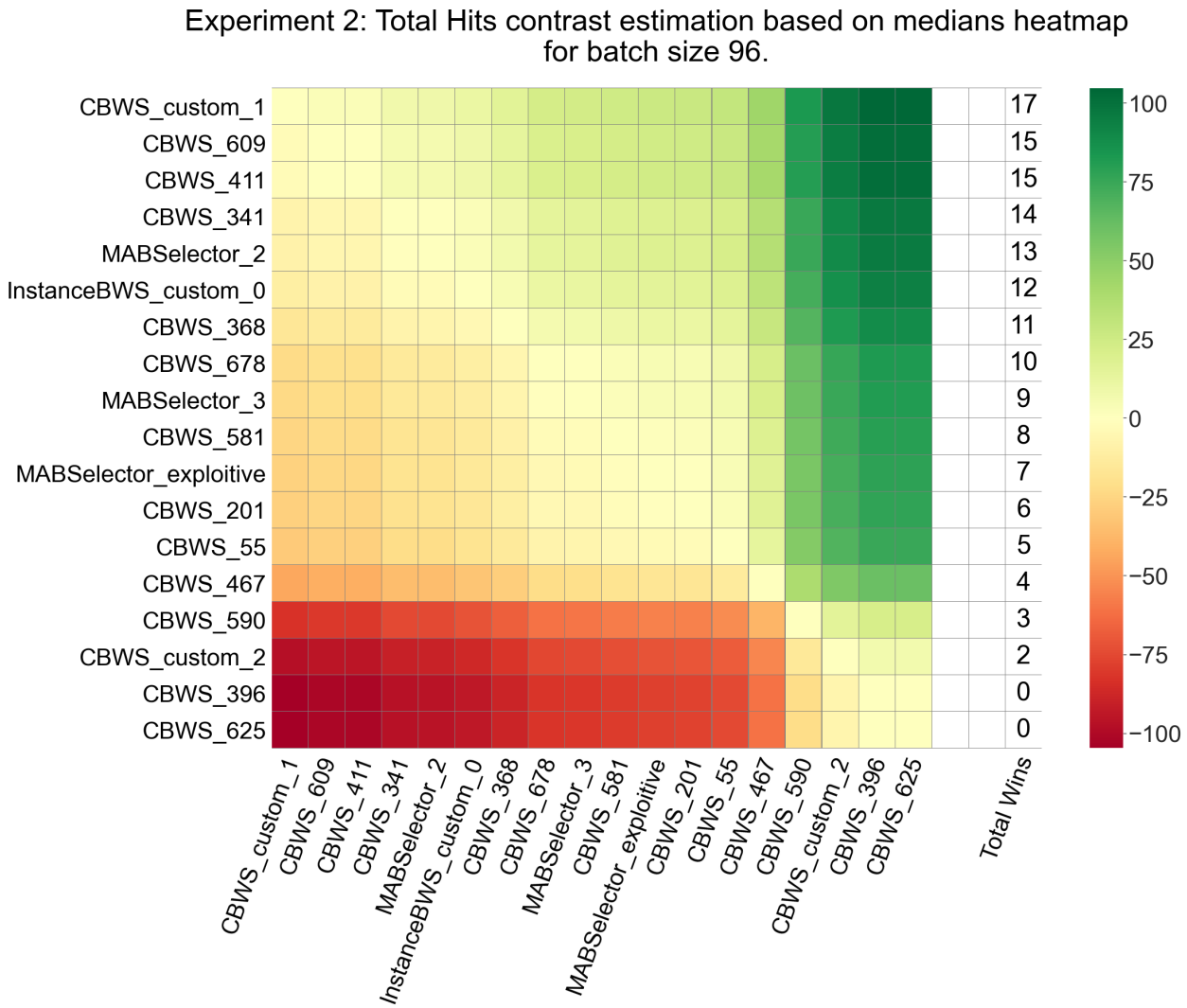


(a) Total hits



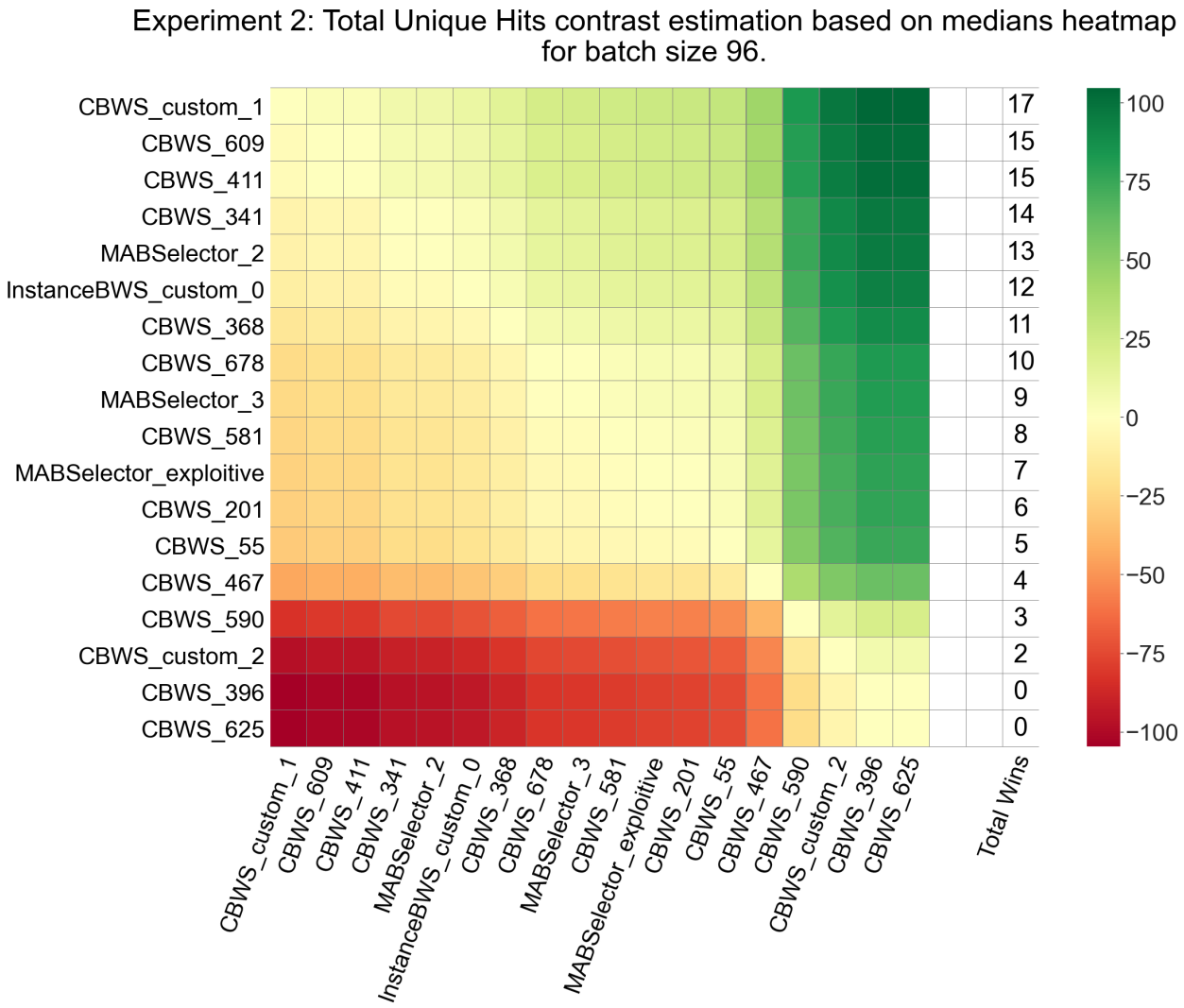
(b) Total unique hits

Figure 5.11: Experiment 2 boxplots and means of strategy runs for **batch size 96** for metrics (a) total hits and (b) total unique hits.



(a) Total hits

Figure 5.12: Experiment 2 contrast estimation based on medians (CEM) heatmaps for **batch size 96** for metrics (a) total hits and (b) total unique hits. Total wins column indicates the number of times a strategy (row) outperforms another strategy (column) by having a positive difference. *(subfigure (b) on next page)*



(b) Total unique hits

Figure 5.12: Experiment 2 contrast estimation based on medians (CEM) heatmaps for **batch size 96** for metrics (a) total hits and (b) total unique hits. Total wins column indicates the number of times a strategy (row) outperforms another strategy (column) by having a positive difference.

Table 5.13: Experiment 2 metric means and standard deviations sorted by total hits mean.

HS ID	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Unique Hits Mean	Total Hits std	Total Unique Hits std
CBWS_custom_1	143.1	114.1	257.2	163.6	8.61	5.17
CBWS_609	0.0	254.0	254.0	162.1	8.30	5.93
CBWS_411	0.0	254.0	254.0	162.1	8.30	5.93
CBWS_341	21.8	226.5	248.3	157.4	13.47	7.71
MABSelector_2	244.3	0.0	244.3	154.9	19.06	11.68
InstBWS_custom_0	120.4	122.8	243.2	146.9	24.72	14.26
CBWS_368	63.0	176.1	239.1	152.1	18.16	10.00
CBWS_678	0.0	235.0	235.0	151.0	15.82	9.46
MABSel_exploitive	231.2	0.0	231.2	147.5	22.93	13.39
CBWS_201	5.5	223.9	229.4	146.0	20.39	12.57
CBWS_55	0.0	224.5	224.5	157.8	19.35	12.71
CBWS_581	211.3	9.7	221.0	138.0	44.74	29.26
MABSelector_3	218.5	0.0	218.5	137.2	69.74	42.01
CBWS_467	34.7	176.9	211.6	150.7	12.82	6.09
CBWS_590	0.0	171.8	171.8	151.3	13.47	11.50
CBWS_custom_2	0.0	154.9	154.9	136.9	21.43	18.17
CBWS_625	0.0	151.0	151.0	146.4	6.34	6.67
CBWS_396	0.0	151.0	151.0	146.4	6.34	6.67
ClusterBasedRandom	0.0	16.2	16.2	16.1	4.05	3.90
InstanceBasedRandom	0.0	5.4	5.4	5.4	2.41	2.41

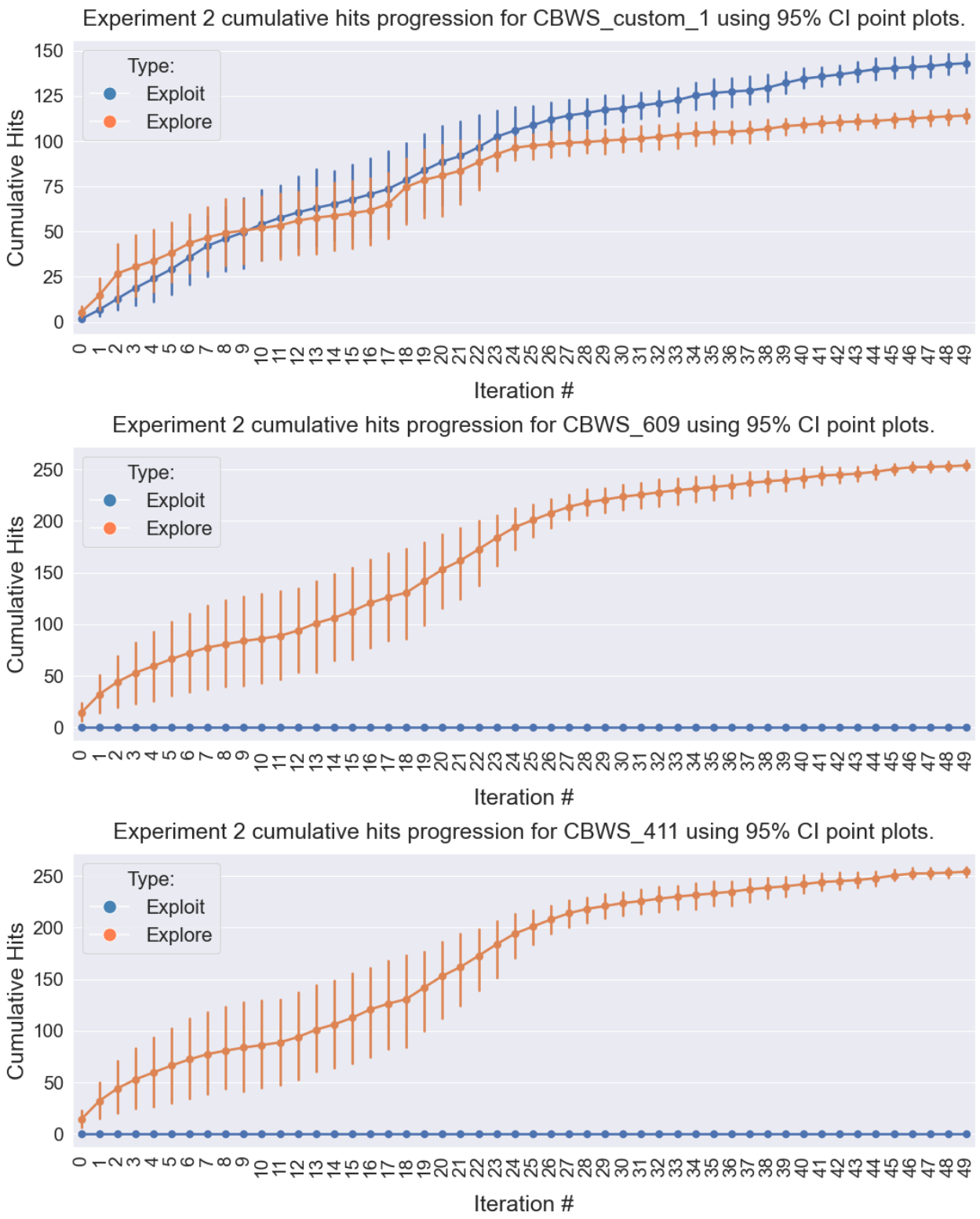


Figure 5.13: Experiment 2 exploitation and exploration cumulative hits progression per-iteration using 95% CI point plots (**batch size 96**). *(1 of 7 cont.)*

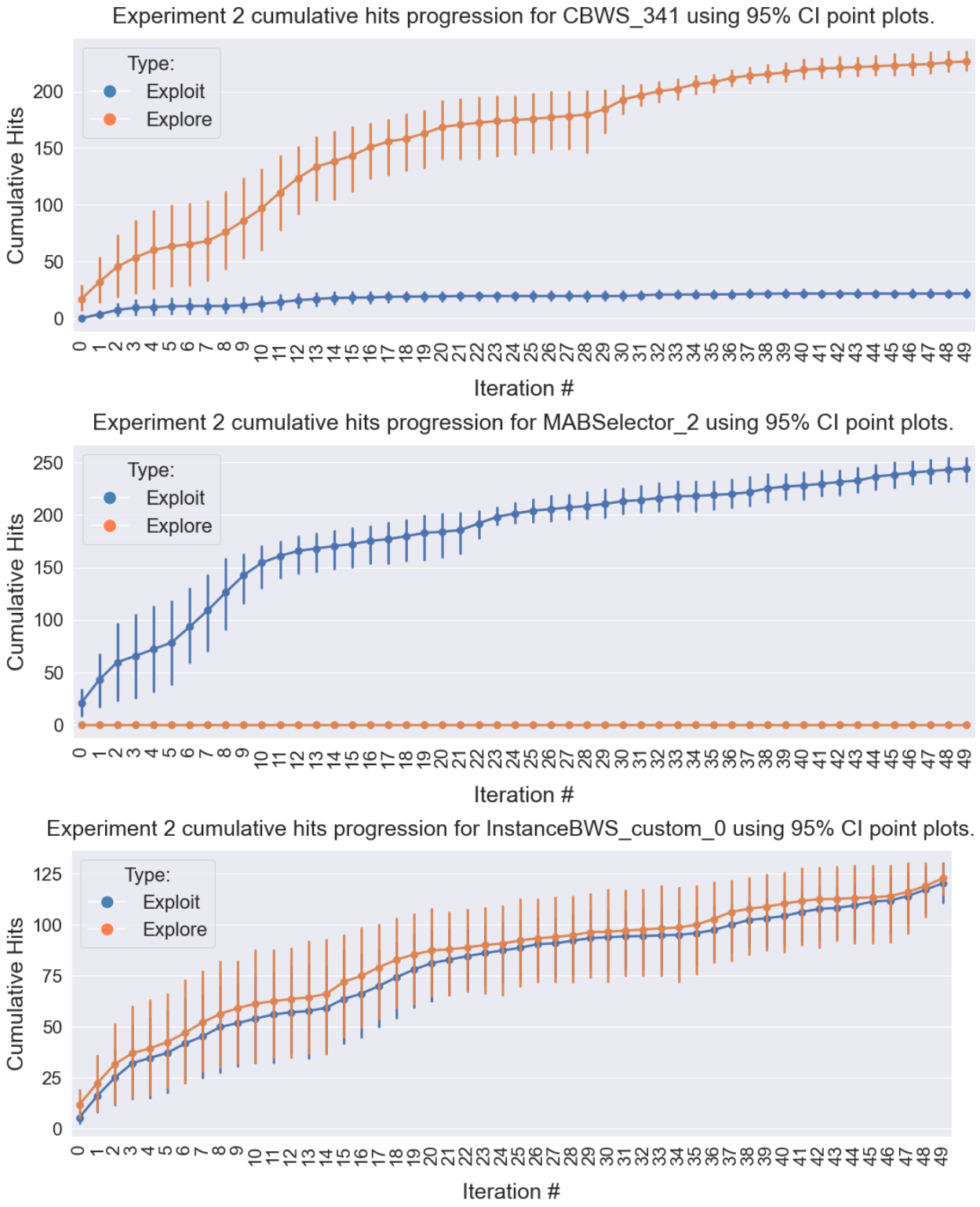


Figure 5.13: Experiment 2 exploitation and exploration cumulative hits progression per-iteration using 95% CI point plots (**batch size 96**). *(2 of 7 cont.)*

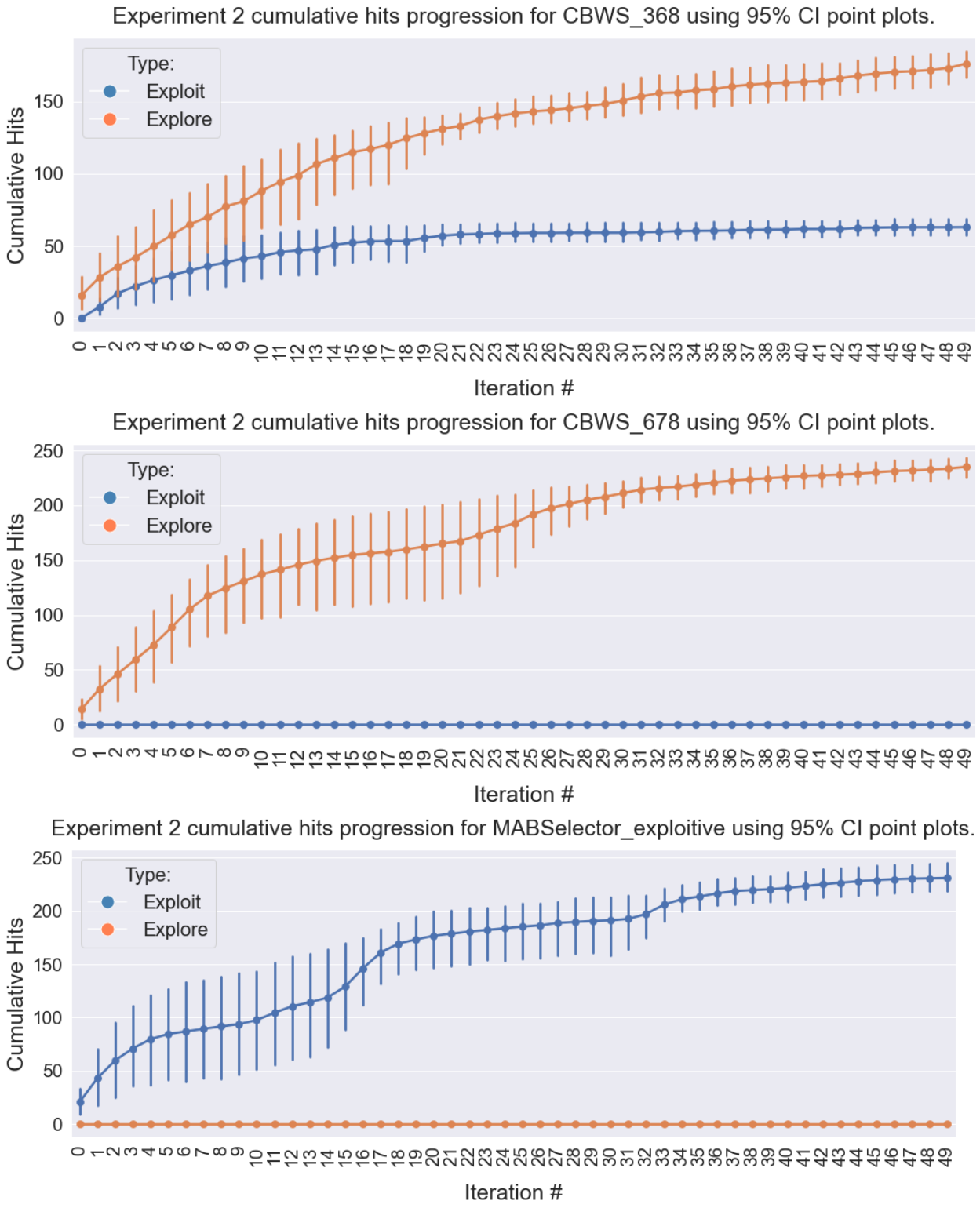


Figure 5.13: Experiment 2 exploitation and exploration cumulative hits progression per-iteration using 95% CI point plots (**batch size 96**). *(3 of 7 cont.)*

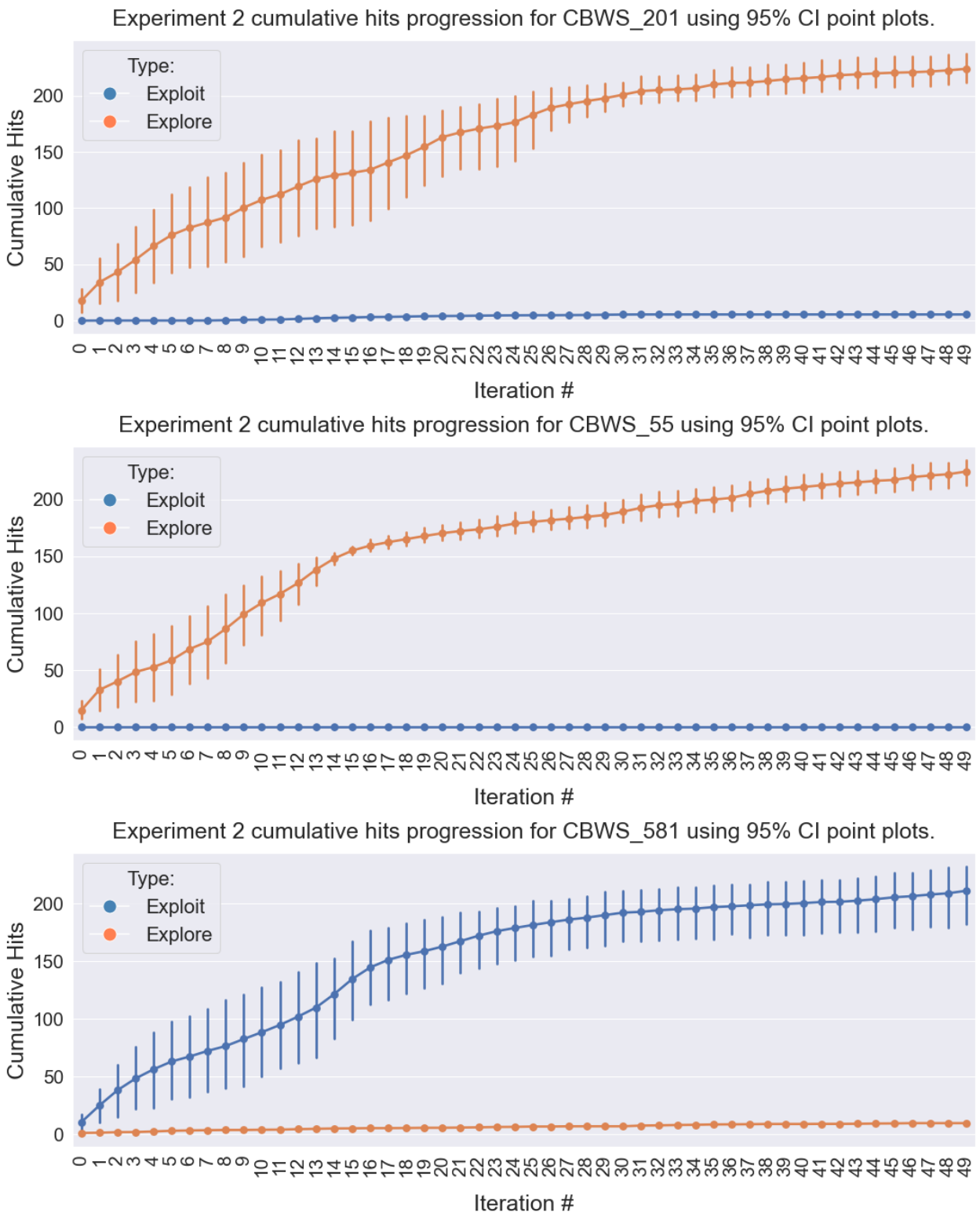


Figure 5.13: Experiment 2 exploitation and exploration cumulative hits progression per-iteration using 95% CI point plots (**batch size 96**). *(4 of 7 cont.)*

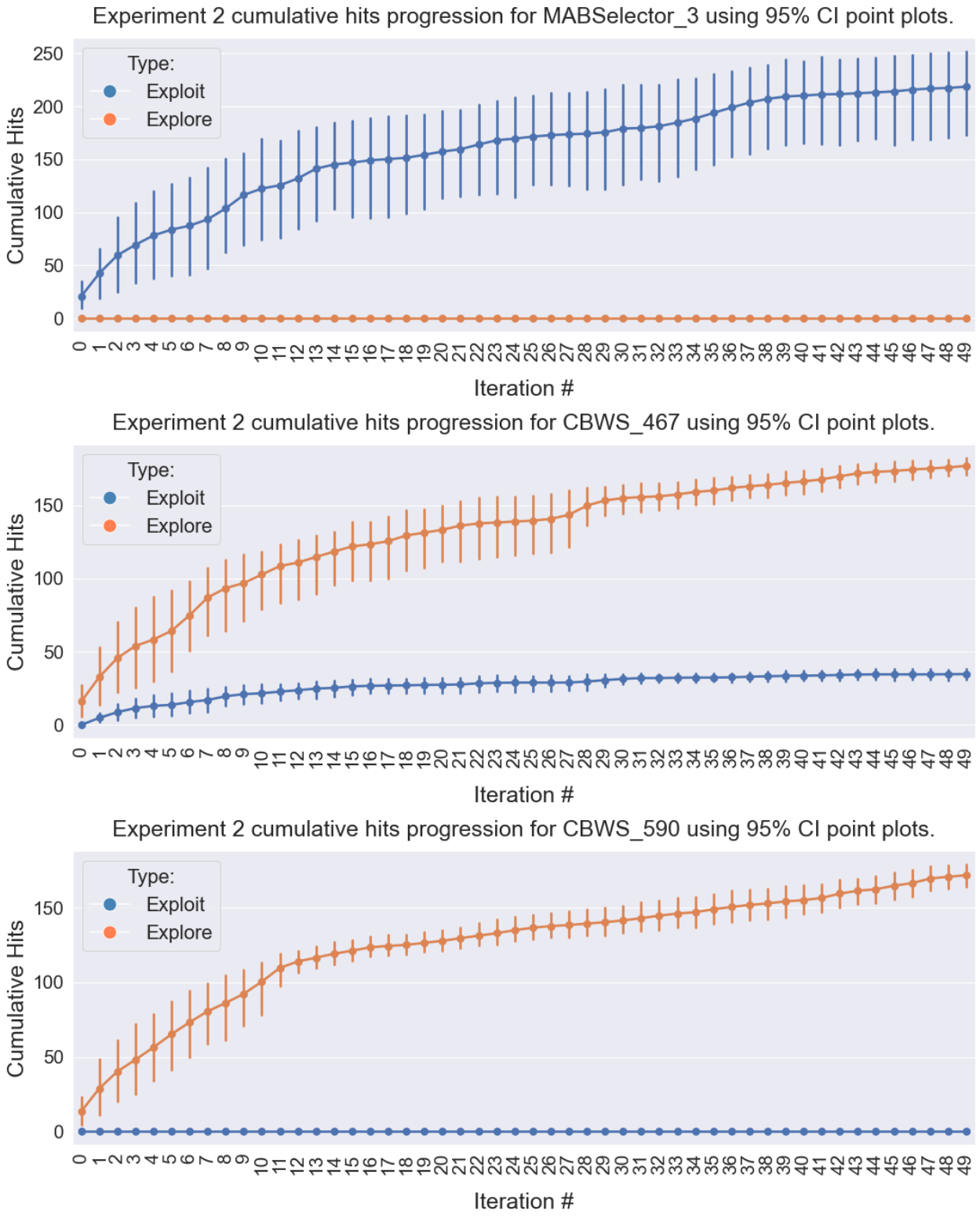


Figure 5.13: Experiment 2 exploitation and exploration cumulative hits progression per-iteration using 95% CI point plots (**batch size 96**). *(5 of 7 cont.)*

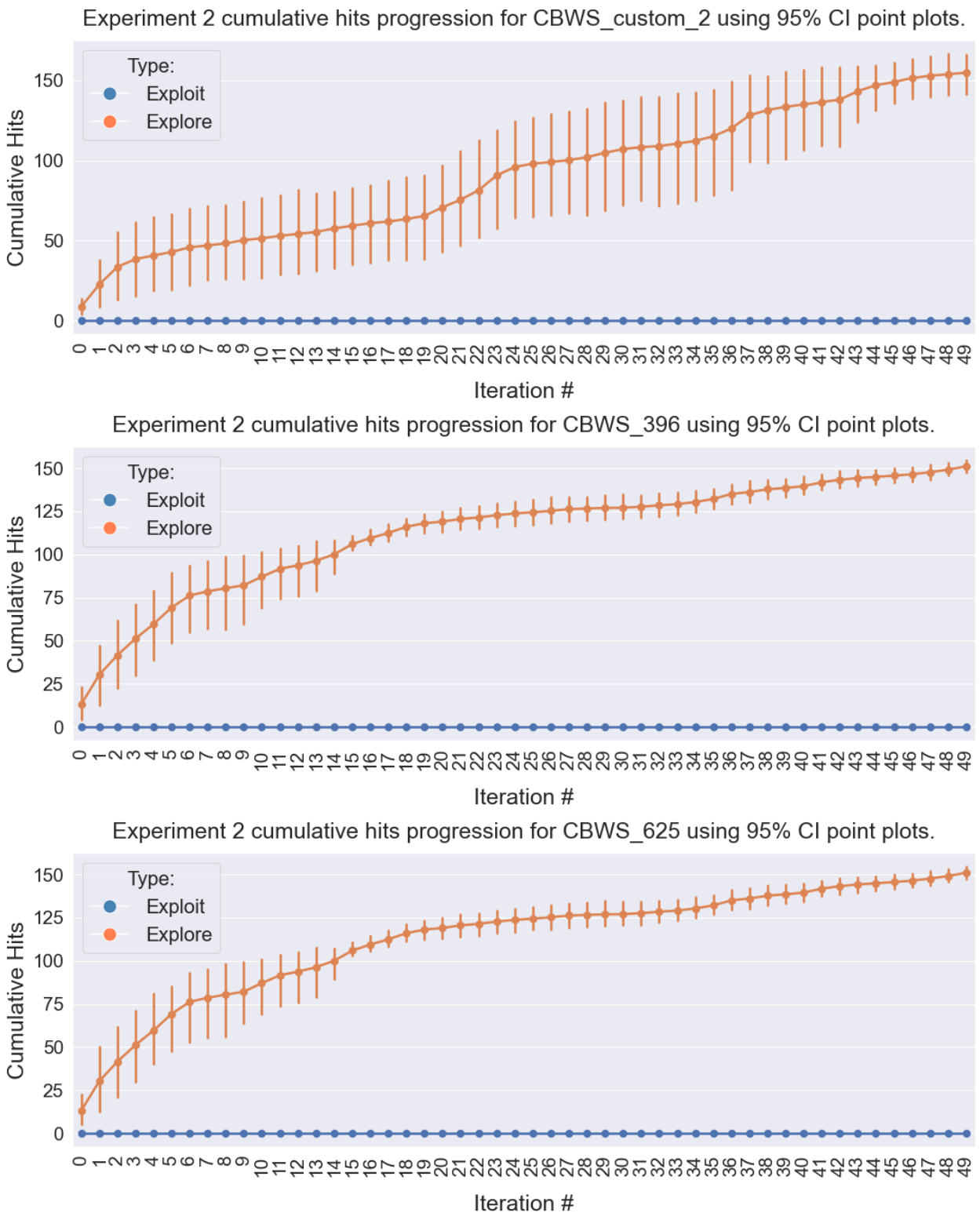


Figure 5.13: Experiment 2 exploitation and exploration cumulative hits progression per-iteration using 95% CI point plots (**batch size 96**). *(6 of 7 cont.)*

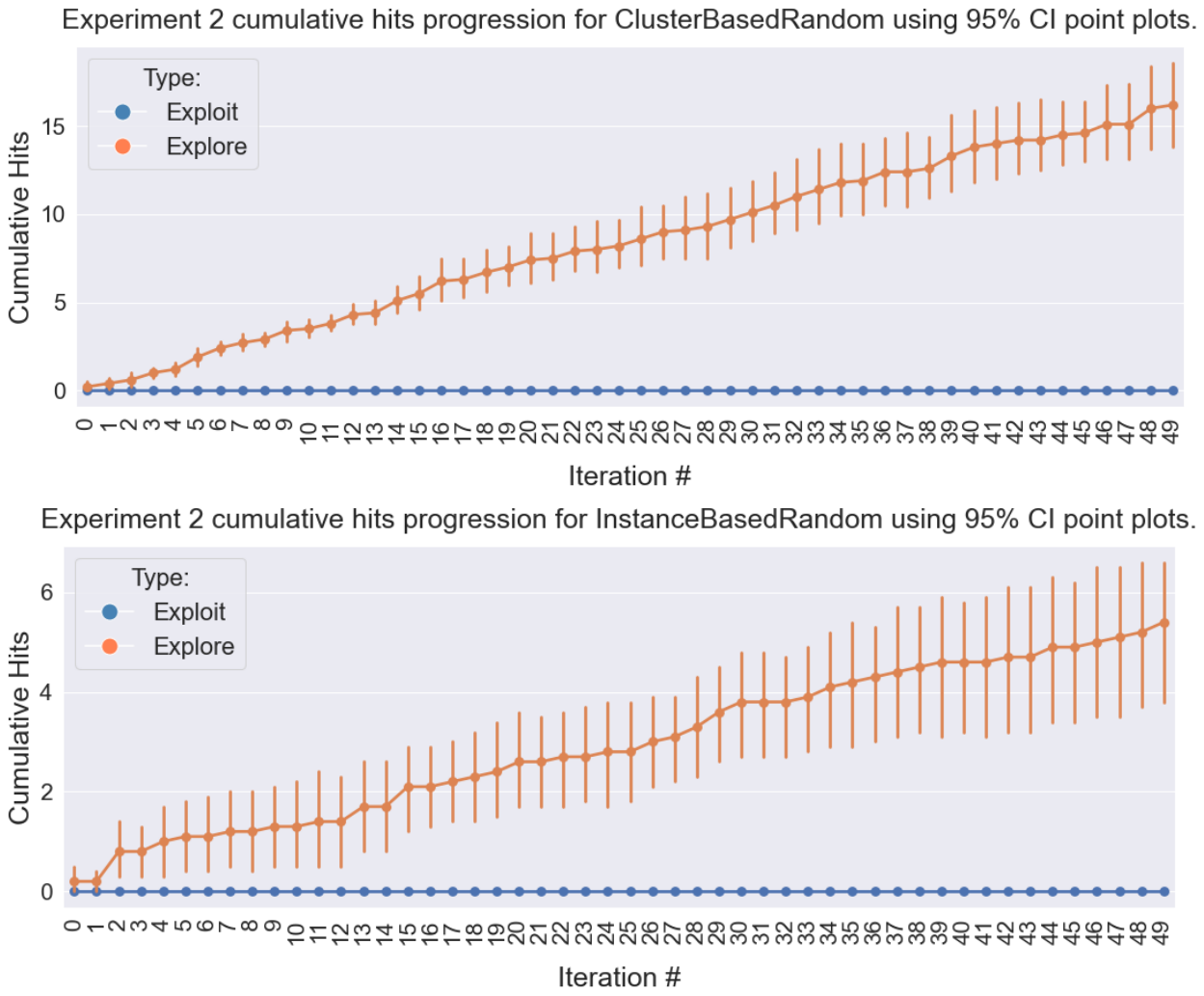


Figure 5.13: Experiment 2 exploitation and exploration cumulative hits progression per-iteration using 95% CI point plots (**batch size 96**). *(7 of 7)*

5.17 Experiment 3: 107 Target Datasets, 96-plate batch size, and 50 iterations

Experiment 2 promoted 8 total strategies: 1 custom CBWS, 4 benchmarks, and 3 sampled CBWS. Now we expand the analysis towards a diverse set of 128 target datasets: 128-PCBA [5, 4]. We limit the targets to those that have more than 100,000 compounds which amounts to 107 valid targets (see Table 5.4). Recall that we have already used aid624173 in previous experiments so we drop it here. We perform two experiments on these 107 targets: experiments 3.1 and 3.2. The overall **goal** of experiment 3 is to select **one** final strategy by analyzing performance on a diverse set of targets. The selected strategy is then run on a prospective target in experiment 4.

Experiment 3.1: 107 Target Datasets, 10 Initial Starting Plates, 96-plate batch size, and 50 iterations

For experiment 3.1, 10 initial 96-compound seeding plates have been randomly prepared (with 1 active in each plate) independently for each of the 107 targets. We run each strategy on each target-plate combination for 50 iterations and batch size of 96. This is a total of $8 \times 107 \times 10 = 8560$ jobs.

In this experiment we wanted to ensure that all jobs were run successfully to completion. Therefore, running these jobs on a compute cluster with time and resource limitations required splitting some of these single jobs into a sequence of multiple jobs, chained one after the other. Each sequence of job would run a number of the iterations from the total 50. All 8560 jobs finished successfully after roughly a month. This includes idle time in which a job waits to be scheduled on a compute node, and transfer time in which a job's files are transferred to and from the compute node. In terms of CPU compute hours, the job average was 23.98 CPU hours with a standard deviation of 24.39 CPU hours, and the max was 112.15 CPU hours.

The results for experiment 3.1 have three dimensions that define a single experimental unit: the strategy, the target, the initial starting plate. Five targets were removed from the analysis since

they had high active ratios (6% to 20%) leading to skewed results; i.e. outliers (see Figure 5.14). We are interested in comparing strategies across a diverse set of targets. To this end, we can look at the results from various perspectives: single-point per strategy metrics and per strategy-task combination metrics.

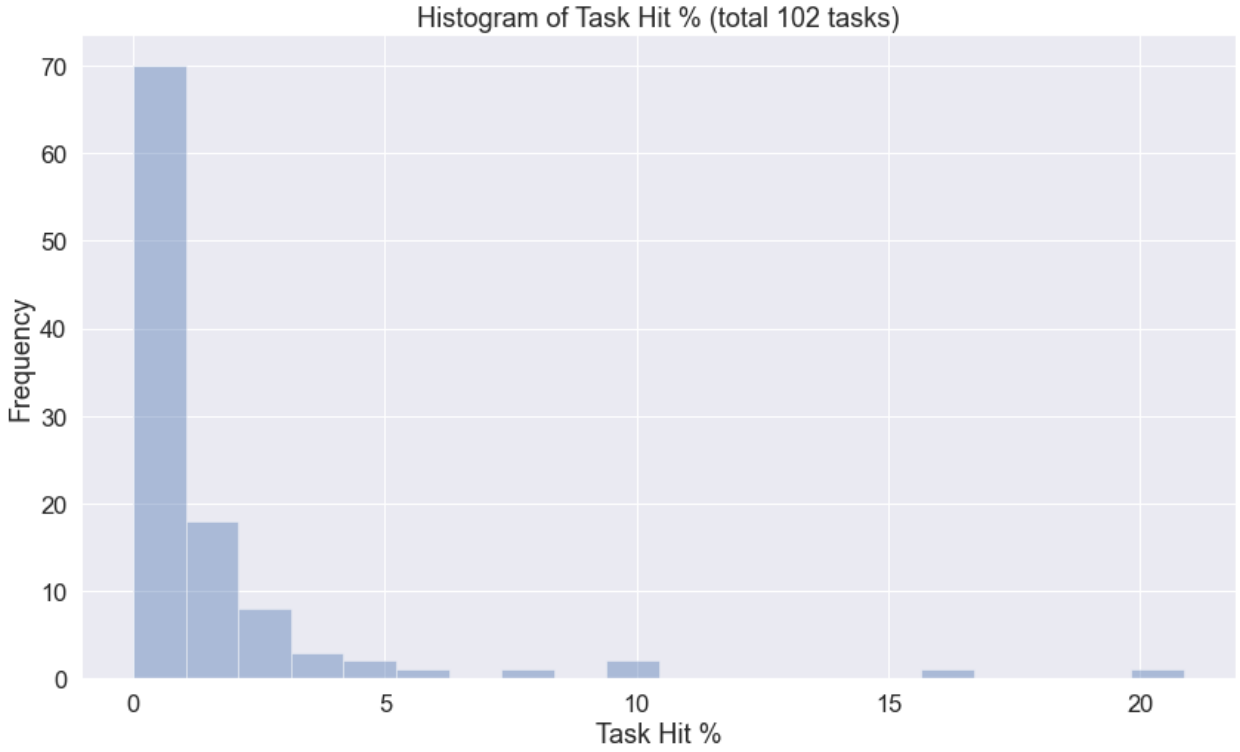


Figure 5.14: Task/Target hit % histogram for the 107 PCBA target datasets.

Single-point per strategy metrics can be derived via grouping by strategy and aggregating across target-plate combinations. Specifically, each strategy has 102 tasks and 10 seeding plates per task; that is a total of 1020 experimental units/results per strategy. For example, the single-point mean metric for a strategy will be the mean of its 1020 runs. Table 5.14 summarizes the total hits and total unique hits of single-point aggregates per strategy for 10, 20, 30, 40, and 50 iterations. When sorted by mean total hits, MABSelector_exploitive is the best across iteration limits. For the total unique hits metric, the exploration heavy CBWS_609 and CBWS_55 strategies are the best across iteration limits. The standard deviation of both metrics is not useful here since we have aggregated across different tasks that vary in their activity ratios. To further touch upon this, basing our promotion

on single-point metrics across a variety of tasks that can vary in *hardness* is a cause for concern. It could be the case that a strategy performs exceedingly well on one task which can cover its bad performance on other tasks.

Table 5.14: Experiment 3.1 single-point per strategy metric summaries at the end of 10, 20, 30, 40, and 50 iterations. These metrics are aggregated over all task-plate runs per strategy (1020 runs per strategy). *(1 of 2 cont.)*

# iterations: 10	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Unique Hits Mean	Total Hits std	Total Unique Hits std
MABSel_exploitive	103.92	0.00	103.92	56.94	140.20	66.67
MABSelector_2	100.05	0.00	100.05	53.98	132.56	60.54
CBWS_341	19.04	80.09	99.13	58.27	131.85	69.53
CBWS_609	0.00	89.83	89.83	69.49	105.22	85.93
CBWS_55	0.00	79.97	79.97	68.54	96.76	88.43
CBWS_custom_1	24.11	37.46	61.58	47.85	90.01	69.56
ClusBasedRandom	0.00	11.86	11.86	11.84	15.27	15.25
InstBasedRandom	0.00	8.85	8.85	8.82	12.00	11.95
# iterations: 20	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Unique Hits Mean	Total Hits std	Total Unique Hits std
MABSel_exploitive	220.75	0.00	220.75	120.61	285.13	134.15
MABSelector_2	210.89	0.00	210.89	114.71	264.77	122.54
CBWS_341	41.52	166.66	208.17	120.10	263.20	135.48
CBWS_609	0.00	190.58	190.58	137.60	214.84	164.18
CBWS_55	0.00	167.79	167.79	139.00	194.97	173.81
CBWS_custom_1	66.82	88.96	155.78	111.85	206.30	148.11
ClusBasedRandom	0.00	23.57	23.57	23.51	29.96	29.86
InstBasedRandom	0.00	17.57	17.57	17.47	23.43	23.27
# iterations: 30	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Unique Hits Mean	Total Hits std	Total Unique Hits std
MABSel_exploitive	320.37	0.00	320.37	176.48	410.89	193.07
MABSelector_2	307.85	0.00	307.85	170.52	381.46	180.67
CBWS_341	59.17	242.97	302.13	174.23	383.12	194.43
CBWS_609	0.00	278.91	278.91	195.18	314.80	232.15
CBWS_custom_1	110.23	135.00	245.23	169.87	313.41	216.71
CBWS_55	0.00	242.11	242.11	197.14	282.29	248.46
ClusBasedRandom	0.00	35.37	35.37	35.22	44.54	44.33
InstBasedRandom	0.00	26.28	26.28	26.05	34.64	34.25

Table 5.14: Experiment 3.1 single-point per strategy metric summaries at the end of 10, 20, 30, 40, and 50 iterations. These metrics are aggregated over all task-plate runs per strategy (1020 runs per strategy). *(2 of 2)*

# iterations: 40	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Unique Hits Mean	Total Hits std	Total Unique Hits std
MABSel_exploitive	409.24	0.00	409.24	226.36	530.52	246.86
MABSelector_2	393.51	0.00	393.51	219.77	491.88	234.40
CBWS_341	73.60	310.69	384.28	221.49	492.37	246.82
CBWS_609	0.00	356.69	356.69	244.59	407.72	291.59
CBWS_custom_1	150.69	174.73	325.41	220.93	410.52	277.19
CBWS_55	0.00	306.13	306.13	245.96	358.12	311.02
ClusBasedRandom	0.00	46.96	46.96	46.69	59.01	58.62
InstBasedRandom	0.00	34.78	34.78	34.35	45.79	45.07
# iterations: 50	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Unique Hits Mean	Total Hits std	Total Unique Hits std
MABSel_exploitive	489.49	0.00	489.49	271.57	643.15	297.50
MABSelector_2	472.26	0.00	472.26	265.74	597.22	286.97
CBWS_341	85.79	372.69	458.49	264.06	594.69	294.77
CBWS_609	0.00	425.95	425.95	287.29	494.00	342.49
CBWS_custom_1	187.75	210.00	397.75	266.50	498.85	329.81
CBWS_55	0.00	363.25	363.25	288.14	428.78	365.09
ClusBasedRandom	0.00	58.88	58.88	58.49	73.67	73.09
InstBasedRandom	0.00	43.65	43.65	42.98	57.44	56.28

Another perspective is to then group/bin *hard* tasks based on hit % and compare single-point metrics per strategy. We split the tasks into bins of size 14 based on hit %. Table 5.15 showcases the total hits and total unique hits single-point aggregates per strategy for 50 iterations for each of these task bins. For small hit %, i.e. 0.01% to 0.15%, total hits differences are not apparent. Similarly, total unique hits differences are not apparent until after 1.06%.

Table 5.15: Experiment 3.1 binned/grouped tasks single-point per strategy metric summaries at the end of 50 iterations. These metrics are aggregated over all task-plate runs per strategy (1020 runs per strategy). *(1 of 3 cont.)*

0.01%-to-0.06% total: 14 tasks	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Unique Hits Mean	Total Hits std	Total Unique Hits std
CBWS_609	0.00	16.89	16.89	12.62	12.32	9.97
MABSel_exploitive	16.46	0.00	16.46	11.76	14.11	10.77
CBWS_341	0.18	16.03	16.21	12.03	14.93	12.09
MABSel_2	15.77	0.00	15.77	11.24	14.04	10.25
CBWS_55	0.00	13.43	13.43	11.13	9.69	8.33
CBWS_custom_1	7.34	4.42	11.76	8.74	9.38	7.41
ClusBasedRandom	0.00	2.89	2.89	2.89	2.21	2.21
InstBasedRandom	0.00	1.66	1.66	1.65	1.50	1.48
0.07%-to-0.14% total: 14 tasks	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Unique Hits Mean	Total Hits std	Total Unique Hits std
MABSel_exploitive	34.01	0.00	34.01	24.75	25.26	16.88
MABSel_2	33.69	0.00	33.69	24.83	24.02	16.43
CBWS_341	0.04	32.82	32.86	24.29	24.38	16.36
CBWS_609	0.00	30.27	30.27	23.79	22.19	15.65
CBWS_55	0.00	23.83	23.83	19.81	14.13	11.05
CBWS_custom_1	13.29	9.90	23.19	18.00	18.42	12.27
ClusBasedRandom	0.00	6.24	6.24	6.22	3.32	3.31
InstBasedRandom	0.00	4.94	4.94	4.91	2.68	2.66
0.15%-to-0.28% total: 14 tasks	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Unique Hits Mean	Total Hits std	Total Unique Hits std
MABSel_exploitive	148.70	0.00	148.70	103.87	66.83	39.47
MABSel_2	142.46	0.00	142.46	100.39	67.79	40.49
CBWS_609	0.00	139.63	139.63	101.98	62.03	38.03
CBWS_341	2.66	136.79	139.44	99.59	62.97	37.75
CBWS_55	0.00	114.18	114.18	90.29	56.92	40.87
CBWS_custom_1	62.96	50.69	113.66	83.37	65.61	42.17
ClusBasedRandom	0.00	16.69	16.69	16.64	5.11	5.07
InstBasedRandom	0.00	10.04	10.04	9.98	3.47	3.43

Table 5.15: Experiment 3.1 binned/grouped tasks single-point per strategy metric summaries at the end of 50 iterations. These metrics are aggregated over all task-plate runs per strategy (1020 runs per strategy). *(2 of 3 cont.)*

0.29%-to-0.42% total: 14 tasks	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Unique Hits Mean	Total Hits std	Total Unique Hits std
MABSel_exploitive	234.76	0.00	234.76	151.21	136.11	85.31
MABSel_2	231.24	0.00	231.24	149.86	133.22	83.47
CBWS_341	5.34	217.47	222.81	144.85	130.05	81.89
CBWS_609	0.00	212.74	212.74	143.92	137.33	89.43
CBWS_custom_1	104.79	76.13	180.92	124.11	143.71	94.23
CBWS_55	0.00	175.99	175.99	131.21	116.66	86.14
ClusBasedRandom	0.00	23.12	23.12	23.01	7.05	7.01
InstBasedRandom	0.00	16.32	16.32	16.10	5.27	5.16
0.44%-to-0.96% total: 14 tasks	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Unique Hits Mean	Total Hits std	Total Unique Hits std
MABSel_exploitive	431.41	0.00	431.41	281.01	189.18	112.65
MABSel_2	429.49	0.00	429.49	278.64	193.98	115.65
CBWS_609	0.00	409.83	409.83	282.91	188.14	124.07
CBWS_341	10.04	398.74	408.78	271.12	185.24	113.26
CBWS_custom_1	194.33	178.32	372.65	261.13	182.34	125.34
CBWS_55	0.00	354.57	354.57	271.55	161.85	125.57
ClusBasedRandom	0.00	46.25	46.25	45.98	15.69	15.61
InstBasedRandom	0.00	31.37	31.37	31.00	9.19	9.02
1.06%-to-1.81% total: 14 tasks	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Unique Hits Mean	Total Hits std	Total Unique Hits std
MABSel_exploitive	733.69	0.00	733.69	444.17	318.82	161.83
MABSel_2	731.43	0.00	731.43	441.14	315.69	160.95
CBWS_609	0.00	705.03	705.03	451.04	287.16	160.89
CBWS_341	54.73	646.00	700.73	427.20	309.04	159.28
CBWS_custom_1	327.69	328.11	655.79	428.91	295.08	167.07
CBWS_55	0.00	592.30	592.30	444.26	233.39	164.34
ClusBasedRandom	0.00	87.78	87.78	87.22	21.43	21.22
InstBasedRandom	0.00	62.13	62.13	61.37	13.82	13.66

Table 5.15: Experiment 3.1 binned/grouped tasks single-point per strategy metric summaries at the end of 50 iterations. These metrics are aggregated over all task-plate runs per strategy (1020 runs per strategy). *(3 of 3)*

1.9%-to-3.37% total: 14 tasks	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Unique Hits Mean	Total Hits std	Total Unique Hits std
MABSel_exploitive	1217.51	0.00	1217.51	661.24	546.76	261.85
MABSel_2	1178.35	0.00	1178.35	646.71	499.61	243.70
CBWS_341	167.58	958.76	1126.34	635.19	474.03	256.65
CBWS_609	0.00	1076.51	1076.51	687.57	383.65	277.98
CBWS_custom_1	453.96	553.98	1007.94	652.83	437.57	282.59
CBWS_55	0.00	914.59	914.59	708.78	328.55	297.13
ClusBasedRandom	0.00	160.67	160.67	159.51	36.18	35.90
InstBasedRandom	0.00	124.75	124.75	122.85	25.03	24.45
3.79%-to-6.17% total: 4 tasks	**Exploit Hits Mean**	**Explore Hits Mean**	**Total Hits Mean**	**Total Unique Hits Mean**	**Total Hits std**	**Total Unique Hits std**
MABSel_exploitive	2624.00	0.00	2624.00	1051.85	409.95	76.44
CBWS_341	1345.75	1080.55	2426.30	1083.53	435.99	86.87
MABSel_2	2374.22	0.00	2374.22	991.60	269.90	91.96
CBWS_custom_1	712.48	1149.47	1861.95	1275.82	163.49	185.73
CBWS_609	0.00	1793.60	1793.60	1362.55	99.69	181.75
CBWS_55	0.00	1601.68	1601.68	1478.02	126.60	209.56
ClusBasedRandom	0.00	298.80	298.80	296.32	48.20	47.83
InstBasedRandom	0.00	233.70	233.70	228.42	44.32	42.17

A more direct perspective is to look at per task performance. per task comparisons involve looking at each task, taking the 10 initial plate runs for each strategy for that task, and comparing strategies accordingly to determine a winner for that task. In the end, we will have a winning strategy for each of the 102 tasks. To start off, let us look at the per task boxplots after 50 iterations for total hits and total unique hits in Figures 5.15 and 5.16, respectively. These 102 tasks are in ascending order of hit % which is used as an intuitive proxy for *hardness* of a task. In general, visually, most strategies outperform ClusterBasedRandom and InstanceBasedRandom except for arguably 10 tasks. It is assuring that even at low hit % tasks, the *smarter* strategies (non-random) do better than random.

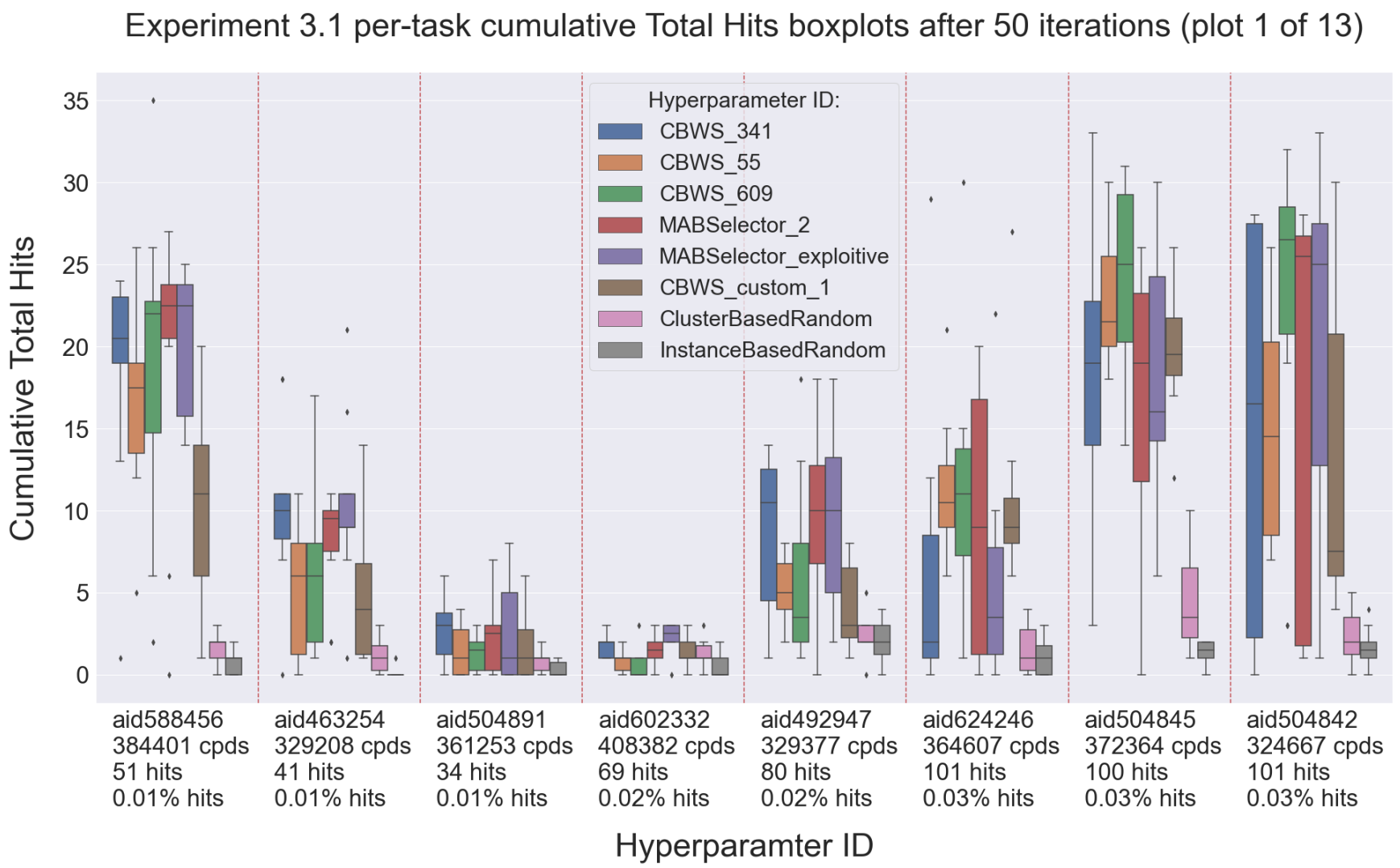


Figure 5.15: Experiment 3.1 per task **Total Hits** boxplots after 50 iterations. The x-tick labels for each task include number of compounds, number of hits, and hit %. This shows 8 of the 102 tasks; the remaining boxplots can be found in Appendix A.1.

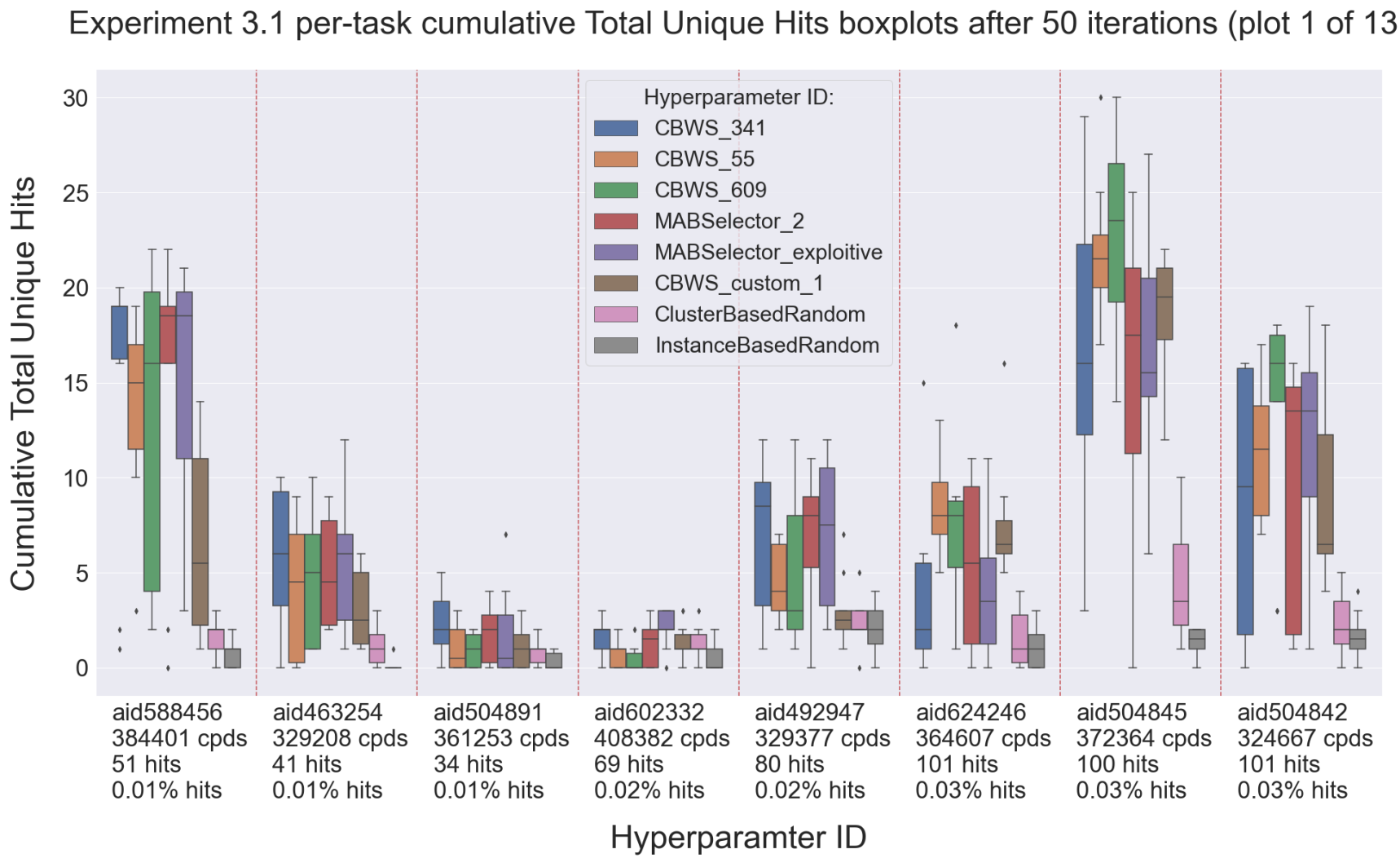


Figure 5.16: Experiment 3.1 per task **Total Unique Hits** boxplots after 50 iterations. The x-tick labels for each task include number of compounds, number of hits, and hit %. This shows 8 of the 102 tasks; the remaining boxplots can be found in Appendix A.2.

Now we need to further scrutinize strategies for each task. We employ contrast estimation based on medians (CEM) to compare strategies across their 10 runs for a specific task. As an example, Figures 5.17 and 5.18 plots a heatmap of the CEM matrix for 4 tasks after 50 iterations on total hits and total unique hits, respectively. The full 102 task CEM heatmaps and total wins plots can be found in Appendix B.1 and B.2. Notice that the CEM plots include the following extra columns:

- **Total Wins:** Number of times the row strategy had a positive CEM difference (i.e. outperformed another strategy).
- **# Failures:** Number of times a strategy's run for that task (total 10 runs) did no better than ClusterBasedRandom or InstanceBasedRandom.
- **#** $\geq 10\%$**:** Number of times a strategy's run for that task (total 10 runs) found at least 10% of the total target actives.
- **#** $\geq 25\%$**:** Number of times a strategy's run for that task (total 10 runs) found at least 25% of the total target actives.
- **#** $\geq 50\%$**:** Number of times a strategy's run for that task (total 10 runs) found at least 50% of the total target actives.

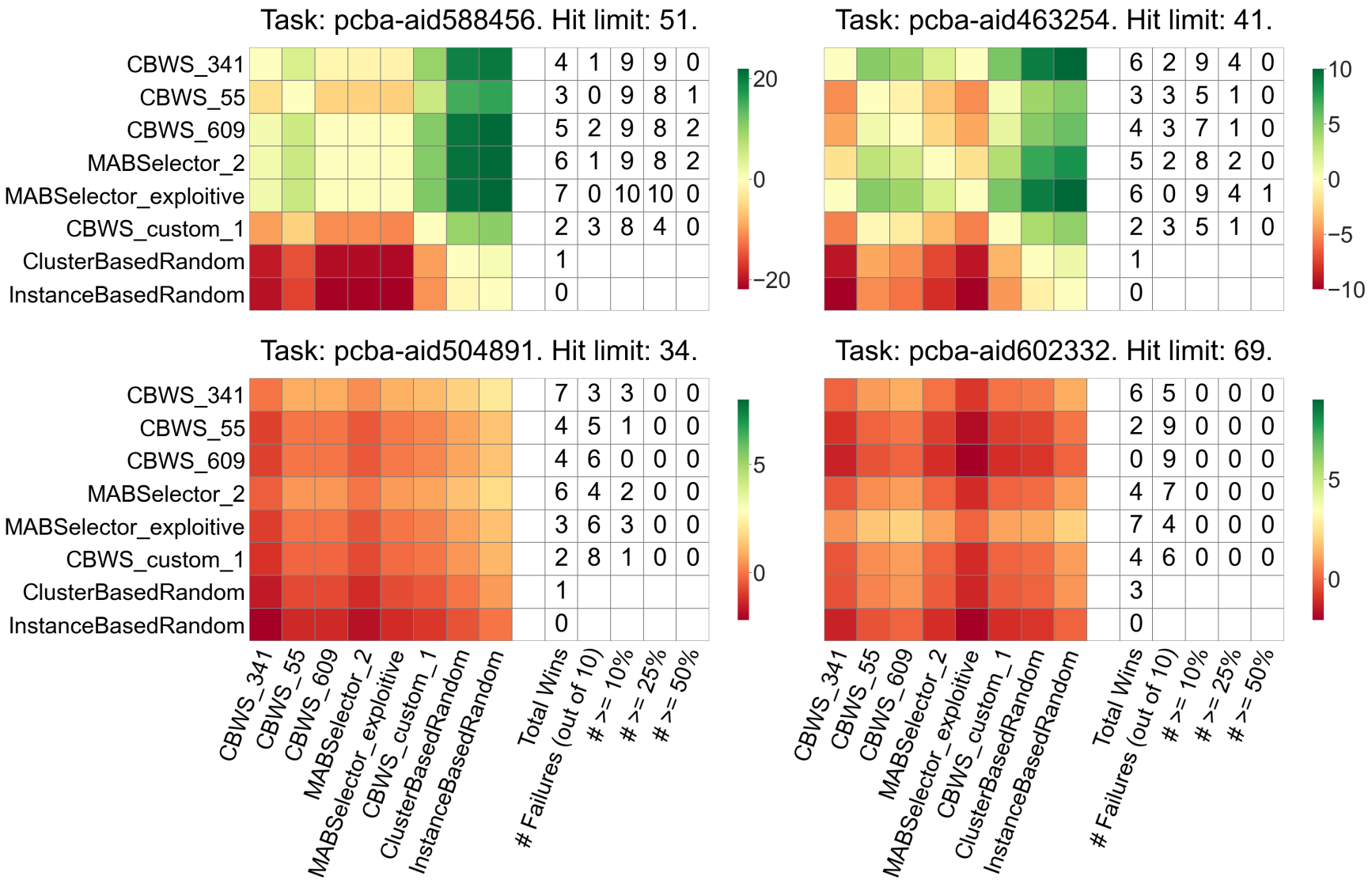


Figure 5.17: Experiment 3.1 per-task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. This is shows 4 of the 102 tasks, the remaining CEM heatmaps can be found in Appendix B.1.

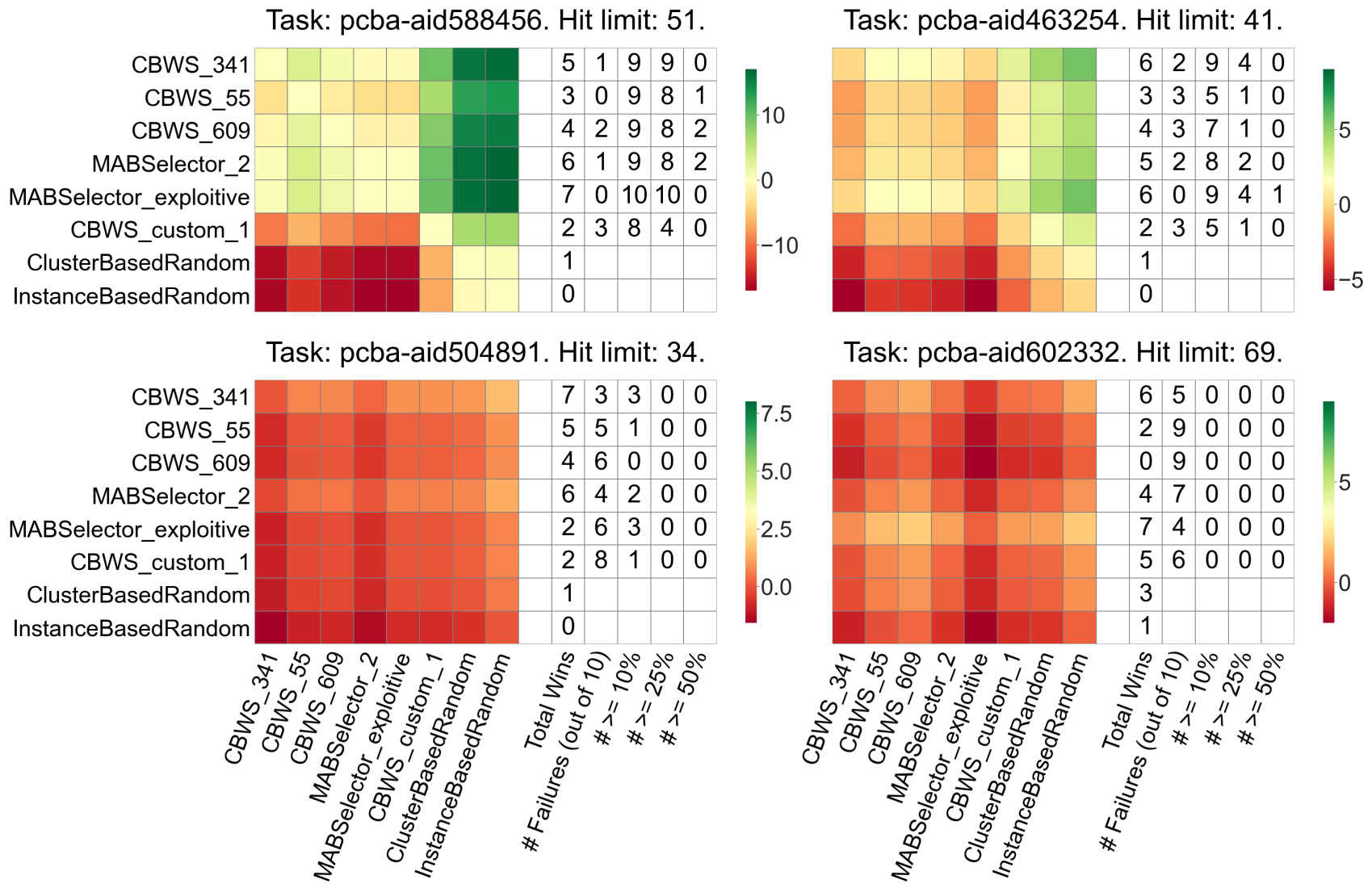


Figure 5.18: Experiment 3.1 per-task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. This is shows 4 of the 102 tasks, the remaining CEM heatmaps can be found in Appendix B.2.

The CEM matrix is a function of the task, metric, and iteration limit. We consider 6 metrics and 5 iteration limits. The 6 metrics we consider are total hits, total unique hits, # failures, # $\geq 10\%$, # $\geq 25\%$, and # $\geq 50\%$. The 5 iteration limits we consider are 10, 20, 30, 40, and 50 iterations. As an example, with regards to the total hits metric and iteration limit 50, then for all 102 tasks, we compute the CEM matrix and tally the row-wise positive strategy wins to determine the best strategy (with ties) per task. This gives 102 best strategies under the total hits metric and iteration limit 50, that we can then sum to count the number of times each strategy appeared as the best for each task. The strategy that appears the most number of times is designated as the best for this metric and iteration limit. Table 5.16 shows this for 2 considered metrics: total hits and total unique hits. The counts in each cell reflect the number of times the strategy (row) appeared as the best under that metric and iteration limit across all 102 tasks using the CEM's total wins as seen in Figure 5.17. As such, the maximum possible count for each cell is 102. Similarly, we want to avoid strategies that fail often (do worse than random) and prefer strategies that succeed often (find a minimum % of hits). Thus, Table 5.17 reflects this with 4 metrics: # failures, # $\geq 10\%$, # $\geq 25\%$, and # $\geq 50\%$. # failures denotes the number of 1020 runs that a strategy does not exceed ClusterBasedRandom or InstanceBasedRandom (the fewer failures the better). # $\geq x\%$ metrics denote the number of 1020 runs a strategy finds at least $x\%$ of hits for the task run (the more successes the better). Tables 5.16 and 5.17 give us a systematically derived compact view of the results.

We summarize the findings of experiment 3.1 as follows:

- Based on **total hits**, MABSelector_exploitive is the best overall from 10 to 50 iterations. Based on **total unique hits**, the best strategy from 10 to 50 iterations is CBWS_609. However, note that from 10 to 30 iterations CBWS_55 and CBWS_609 have similar counts at the top. Then as we get into 50 iterations, CBWS_55, CBWS_609, and MABSelector_exploitive have similar counts.

- Based on **# failures**, all strategies exhibit similar counts after 30 iterations except for CBWS custom_1. After 50 iterations, *smart* strategies (non-random) fail to surpass random strategies in less than 100 runs out 1020. These failure runs could be related to a particular task, and

so we can inspect Table 5.18 that shows the failures after 50 iterations. We restrict our view to only tasks that have non-zero failures which is 37 tasks; the remaining 65 tasks have no failures for any of the strategies. It is evident that certain tasks exhibit higher fail rates across all strategies, as can be seen for task IDs: 504891, 602332, 624291, and 743266 that fail in more than half their runs (10 runs per task per strategy).

- Based on **#** $\geq x\%$, it is biased towards highlighting strategies that have high total hits. Thus we can see that exploitive heavy strategies shine. At 50 iterations we can see that MABSelector methods outperform others but CBWS_609 is not far off. Furthermore, at 50 iterations many *smart* strategies (non-random) find more than 10% of the task hits in more than 800 (out of 1020) runs. However, 2 strategies are able to find more than 50% of task hits in only 10 out of 1020 runs. Upon close inspection, the counts for **#** $\geq 50\%$ involve only 4 tasks as seen in Table 5.19 with a significant majority from task ID 2242, indicating that these successes are task specific.

Taking these criteria and results under consideration, we are left with three strategies: CBWS_55, CBWS_609, and MABSelector_exploitive. Both CBWS_55 and CBWS_609 are exploration heavy, and objectively, CBWS_609 outperforms CBWS_55 on total unique hits. So with this we are left with two strategies: CBWS_609 and MABSelector_exploitive. Thus, just based on experiment 3.1 conditions and conducted analysis, one would select MABSelector_exploitive for maximizing total hits, and CBWS_609 for maximizing total unique hits (i.e. novelty).

Table 5.16: Experiment 3.1 top 1 CEM counts for each strategy under a metric and iteration limit setting. The counts in each cell reflect the number of times the strategy (row) appeared as the best under that metric and iteration limit across all 102 tasks using the CEM's total wins as seen in Figure 5.17. As such, the maximum value possible for each cell is 102.

	Top 1 CEM Counts for Total Hits				
# iterations:	10	20	30	40	50
CBWS_341	25	18	14	17	16
CBWS_55	5	3	2	2	2
CBWS_609	15	18	19	15	12
MABSel_2	14	18	21	23	22
MABSel_exploit	39	41	43	44	50
CBWS_custom_1	4	5	3	2	1
ClusBasedRandom	1	1	1	0	1
InstBasedRandom	0	0	0	0	0
Best	MABSel_expl	MABSel_expl	MABSel_expl	MABSel_expl	MABSel_expl
	Top 1 CEM Counts for Total Unique Hits				
# iterations:	10	20	30	40	50
CBWS_341	15	14	9	13	13
CBWS_55	30	29	28	23	22
CBWS_609	32	32	28	28	24
MABSel_2	11	8	14	15	16
MABSel_exploit	14	16	19	21	23
CBWS_custom_1	8	3	4	3	4
ClusBasedRandom	2	1	1	0	1
InstBasedRandom	1	0	0	0	0
Best	CBWS_609	CBWS_609	CBWS_55 CBWS_609	CBWS_609	CBWS_609

Table 5.17: Experiment 3.1 failure and success counts for each strategy over all task-plate runs (102 tasks and 10 initial starting plates = 1020 runs). # failures denotes the number of runs that a strategy does not exceed ClusterBasedRandom or InstanceBasedRandom in performance based on total hits or total unique hits. # $\geq x\%$ metrics denote the number of runs a strategy finds at least $x\%$ of hits for the task run.

	# Failures for all 1020 task-plate runs				
# iterations:	10	20	30	40	50
CBWS_341	164	106	92	73	61
CBWS_55	190	121	92	59	48
CBWS_609	181	116	82	67	49
MABSelector_2	159	111	89	68	66
MABSelector_exploitive	171	104	81	62	50
CBWS_custom_1	242	173	129	104	87
	# $\geq$ 10% hits for all 1020 task-plate runs				
# iterations:	10	20	30	40	50
CBWS_341	106	347	585	715	806
CBWS_55	55	277	436	575	697
CBWS_609	81	367	553	671	803
MABSelector_2	117	348	577	716	813
MABSelector_exploitive	126	361	592	726	810
CBWS_custom_1	40	215	417	547	682
	# $\geq$ 25% hits for all 1020 task-plate runs				
# iterations:	10	20	30	40	50
CBWS_341	2	28	70	180	254
CBWS_55	2	8	27	59	126
CBWS_609	2	28	76	159	243
MABSelector_2	1	31	90	180	257
MABSelector_exploitive	5	35	108	201	280
CBWS_custom_1	1	11	31	100	182
	# $\geq$ 50% hits for all 1020 task-plate runs				
# iterations:	10	20	30	40	50
CBWS_341	0	0	0	0	5
CBWS_55	0	0	0	0	1
CBWS_609	0	1	1	1	6
MABSelector_2	0	0	0	3	10
MABSelector_exploitive	0	0	0	3	10
CBWS_custom_1	0	0	0	0	5

Table 5.18: Experiment 3.1 per task # failures for each strategy after 50 iterations (10 runs per task). # failures denotes the number of runs that a strategy does not exceed ClusterBasedRandom or InstanceBasedRandom based on total hits or total unique hits.

Task ID	CBWS_341	CBWS_55	CBWS_609	MABSel_2	MABSel expl	CBWS cstm_1	Task Total
588456	1	0	2	1	0	3	7
463254	2	3	3	2	0	3	13
504891	3	5	6	4	6	8	32
602332	5	9	9	7	4	6	40
492947	3	3	5	2	3	7	23
624246	6	0	1	3	3	0	13
504845	1	0	0	3	0	0	4
504842	4	0	1	3	1	1	10
2662	1	0	0	0	2	1	4
602233	0	1	0	1	1	2	5
2675	6	2	0	7	5	0	20
485294	2	1	0	3	3	1	10
1469	0	0	0	0	0	2	2
1634	0	1	0	0	1	1	3
652025	0	0	0	1	0	2	3
624291	7	5	2	8	5	2	29
720707	2	0	0	1	1	0	4
504706	0	2	2	3	1	2	10
720711	0	0	0	1	1	1	3
743266	8	6	9	7	9	8	47
485281	0	0	0	1	1	0	2
602310	0	0	0	0	1	0	1
2101	0	1	1	0	0	0	2
602179	0	1	2	0	0	2	5
1452	0	0	0	1	0	1	2
1471	5	1	1	4	2	0	13
652106	3	2	2	1	0	5	13
624287	0	0	1	0	0	0	1
485353	0	1	1	0	0	4	6
485349	0	0	0	2	0	3	5
1631	0	0	0	0	0	2	2
720551	2	4	1	0	0	8	15
2326	0	0	0	0	0	8	8
624288	0	0	0	0	0	1	1
485341	0	0	0	0	0	3	3
Strategy Total	61	48	49	66	50	87	-

Table 5.19: Experiment 3.1 per task $\# \geq 50\%$ counts each strategy after 50 iterations (10 runs per task). Counts denote the number of runs a strategy finds at least 50% of hits for the task run.

Task ID	CBWS_341	CBWS_55	CBWS_609	MABSel_2	MABSel expl	CBWS cstm_1	Task Total
588456	0	1	2	2	0	0	5
463254	0	0	0	0	1	0	1
2242	5	0	4	8	8	5	30
881	0	0	0	0	1	0	1
Strategy Total	5	1	6	10	10	5	-

Experiment 3.1: CBWS_609 vs MABSelector_exploitive

From experiment 3.1 analysis, we concluded that CBWS_609 was the best for maximizing total unique hits, and MABSelector_exploitive for maximizing total hits. Here we compare the two strategies' compound selections by looking at the overlap/intersection of their hits. Table 5.20 shows the per task mean union and symmetric difference between the hits found by CBWS_609 and MABSelector_exploitive after 50 iterations. The means are aggregated over the 10 initial plate runs for each task. The mean disagreement ratio is the ratio between the mean union and the mean symmetric difference columns. This value is used as a proxy to quantify the difference between the two strategies in terms of their hit selection. A higher mean disagreement ratio indicates that the two strategies differ more in their hit selection. 65 of the 102 tasks have a mean disagreement ratio greater than 0.5, which means that the two strategies have more than 50% exclusive hits. Due to the large exclusive compound selection, future work should consider a strategy that tries to merge the properties of these two strategies. Another option is to perhaps run the exploration strategy for the early iterations, and then the exploitation strategy for the later iterations.

Table 5.20: Experiment 3.1 per task mean hit compound union and symmetric difference between CBWS_609 and MABSelector_exploit after 50 iterations. These are two set mean counts aggregated over the 10 runs per task.

Task ID	CBWS_609 Mean Total Hits	MAB_exploit Mean Total Hits	Union Mean Total Hits	Symmetric Diff. Mean Total Hits	Mean Disagreement Ratio	Hit %
588456	18.9	20.3	24.8	10.4	0.42	0.01
463254	6.2	10.3	13.5	10.5	0.78	0.01
504891	1.3	2.4	3.0	2.3	0.77	0.01
602332	0.6	2.3	2.6	2.3	0.88	0.02
492947	6.1	9.8	13.0	10.1	0.78	0.02
624246	11.7	5.8	14.0	10.5	0.75	0.03
504845	24.2	18.2	35.7	29.0	0.81	0.03
504842	22.2	20.7	28.4	13.9	0.49	0.03
2662	19.1	15.4	26.9	19.3	0.72	0.04
602233	32.1	37.4	47.9	26.3	0.55	0.04
2675	21.0	10.0	23.3	15.6	0.67	0.04
485294	19.0	20.1	33.8	28.5	0.84	0.05
1469	30.9	39.8	53.6	36.5	0.68	0.06
1634	23.1	18.0	33.8	26.5	0.78	0.06
652025	20.8	28.0	39.3	29.8	0.76	0.07
624291	6.0	3.4	9.3	9.2	0.99	0.07
720707	37.8	41.0	61.6	44.4	0.72	0.07
504706	13.8	15.4	23.5	17.8	0.76	0.07
720711	34.8	46.3	66.3	51.5	0.78	0.08
743266	3.0	2.9	5.7	5.5	0.96	0.08
485281	36.9	25.0	51.2	40.5	0.79	0.08
602310	64.2	64.4	89.3	50.0	0.56	0.08
2101	47.7	63.6	72.9	34.5	0.47	0.09
602179	16.9	23.4	33.7	27.1	0.80	0.09
1452	60.0	60.7	76.9	33.1	0.43	0.12
1471	16.0	15.8	29.7	27.6	0.93	0.13
652106	30.9	38.0	58.9	48.9	0.83	0.14
624287	35.0	48.2	71.5	59.8	0.84	0.14
720709	104.2	114.3	155.6	92.7	0.60	0.15
485367	96.2	97.6	138.5	83.2	0.60	0.17

Continued on next page

Table 5.20: Experiment 3.1 per task mean hit compound union and symmetric difference between CBWS_609 and MABSelector_exploit after 50 iterations. These are two set mean counts aggregated over the 10 runs per task.

Task ID	CBWS_609 Mean Total Hits	MAB_exploit Mean Total Hits	Union Mean Total Hits	Symmetric Diff. Mean Total Hits	Mean Disagreement Ratio	Hit %
720708	78.8	76.6	111.9	68.4	0.61	0.18
485353	101.5	115.6	138.4	59.7	0.43	0.19
485349	40.7	36.0	68.4	60.1	0.88	0.19
2528	163.7	163.4	215.1	103.1	0.48	0.19
602313	133.9	134.9	194.0	119.2	0.61	0.20
651644	179.5	190.9	242.1	113.8	0.47	0.21
720542	210.6	229.3	267.1	94.3	0.35	0.21
504327	147.4	142.9	208.3	126.3	0.61	0.21
624170	131.7	141.2	192.1	111.3	0.58	0.21
743255	283.6	303.5	367.4	147.7	0.40	0.24
1379	145.9	182.9	228.2	127.6	0.56	0.28
485290	137.1	152.7	195.1	100.4	0.51	0.28
1479	93.8	115.9	137.8	65.9	0.48	0.29
2676	120.5	146.9	203.4	139.4	0.69	0.30
624171	255.2	268.5	345.4	167.1	0.48	0.31
1631	100.0	147.9	189.4	130.9	0.69	0.34
2517	323.0	371.1	428.9	163.7	0.38	0.34
588795	476.8	492.8	575.1	180.6	0.31	0.35
1457	147.5	145.8	216.8	140.3	0.65	0.35
720551	49.8	83.6	114.3	95.2	0.83	0.37
1721	341.4	332.1	416.5	159.5	0.38	0.37
2242	346.2	363.7	414.9	119.9	0.29	0.39
2100	359.4	381.4	436.7	132.6	0.30	0.39
2326	64.6	93.6	130.3	102.4	0.79	0.41
1468	239.5	277.9	374.8	232.2	0.62	0.41
624288	60.6	65.4	111.0	96.0	0.86	0.42
1454	188.4	201.8	239.3	88.4	0.37	0.44
651768	499.5	529.6	637.9	246.7	0.39	0.47
720580	264.3	271.4	362.3	188.9	0.52	0.49
588579	657.2	703.5	819.5	278.3	0.34	0.51

Continued on next page

Table 5.20: Experiment 3.1 per task mean hit compound union and symmetric difference between CBWS_609 and MABSelector_exploit after 50 iterations. These are two set mean counts aggregated over the 10 runs per task.

Task ID	CBWS_609 Mean Total Hits	MAB_exploit Mean Total Hits	Union Mean Total Hits	Symmetric Diff. Mean Total Hits	Mean Disagreement Ratio	Hit %
2549	339.7	369.9	436.6	163.6	0.37	0.52
485341	83.2	100.4	160.5	137.4	0.86	0.53
881	267.9	277.9	326.6	107.4	0.33	0.57
540317	539.9	540.6	685.0	289.5	0.42	0.58
720579	428.0	453.4	544.1	206.8	0.38	0.67
485360	444.4	475.7	564.7	209.3	0.37	0.68
2451	423.3	409.5	548.8	264.8	0.48	0.73
504847	838.7	829.1	1074.1	480.4	0.45	0.92
924	323.0	360.0	450.9	218.8	0.49	0.95
720553	440.1	516.9	635.1	313.2	0.49	0.96
588453	787.7	791.3	1051.0	523.0	0.50	1.06
624202	253.2	265.7	439.4	359.9	0.82	1.08
651635	617.3	599.3	904.6	592.6	0.66	1.09
588590	899.2	955.2	1117.0	379.6	0.34	1.10
1461	233.6	246.8	395.2	310.0	0.78	1.11
1688	396.0	381.9	517.1	256.3	0.50	1.16
588591	1116.7	1228.8	1484.0	622.5	0.42	1.26
652105	1107.1	1201.2	1414.4	520.5	0.37	1.26
504466	822.6	869.3	1085.4	478.9	0.44	1.34
588855	649.9	635.5	956.3	627.2	0.66	1.39
485314	944.6	988.2	1201.2	469.6	0.39	1.42
902	387.0	390.5	545.7	313.9	0.58	1.57
686970	727.3	723.5	1090.3	729.8	0.67	1.76
2147	928.2	994.4	1147.1	371.6	0.32	1.81
652104	454.4	432.1	721.4	556.3	0.77	1.90
651965	616.1	639.3	952.6	649.8	0.68	1.96
624417	747.7	747.4	1070.7	646.3	0.60	1.96
624297	796.1	783.4	1158.4	737.3	0.64	2.02
540276	793.0	845.4	1139.2	640.0	0.56	2.23
485313	1436.1	1673.7	2090.0	1070.2	0.51	2.43

Continued on next page

Table 5.20: Experiment 3.1 per task mean hit compound union and symmetric difference between CBWS_609 and MABSelector_exploit after 50 iterations. These are two set mean counts aggregated over the 10 runs per task.

Task ID	CBWS_609 Mean Total Hits	MAB_exploit Mean Total Hits	Union Mean Total Hits	Symmetric Diff. Mean Total Hits	Mean Disagreement Ratio	Hit %
504444	769.8	838.4	1135.0	661.8	0.58	2.54
1460	1793.5	2385.2	2799.8	1420.9	0.51	2.54
720504	1224.7	1352.1	1728.8	880.8	0.51	2.90
485297	1516.3	1868.1	2323.0	1261.6	0.54	2.94
1458	1292.2	1416.8	1711.0	713.0	0.42	2.97
485364	1469.0	1843.0	2402.4	1492.8	0.62	3.13
504467	1076.9	1115.8	1478.5	764.3	0.52	3.14
624296	1085.3	1104.5	1624.3	1058.8	0.65	3.37
2546	1747.0	2388.7	2954.5	1773.3	0.60	3.79
504339	1727.7	2468.4	3465.7	2735.3	0.79	4.74
504333	1741.8	2354.5	3233.8	2371.3	0.73	4.81
2551	1957.9	3284.4	4365.7	3489.1	0.80	6.17

Experiment 3.2: 107 Task Datasets, initial plate with no actives, 96-plate batch size, and 50 iterations

In experiment 3.2, there was an interest to look at performance when the initial plate has no actives; which is to be expected in drug screening at the start of a campaign. When running any of the 8 strategies considered in experiment 3 with an initial plate of inactives, then strategies that make use of a machine learning model will not benefit from the model's predictions since there is only a single class in the training set (i.e. all training examples are inactive). In experiment 3.2, for each of the 107 task, independently, we randomly sampled a 96-compound plate of all inactives. Thus, for each task, we ran each strategy on the task's initial all-inactives plate for 50 iterations.

The results here have two dimensions: task and strategy. Thus, we group by strategy and aggregate across the tasks. Table 5.21 summarizes the single-point aggregates for total hits and total unique hits. In contrast to experiment 3.1, MABSelector_exploitive is not among the top

performers in iterations 10, 20, 30, and 40. With no actives in the initial plate, it appears that exploration heavy strategies are key in the early iterations. CBWS_341 is particularly interesting as it assigns roughly 75% of its budget to exploration, and consequently, dominates total hits across iterations. CBWS_609 and CBWS_55 are still the best for total unique hits across iterations, with CBWS_609 still doing well on total hits at 50 iterations. We can also look at the CEM heatmaps across iterations for total hits and total unique hits in Figures 5.19 and 5.20, respectively. These CEM heatmaps are computed by using the task as the block factor (row) and strategies as algorithms (columns) for iterations 10, 20, 30, 40, and 50 under a specific metric (one run per task). This is in contrast to experiment 3.1, whereby we generated a CEM heatmap for each task and blocking on the initial plate for that task. The CEM heatmaps tell a similar story to Table 5.21, except that in iteration 50, CBWS_609 is deemed the best on both metrics.

Table 5.21: Experiment 3.2 single-point per strategy metric summaries at the end of 10, 20, 30, 40, and 50 iterations. These metrics are aggregated over all 102 tasks which have one run (one plate of 96 inactive compounds). *(1 of 2 cont.)*

# iterations: 10	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Unique Hits Mean	Total Hits std	Total Unique Hits std
CBWS_341	17.84	69.65	87.49	54.05	123.67	65.95
CBWS_609	0.00	76.31	76.31	61.40	95.29	78.23
CBWS_55	0.00	69.74	69.74	61.32	93.49	86.62
CBWS_custom_1	17.00	31.48	48.48	39.70	78.04	61.03
MABSel_exploitive	41.58	0.00	41.58	21.32	81.19	39.09
MABSel_2	38.38	0.00	38.38	19.69	81.24	38.48
ClustBasedRandom	0.00	12.62	12.62	12.59	15.87	15.85
InstBasedRandom	0.00	8.25	8.25	8.23	10.75	10.72
# iterations: 20	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Unique Hits Mean	Total Hits std	Total Unique Hits std
CBWS_341	41.98	156.91	198.89	115.63	262.11	129.45
CBWS_609	0.00	179.12	179.12	131.38	210.41	159.08
CBWS_55	0.00	157.69	157.69	130.69	188.53	168.40
CBWS_custom_1	59.99	85.80	145.79	107.02	200.14	147.07
MABSel_exploitive	143.18	0.00	143.18	72.88	237.67	104.29
MABSel_2	132.18	0.00	132.18	68.96	219.54	99.51
ClustBasedRandom	0.00	24.40	24.40	24.34	30.05	29.95
InstBasedRandom	0.00	17.36	17.36	17.28	22.82	22.69
# iterations: 30	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Unique Hits Mean	Total Hits std	Total Unique Hits std
CBWS_341	61.39	231.20	292.59	169.04	384.40	188.82
CBWS_609	0.00	272.87	272.87	192.92	312.54	229.88
MABSel_exploitive	245.18	0.00	245.18	128.80	371.87	165.34
CBWS_55	0.00	237.26	237.26	193.07	278.96	247.20
CBWS_custom_1	103.38	132.42	235.80	164.89	312.21	218.59
MABSel_2	234.85	0.00	234.85	121.60	345.61	152.21
ClustBasedRandom	0.00	36.27	36.27	36.14	44.19	43.91
InstBasedRandom	0.00	26.25	26.25	26.08	34.75	34.44

Table 5.21: Experiment 3.2 single-point per strategy metric summaries at the end of 10, 20, 30, 40, and 50 iterations. These metrics are aggregated over all 102 tasks which have one run (one plate of 96 inactive compounds). *(2 of 2)*

# iterations: 40	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Unique Hits Mean	Total Hits std	Total Unique Hits std
CBWS_341	75.30	305.37	380.68	218.72	495.02	242.51
CBWS_609	0.00	350.29	350.29	242.17	404.17	290.20
MABSel_exploitive	342.72	0.00	342.72	182.23	506.03	222.51
MABSel_2	331.62	0.00	331.62	174.03	467.81	205.27
CBWS_custom_1	143.44	172.41	315.85	215.91	411.73	279.37
CBWS_55	0.00	300.83	300.83	241.62	356.72	311.66
ClustBasedRandom	0.00	47.82	47.82	47.59	59.12	58.67
InstBasedRandom	0.00	34.80	34.80	34.48	46.32	45.71
# iterations: 50	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Unique Hits Mean	Total Hits std	Total Unique Hits std
CBWS_341	83.99	373.35	457.34	263.44	594.19	294.49
MABSel_exploitive	428.92	0.00	428.92	229.74	621.99	274.66
CBWS_609	0.00	419.84	419.84	285.49	491.84	342.62
MABSel_2	415.98	0.00	415.98	222.51	574.93	260.64
CBWS_custom_1	181.61	208.73	390.33	263.40	501.74	333.60
CBWS_55	0.00	359.62	359.62	284.61	427.21	364.69
ClustBasedRandom	0.00	59.51	59.51	59.13	73.46	72.80
InstBasedRandom	0.00	43.56	43.56	42.89	58.64	57.33

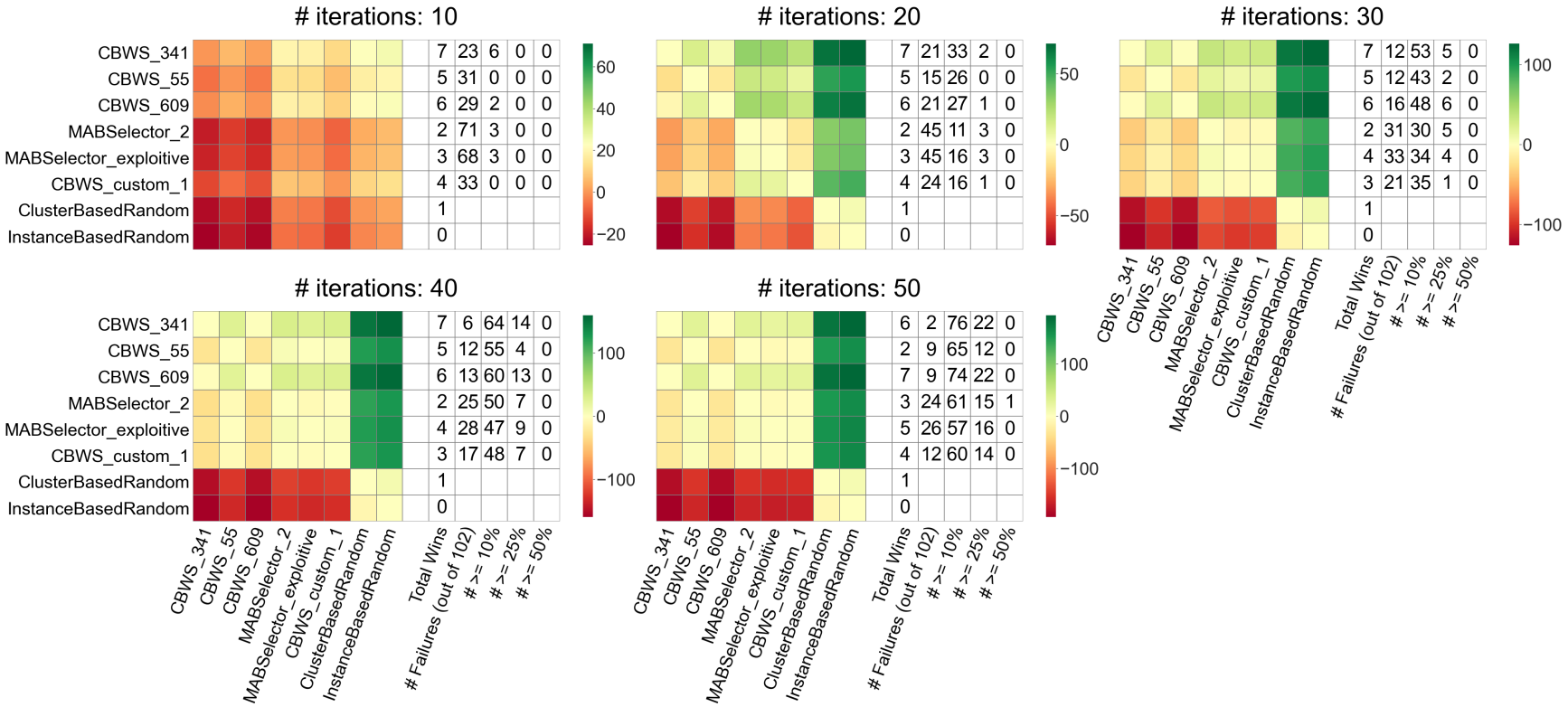


Figure 5.19: Experiment 3.2 contrast estimation based on medians (CEM) heatmaps for **Total Hits** at 10, 20, 30, 40, and 50 iterations. These CEMs are aggregated across all 102 tasks for the single run in Experiment 3.2.

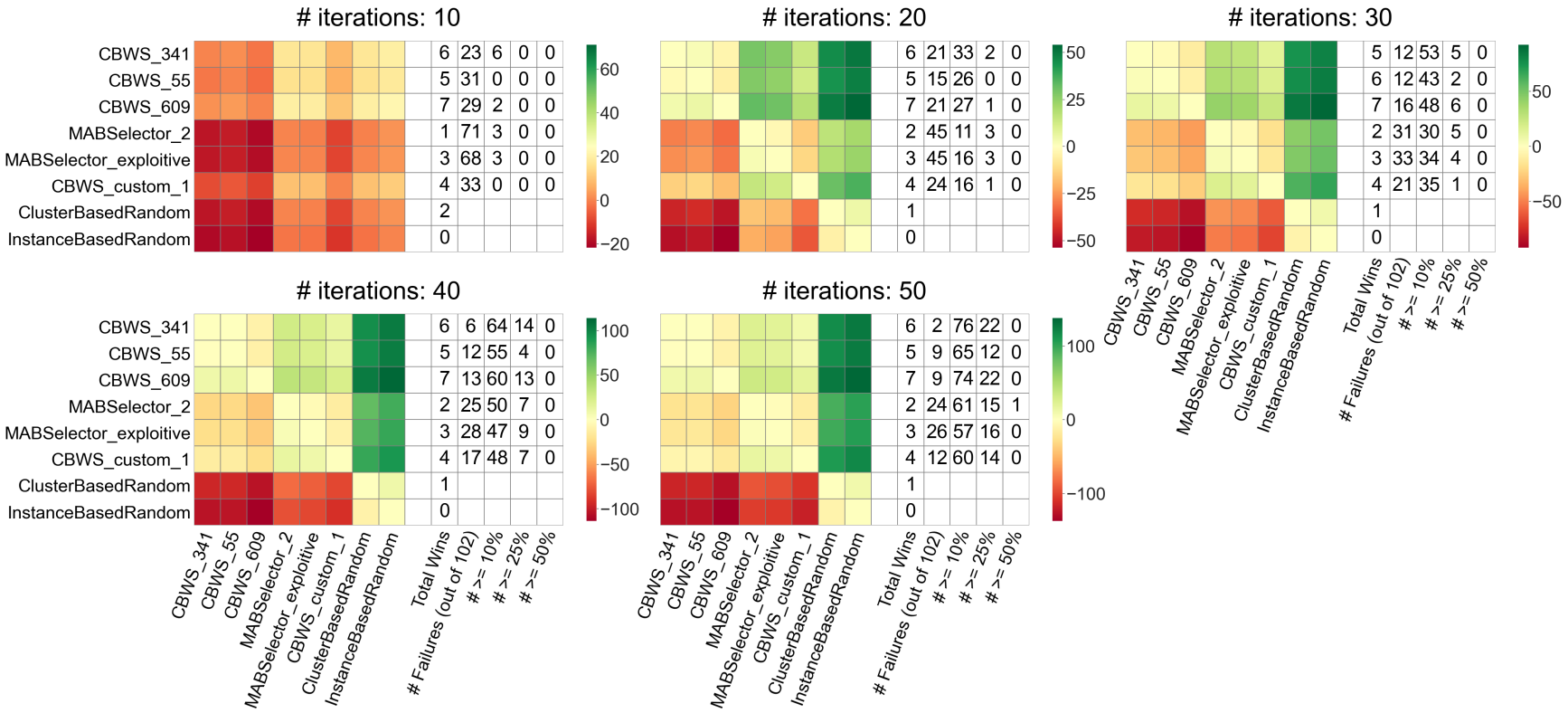


Figure 5.20: Experiment 3.2 contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** at 10, 20, 30, 40, and 50 iterations. These CEMs are aggregated across all 102 tasks for the single run in Experiment 3.2.

In Table 5.22, we divide the tasks into eight bins based on their hit % and aggregate results under each bin at 50 iterations. This gives us a better look at the effect of the *hardness* of a task, as we intuitively expect lower hit % tasks to be dominated by exploration heavy models under this no-actives initial plate setting. For smaller hit % (first bin), we see that exploration heavy strategies: CBWS_55, CBWS_341, and CBWS_609 perform similarly and are the top. Whereas MABSelector_exploitive suffers significantly, doing worse than ClusterBasedRandom. In the second and third bins, CBWS_341 clearly outperforms other strategies on total hits and total unique hits, with CBWS_609 trailing close on total unique hits. In bins four to six, CBWS_341 and CBWS_609 are among the top 2 performers on both metrics. In the seventh bin, CBWS_55 has the highest total unique hits, but falls behind on total hits. Here we also start to see MABSelector_exploitive retain its top rankings on total hits. In essence, we see that lower hit % tasks are dominated by exploration heavy strategies in a no-active plate setting. Furthermore, seeing CBWS_341 maintain many of the bins top rankings indicates that a exploration heavy, quarter-budget exploitation approach is certainly an avenue worthy of future investigation.

Table 5.22: Experiment 3.2 binned/grouped tasks single-point per strategy metric summaries after 50 iterations. These metrics are aggregated over all 102 tasks which have one run (one plate of 96 inactive compounds). *(1 of 3 cont.)*

0.01%-to-0.06% hits total: 14 tasks	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Unique Hits Mean	Total Hits std	Total Unique Hits std
CBWS_55	0.00	12.86	12.86	10.71	11.16	9.28
CBWS_341	0.00	12.64	12.64	11.07	10.90	9.39
CBWS_609	0.00	11.36	11.36	9.07	11.49	9.53
CBWS_custom_1	4.43	3.71	8.14	6.93	8.03	7.08
MABSel_2	3.07	0.00	3.07	2.07	5.01	3.17
ClustBasedRandom	0.00	2.57	2.57	2.57	2.14	2.14
MABSel_exploitive	2.29	0.00	2.29	1.64	5.30	3.05
InstBasedRandom	0.00	1.79	1.79	1.79	1.63	1.63
0.07%-to-0.14% hits total: 14 tasks	**Exploit Hits Mean**	**Explore Hits Mean**	**Total Hits Mean**	**Total Unique Hits Mean**	**Total Hits std**	**Total Unique Hits std**
CBWS_341	0.00	28.43	28.43	21.36	23.99	14.90
CBWS_custom_1	12.79	8.86	21.64	16.86	16.65	11.67
CBWS_609	0.00	20.93	20.93	17.64	17.98	13.21
MABSel_2	19.21	0.00	19.21	13.64	25.76	18.59
CBWS_55	0.00	18.00	18.00	15.29	12.55	9.82
MABSel_exploitive	16.43	0.00	16.43	12.57	21.91	17.31
ClustBasedRandom	0.00	7.36	7.36	7.36	3.95	3.95
InstBasedRandom	0.00	4.57	4.57	4.50	2.53	2.59
0.15%-to-0.28% hits total: 14 tasks	**Exploit Hits Mean**	**Explore Hits Mean**	**Total Hits Mean**	**Total Unique Hits Mean**	**Total Hits std**	**Total Unique Hits std**
CBWS_341	3.79	137.29	141.07	99.79	64.82	40.39
CBWS_609	0.00	132.57	132.57	97.93	68.48	47.37
CBWS_55	0.00	114.50	114.50	90.14	59.99	44.06
CBWS_custom_1	59.43	48.93	108.36	80.86	61.32	42.00
MABSel_exploitive	82.21	0.00	82.21	55.86	62.53	42.67
MABSel_2	75.14	0.00	75.14	50.50	63.37	41.76
ClustBasedRandom	0.00	17.50	17.50	17.36	5.33	5.21
InstBasedRandom	0.00	10.50	10.50	10.36	3.44	3.30

Table 5.22: Experiment 3.2 binned/grouped tasks single-point per strategy metric summaries after 50 iterations. These metrics are aggregated over all 102 tasks which have one run (one plate of 96 inactive compounds). *(2 of 3 cont.)*

0.29%-to-0.42% hits total: 14 tasks	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Unique Hits Mean	Total Hits std	Total Unique Hits std
CBWS_609	0.00	214.79	214.79	142.50	140.85	92.52
CBWS_341	4.00	201.43	205.43	136.64	127.31	84.31
MABSel_exploitive	190.00	0.00	190.00	117.86	149.91	88.73
MABSel_2	182.79	0.00	182.79	112.79	150.59	89.97
CBWS_custom_1	100.50	70.93	171.43	116.43	155.93	102.65
CBWS_55	0.00	170.07	170.07	126.71	125.07	90.62
ClustBasedRandom	0.00	25.07	25.07	24.86	9.87	9.66
InstBasedRandom	0.00	16.43	16.43	16.43	4.83	4.83
0.44%-to-0.96% hits total: 14 tasks	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Unique Hits Mean	Total Hits std	Total Unique Hits std
CBWS_341	10.64	412.14	422.79	279.93	193.30	120.72
CBWS_609	0.00	416.00	416.00	286.79	189.52	123.25
CBWS_custom_1	189.36	175.14	364.50	255.57	188.70	128.52
MABSel_2	362.64	0.00	362.64	237.21	146.44	86.06
MABSel_exploitive	358.00	0.00	358.00	234.64	147.48	87.19
CBWS_55	0.00	356.86	356.86	271.71	169.95	130.62
ClustBasedRandom	0.00	48.07	48.07	47.93	16.09	16.02
InstBasedRandom	0.00	30.00	30.00	29.43	10.98	10.73
1.06%-to-1.81% hits total: 14 tasks	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Unique Hits Mean	Total Hits std	Total Unique Hits std
CBWS_609	0.00	703.43	703.43	456.29	292.37	162.99
CBWS_341	53.79	643.57	697.36	427.00	322.42	163.75
MABSel_exploitive	653.14	0.00	653.14	389.57	311.63	165.25
MABSel_2	645.00	0.00	645.00	382.57	310.20	162.11
CBWS_custom_1	315.64	321.79	637.43	419.36	315.96	180.33
CBWS_55	0.00	594.07	594.07	442.93	242.31	172.28
ClustBasedRandom	0.00	87.57	87.57	87.36	18.22	18.11
InstBasedRandom	0.00	62.14	62.14	61.57	15.89	15.49

Table 5.22: Experiment 3.2 binned/grouped tasks single-point per strategy metric summaries after 50 iterations. These metrics are aggregated over all 102 tasks which have one run (one plate of 96 inactive compounds). *(3 of 3)*

1.90%-to-3.37% hits total: 14 tasks	Exploit Hits Mean	Explore Hits Mean	Total Hits Mean	Total Unique Hits Mean	Total Hits std	Total Unique Hits std
CBWS_341	169.21	976.93	1146.14	640.36	484.83	258.93
MABSel_exploitive	1071.14	0.00	1071.14	589.07	489.96	238.06
MABSel_2	1067.93	0.00	1067.93	572.29	439.48	214.30
CBWS_609	0.00	1055.57	1055.57	687.64	408.45	289.76
CBWS_custom_1	438.64	562.64	1001.29	655.93	459.17	290.93
CBWS_55	0.00	901.86	901.86	694.64	335.26	305.20
ClustBasedRandom	0.00	159.36	159.36	158.14	38.33	37.95
InstBasedRandom	0.00	121.64	121.64	120.00	24.78	24.08
3.79%-to-6.17% hits total: 4 tasks	**Exploit Hits Mean**	**Explore Hits Mean**	**Total Hits Mean**	**Total Unique Hits Mean**	**Total Hits std**	**Total Unique Hits std**
MABSel_exploitive	2631.25	0.00	2631.25	954.00	482.00	125.81
CBWS_341	1296.75	1077.00	2373.75	1061.25	484.12	46.96
MABSel_2	2362.25	0.00	2362.25	875.25	211.31	116.88
CBWS_custom_1	708.25	1150.50	1858.75	1285.00	189.66	204.85
CBWS_609	0.00	1764.75	1764.75	1337.50	105.61	204.83
CBWS_55	0.00	1581.50	1581.50	1475.00	141.26	231.34
ClustBasedRandom	0.00	301.25	301.25	298.25	42.51	41.12
InstBasedRandom	0.00	246.00	246.00	239.50	44.14	40.93

Table 5.23 shows the # failures per strategy for iterations 10, 20, 30, 40, and 50. Recall that a failure is the number of times a strategy does not exceed ClusterBasedRandom or InstanceBasedRandom on any of the two metrics (total hits and total unique hits). Across all iterations MABSelector strategies fail the most, particularly at 10 iterations. Exploration heavy strategies CBWS_341, CBWS_609, and CBWS_55 fail the least across iterations, particularly, CBWS_341 failing only 2 times at iteration 50. Table 5.24 is a detailed per task-strategy failure matrix at 50 iterations. Since each task has only one run (one no-actives plate), the task total failure maximum is 6. Seven tasks are problematic/hard, as they fail on more than 3 strategies, these are task IDs: 588456, 504891, 602332, 492947, 624246, 743266, and 2326.

Thus, based only on experiment 3.2 conditions (no-actives starting plate) and conducted analysis, one would select CBWS_341 for maximizing total hits, and CBWS_609 for maximizing total unique

hits (i.e. novelty).

Table 5.23: Experiment 3.2 failure counts for each strategy over all task-plate runs (102 tasks and 1 initial starting plates = 102 runs). # failures denotes the number of runs that a strategy does not exceed ClusterBasedRandom or InstanceBasedRandom on total hits or total unique hits.

	# Failures for all 102 task-plate runs				
# iterations:	10	20	30	40	50
CBWS_341	23	21	12	6	2
CBWS_55	31	15	12	12	9
CBWS_609	29	21	16	13	9
MABSelector_2	71	45	31	25	24
MABSelector_exploitive	68	45	33	28	26
CBWS_custom_1	33	24	21	17	12

Table 5.24: Experiment 3.2 per task failures for each strategy after 50 iterations (1 run per task). # failures denotes the number of runs that a strategy does not exceed ClusterBasedRandom or InstanceBasedRandom on total hits or total unique hits.

Task ID	CBWS_341	CBWS_55	CBWS_609	MABSel_2	MABSel expl	CBWS cstm_1	Task Total
588456	0	0	1	1	1	1	4
463254	0	1	0	0	0	0	1
504891	1	1	1	1	1	1	6
602332	0	1	1	1	1	1	5
492947	0	1	1	1	1	1	5
624246	1	0	0	1	1	1	4
504845	0	0	0	1	1	0	2
504842	0	0	0	1	0	0	1
2662	0	0	0	0	1	0	1
602233	0	0	0	1	1	0	2
2675	0	0	0	1	1	0	2
485294	0	0	0	0	1	0	1
1469	0	0	0	1	1	1	3
720707	0	0	0	1	1	0	2
504706	0	0	0	1	1	0	2
652025	0	0	0	1	1	0	2
624291	0	0	0	1	1	0	2
720711	0	0	0	1	1	0	2
743266	0	1	1	1	1	1	5
485281	0	0	0	1	1	0	2
602310	0	0	0	1	1	0	2
602179	0	1	1	0	0	1	3
1471	0	1	0	1	1	0	3
720709	0	0	0	1	1	0	2
720708	0	0	0	1	1	0	2
485353	0	0	1	0	0	0	1
602313	0	0	0	1	1	0	2
651644	0	0	0	1	1	0	2
624171	0	0	0	1	1	0	2
1631	0	0	0	0	0	1	1
720551	0	0	1	0	0	1	2
2326	0	1	1	0	1	1	4
624288	0	1	0	0	0	1	2
Strategy Total	2	9	9	24	26	12	-

Promotion to Prospective Experiment 4

A final strategy is to be selected for prospective experiment 4. Based on the conducted analysis in experiments 3.1 and 3.2, we list the following arguments for promotion:

- Overall best performance based on **total hits** from early (0 to 30 iterations) to late retrieval (30 to 50 iterations): **MABSelector_exploitive**.
- Overall best performance based on **total unique hits** from early (0 to 30 iterations) to late retrieval (30 to 50 iterations): **CBWS_609**.
- Early retrieval for a small number of iterations (i.e. 10 or 20 iterations): **MABSelector_exploitive** for **total hits**, but **CBWS_609** for **total unique hits**.
- For small hit % like 0.01% to 0.15%, **total hits** differences are not apparent. For **total unique hits** differences are not apparent until hit $\% \geq 1.06\%$.
- In experiment 3.2, looking at the summary for 10, 20, 30, 40, and 50 iterations, it is evident that an exploration heavy method is better than exploitation heavy or even exploitation moderate. Overall from early to late iterations, **CBWS_341** is the best; this strategy assigns about 75% of the iteration budget towards exploration.
- Discussion with team members garners a sentiment towards a strategy that finds more novel hits, since these can later be optimized for desired properties. This argument is stronger in the case of academic labs that do not have the resources to run for 50 iterations, i.e. 10 to 20 iterations is more appropriate.
- Prospective experiment 4 is set to run for 50 iterations on a task with unknown labels (hit % is unknown). An initial starting plate will **not** be provided. Thus, because we favor maximizing unique hits, and we have no initial starting plate, we opted towards the **CBWS_609** strategy. From experiment 3.1 and 3.2 results, CBWS_609 offers good evidence for overcoming these two concerns and it is competitive.

5.18 Experiment 4: Prospective Target PstP using CBWS_609

Overview and Goal

In experiment 4 we have the prospective target phosphoserine/threonine phosphatase (PstP) with a 94,044 compound pool. We have no % inhibition or hit labelling information for this pool. We only have access to compound-only information like SMILES, structure, and associated features. The goal is to conduct a prospective iterative screening for 50 iterations on this target using CBWS_609.

There are two parties involved in this experiment: the **query-selector** and the **oracle** (also known as the label-provider). The query-selector is the promoted experiment 3 strategy (**CBWS_609**) that selects compounds for labelling in each iteration. The oracle then takes query requests for labelling, and knowing the *true* labelling, generates these labels for the query-selector. Thus a single iteration involves: running the strategy with the current iteration's training and unlabelled pool, presenting the strategy's current iteration selected compounds to the oracle for labelling, and the oracle generating the requested labelling back to the query-selector. In other words, each iteration is a request-generate event to mimic a prospective screen.

This experiment was run for 50 iterations, with each iteration's request-generate event being documented as commits in an independent GitHub repository. This repository will be made public once the PstP dataset is published and made public. The query-selector commits its compound requests for the current iteration, and in response, the oracle downloads the request and commits its label generation response. The **batch size** is set to 80 since it is common to have 16 controls for a 96-well plate. The first iteration was run with **no training data**; essentially making no use of the machine learning model in **CBWS_609**. Although there is no initial training data in the first iteration, CBWS_609 will still run and select a batch of compounds from the unlabelled pool. In such a case, the supervised learning model has no training data, and thus will output 0.5 activity probabilities for all pool compounds. Finally, we have a single compound that is hypothesized to be active that was included in the first iteration's request. This compound is Aurintricarboxylic acid (ATA) with PubChem CID 2259, which is a known inhibitor of a related target serine/threonine

phosphatase (Stp1) [147].

As for the hit/label generation process, the PstP compound pool of 94,044 have already been in vitro screened prior to this experiment. The screening results were processed and normalized to determine an activity cutoff based on primary and secondary screens. This was done by a collaborative lab, and so, the query-selector has no access to the *true* hit labels. This is why there is a need for an oracle contact that has access to these labels. The oracle is essentially an individual from the screening group.

One major assumption we are making in this experiment is that there already exists a confident hit labelling scheme for the PstP pool of 94,044 compounds that is provided by the oracle. Recall that the oracle generated these hit labels by screening the entire 94,044 compounds. A more realistic option is that hit/label generation can also be done adaptively by processing each batch's screening results or by processing all screening results so far (up to the current iteration). This would mimic hit determination in a practical iterative screening setting. That is, with each iteration, we screen more compounds in vitro and gather more screening data, which in turn would give us more information about the distribution of % inhibition scores. Thus, with each iteration, we refine the hit threshold thereby changing the labelling of compounds. However, as we mentioned, the 94,044 pool compounds were already pre-plated, screened, and a hit threshold was established based on the pool's % inhibition distribution. For future work, we plan to work on developing an adaptive hit labelling scheme for iterative screening.

Results and Discussion

After completing this experiment, the oracle provided the true labelling for performance analysis. The results are summarized as follows:

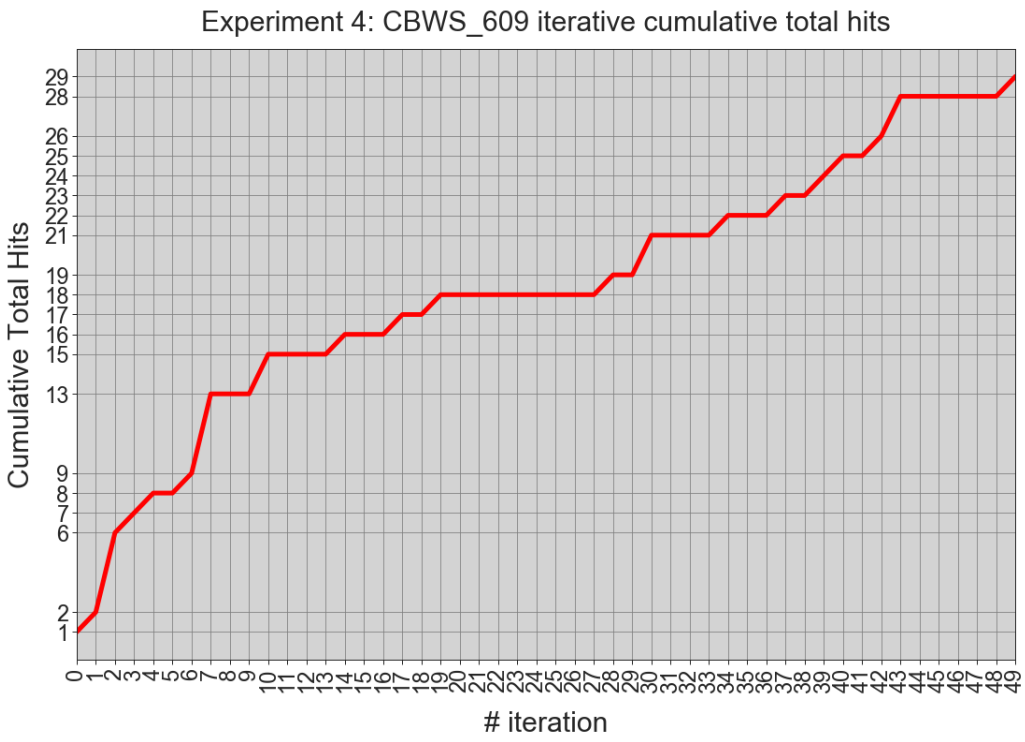
- 50 iterations. Batch size: 80 compounds. Total selected compounds: 4000.
- Dataset size: 94,044. Total hits in dataset: 143. Total unique hits in dataset: 127.
- Hit % in dataset: 0.15%. Hits expected at random in 4000 compounds: 6.04 hits.

- Total hits found by CBWS_609: 29. Total unique hits found by CBWS_609: 27.
- Percent of dataset screened: 4.25%. Percent of hits found: 20.28%. Percent of unique hits found: 21.26%.

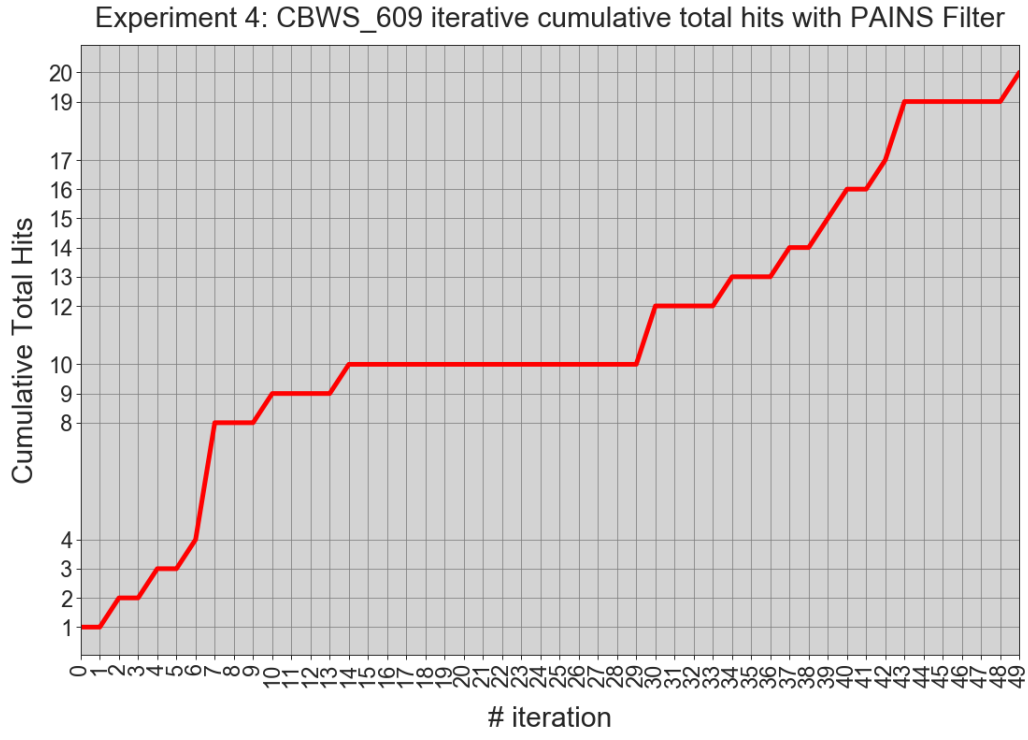
Uniqueness is defined by Taylor-Butina [80, 81] clustering at the 0.4 threshold. The hit labelling by the oracle did not take into account a PAINS filter [59] when generating the labelling. After taking the PAINS filter into account, the total hits in the dataset are 125, total unique hits in the dataset are 110, and CBWS_609 finds 20 and 19 total hits and total unique hits, respectively. Consequently, the expected hits to be found with the PAINS filter is 5.32 hits. Percent of hits and unique hits found are 16.00% and 17.27%, respectively.

Figure 5.21 shows the iterative cumulative hits progress with and without the PAINS filter. Both plots show similar trends across the iterations. From iterations 0 to 7, CBWS_609 makes reasonable progress, averaging a hit per iteration. Iterations 7 to 14 CBWS_609 slows down with only a 2-3 hit increase. The strategy stagnates from iterations 14 to 29 with no hit increase with the PAINS filter. Iterations 29 to 50 show uptick in CBWS_609 hit finding in both plots. We see interchanging periods of hit-accumulation and exploration. Intuitively, we expect that periods of hit-accumulation to become fewer as we progress in iterations; the few remaining hits become increasingly problematic to find.

With and without the PAINS filter, the results exceed the expectation of hits at random. Without the PAINS filter, we expect to find 6.04 hits in 4000 randomly selected compounds. CBWS_609 finds 29 hits, that is ~4.8 times more than expected at random. With the PAINS filter, we expect to find 5.32 hits in 4000 randomly selected compounds. CBWS_609 finds 20 hits, that is ~3.76 times more than expected at random. Under both settings, more than 90% of the hits found belong to distinct clusters.



(a) Without PAINS filter



(b) With PAINS filter

Figure 5.21: Experiment 4 per iteration cumulative total hits (a) without and (b) with PAINS filter.

5.19 Why are the top CBWS strategies exploration heavy?

To answer this question, there are two aspects to consider. The first is which CBWS parameter settings promote full-exploration and no-exploitation selections? This helps us understand, technically or algorithmically, why strategies like CBWS_609 and CBWS_55 select no exploitation compounds. The second is how are full-exploration CBWS strategies able to find many hits, considering that exploration relies on the uncertainty score of the machine learning model? Since exploration should promote the most uncertain compounds, then it follows that these are not the most predicted to be active compounds. We tackle each question separately, and together, they help explain why exploration heavy CBWS strategies do well.

What CBWS parameter settings lead to few or no exploitation selections?

Analyzing why some CBWS hyperparameters do little to no exploitation selection was done by manually checking the top performers' settings for exploitation parameters. Most of the top performers had settings that should, conceivably, perform exploitation selections. There are four main parameters that affect exploitation selection:

- **exploit-activity-threshold**: Each cluster's compounds whose hit predictions exceed this threshold are used for computing the cluster's mean activity. Thus, if no compound prediction exceeds this, the cluster's mean activity will be set to zero.
- **exploit-use-quantile-for-activity**: If this is set to true, then **exploit-activity-threshold** is based on the quantile prediction of the cluster (rather than an absolute value in the false case).
- **exploit-weight-threshold**: Clusters with exploitation weights exceeding this threshold are selected for exploitation. If no cluster exploitation weight exceeds this, then no exploitation takes place. This can happen if, for example, all cluster mean activity are low due to the model, or zeroed out due to not passing the activity threshold.

- **exploit-use-quantile-for-weight**: If this is set to true, then **exploit-weight-threshold** is based on the quantile exploitation weight among exploitation clusters.

Thus, to guarantee that some exploitation takes place every iteration (although this may not be the best strategy), then set **exploit-use-quantile-for-activity** and **exploit-use-quantile-for-weight** to true. This would ensure that the top-quantile/top-percentile of unlabeled pool predictions are selected for exploitation.

Table 5.25 shows the settings of these parameters for exploration heavy CBWS strategies seen in experiments 2 and 3. First consider the exploration heavy strategies that were used in experiment 2, but not experiment 3: CBWS_411, CBWS_678, CBWS_590, CBWS_396, and CBWS_467. These were run on task aid624173 with 400,122 compounds and 487 actives. Due to the low hit %, the model predictions for this task will have a small range for prediction scores, lower than 0.5, as seen in Figures 5.22 for tasks aid588456 and aid602310. Recall that the supervised learning model used in CBWS is a random forest (RF). We also see that all these strategies have at least one of **exploit-use-quantile-for-activity** and **exploit-use-quantile-for-weight** set to False. Furthermore, all these strategies have either of the thresholds set to at least 0.5. These settings, along with the low prediction score range (less than 0.5) for task aid624173, lead to having few or no exploitation selections. Conceivably, these strategies may perform exploitation in later iterations when the model matures and the activity predictions are pushed to the edges. Finally, if we look at **exploitation-weight-threshold** setting for CBWS_609 and CBWS_55, it is set to 1.0, and so no exploitation will ever take place.

Table 5.25: Exploitation parameter setting values for the exploration heavy CBWS strategies used in experiments 2 and 3.

HS ID	exploit alpha	exploit activity_threshold	exploit weight_threshold	exploit use_quantile for_activity	exploit use_quantile for_weight
CBWS_609	0.75	0.50	1.00	False	False
CBWS_411	0.25	0.50	0.00	False	False
CBWS_678	0.50	0.25	0.50	True	False
CBWS_55	0.75	1.00	1.00	False	True
CBWS_590	0.25	0.50	0.50	False	True
CBWS_396	0.25	0.25	0.50	True	False
CBWS_341	1.00	0.75	0.75	True	False
CBWS_467	0.75	0.75	0.25	False	False

Why do CBWS compounds selected for exploration lead to many hits?

Having seen the technical settings that lead to few or no exploitation selections, a logical followup is why does exploration lead to many hits. Recall that exploration makes use of the machine learning model's uncertainty scores and dissimilarity of compounds. If maximum exploration favors the most uncertain compounds, why does this lead to many hits. To investigate this, we restrict our focus on CBWS_609's iterative selections on the 102 tasks in experiment 3.1 with the initial plate #0. Since we have already run experiment 3.1, we have the compounds selected at each iteration; we have the training and unlabeled pool compounds at each iteration. Particularly, for each task x, we use the training set for task x at iteration i to train a random forest model (RF is used for all CBWS strategies), and then predict on the unlabeled pool at iteration i. This allows us to look at the range and distribution of prediction and uncertainty scores at iteration snapshots.

First, we look at side-by-side prediction vs uncertainty comparisons for three tasks of varying hit %. Figure 5.22 shows the kernel density estimate (KDE) distribution plots for RF predictions and uncertainty scores of the unlabeled pool compounds at iterations 1, 10, 20, 30, 40, and 50. Plots are denoted by the task and iteration (a.x) aid588456 with 0.01% hits, (b.x) aid602310 with 0.08% hits, and (c.x) aid1458 with 2.97% hits. Interestingly, we can see that for the first two low hit % tasks, the max of prediction scores is less than 0.5. This is important since we used least-confidence

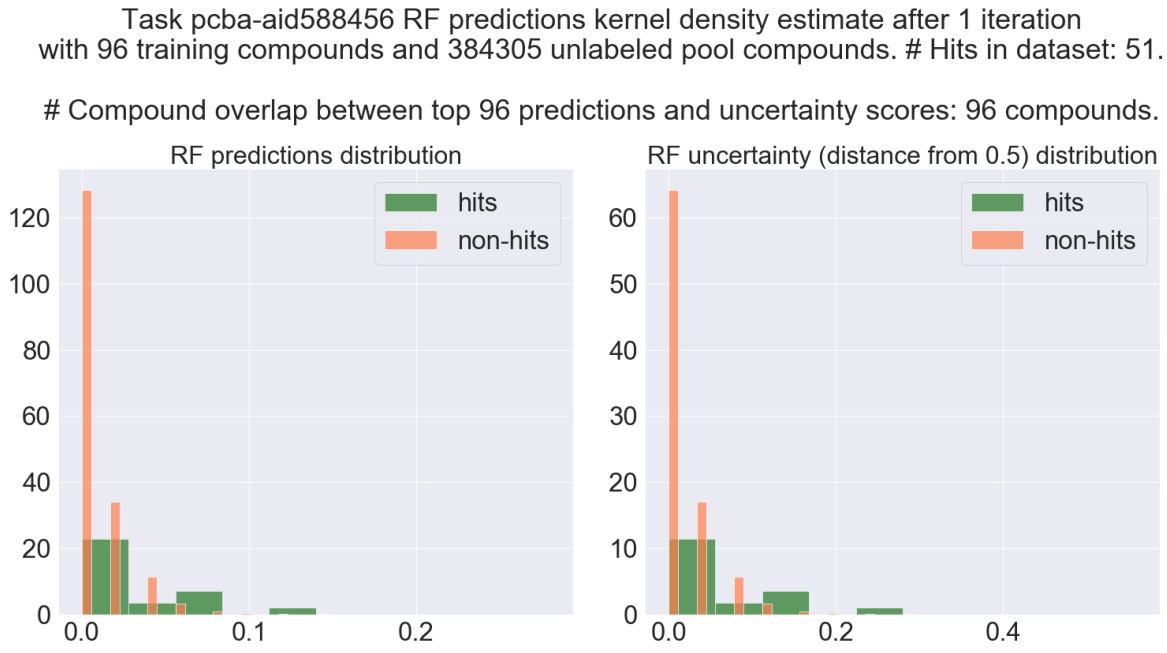
to quantify uncertainty, and if you recall from section 5.8 in Equation 5.6, least-confidence is the hit prediction distance from 0.5; i.e. predictions closer to 0.5 are closer to 1 uncertainty, whereas predictions closer to the 0 or 1 are closer to 0 uncertainty. So if all predictions are less than 0.5, then the ordering of compounds induced by prediction score and uncertainty score are the same. In other words, **exploration in this case is the same as exploitation**. Furthermore, if the predictions are at most 0.75, then there is potential for large overlap/intersection in the top compound ordering induced by prediction and uncertainty scores.

For the higher hit % task, aid1458, we see that the first iteration is problematic as the predictions are still less than 0.5. But at iteration 10 and beyond, we see that prediction scores encompass the full range from 0 to 1. The compound overlap in the top 96 based on predictions vs uncertainty goes to 0, meaning that prediction and uncertainty ordering are not the same. However, if we look at the uncertainty distribution of hits vs non-hits in Figures c.10, c.20, c.30, c.40, and c.50, we see that the tail of the hit distribution is heavier at high uncertainty. This means that exploration based on uncertainty in this case is favoring selection of hit compounds.

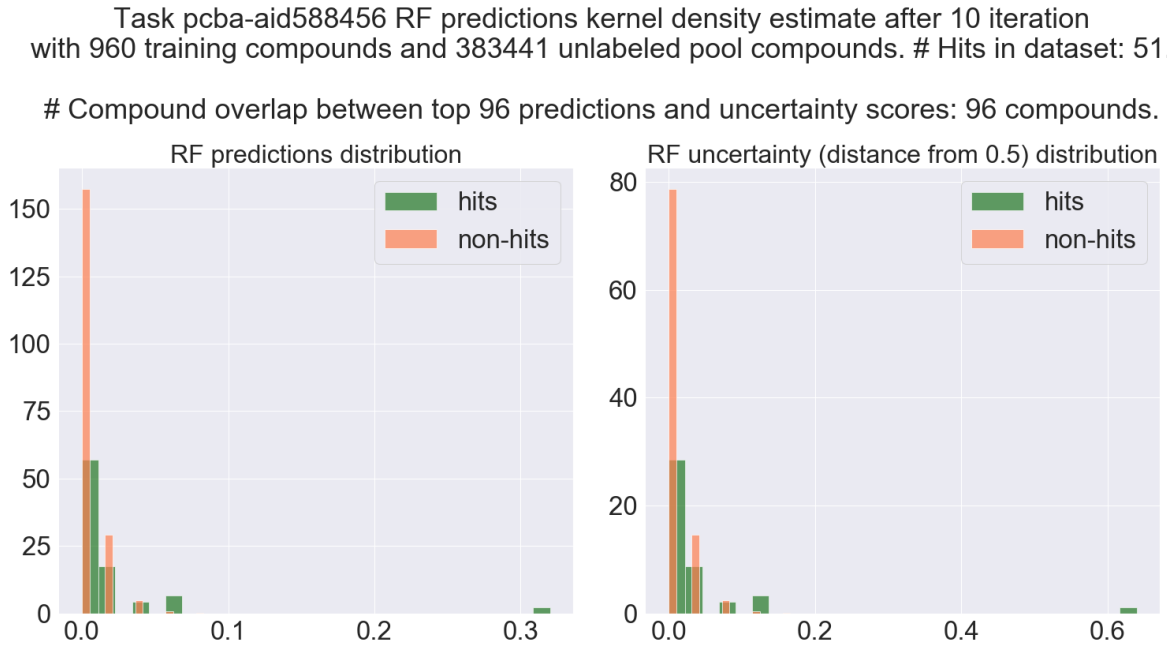
Table 5.26 summarizes the overlap counts for all 102 tasks from experiment 3.1. These are the task overlap/intersection counts between the top 96 unlabeled compound RF predictions and uncertainty scores at iterations 1, 10, 20, 30, 40, and 50. The mean overlap count is aggregated over these iterations. Figure 5.23 plots the histogram of the mean overlap counts for all 102 tasks. More than 75 (out of 102) tasks have a mean overlap count over 80 (max overlap is 96). In contrast, only 11 tasks have a mean overlap count less than 20; these tasks have some of the highest hit % ranging from 1.26% to 6.17%.

To summarize, we have two issues that can occur. The first case is when the prediction scores have a maximum that is close to 0.5. This results in the top compound ordering based on prediction and uncertainty being very similar; i.e. exploration and exploitation in CBWS are the same. The second case is when the prediction scores go from 0 to 1, but the hit distribution is heavier than the non-hit distribution around the 0.5 prediction mark. This results in exploration, that is based on uncertainty, favoring selection of hit compounds. We want exploration to exhibit a different

behavior than exploitation. These two issues can stem from a number of factors, mainly the small training sets, low hit counts within the training sets, and a random forest model. The random forest model is an ensemble of decision trees which are built using random sample and feature subsets [63]. Thus, when trained on a small training set, this randomness coupled with ensembling predictions can lead to the low prediction scores, particularly at the starting iterations. The hit distribution having heavier tails can also be a byproduct of the randomness involved in sampling feature subsets when building individual decision trees. This random sampling of feature subsets leads to more robustness at the cost of confidence when making predictions; i.e. the predictions shy away from giving a compound a score close to 1. The model will still favor giving hits a higher prediction score than non-hits, but most hits will not have a prediction close to 1. Rectifying this issue will require investigation into other model types and their corresponding prediction distributions. This is beyond the scope of this body of work and will be deferred to future work.

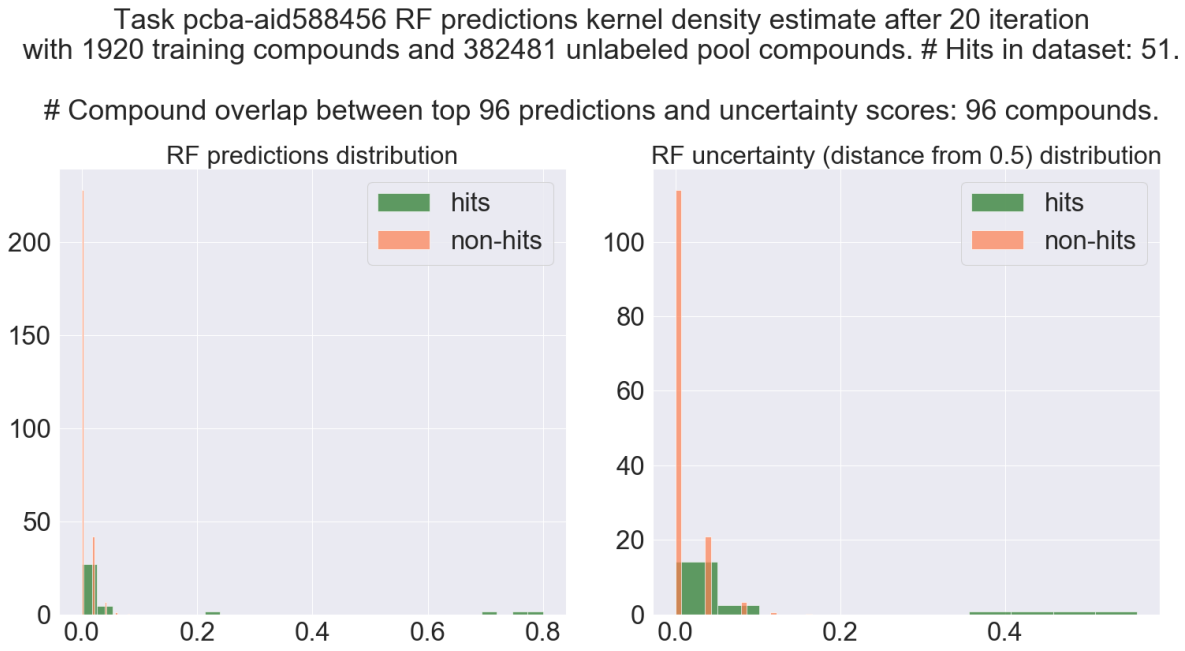


(a.1) Task aid588456 at iteration 1

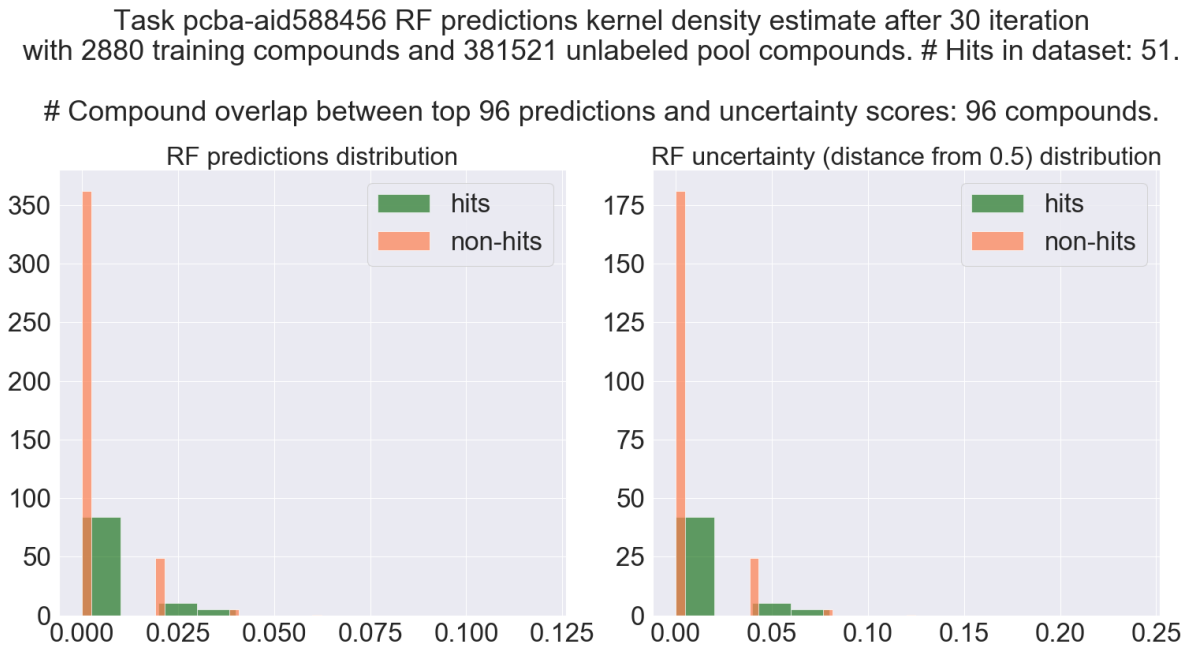


(a.10) Task aid588456 at iteration 10

Figure 5.22: Normalized histogram plots for RF predictions and uncertainty scores for unlabeled compounds at iterations 1, 10, 20, 30, 40, and 50. The RF is trained on the training set at the defined iteration and task with plate #0 in experiment 3.1. The plot sub-caption denotes the task and iteration: (a.x) aid588456, (b.x) aid602310, and (c.x) aid1458. *(1 of 9 cont.)*

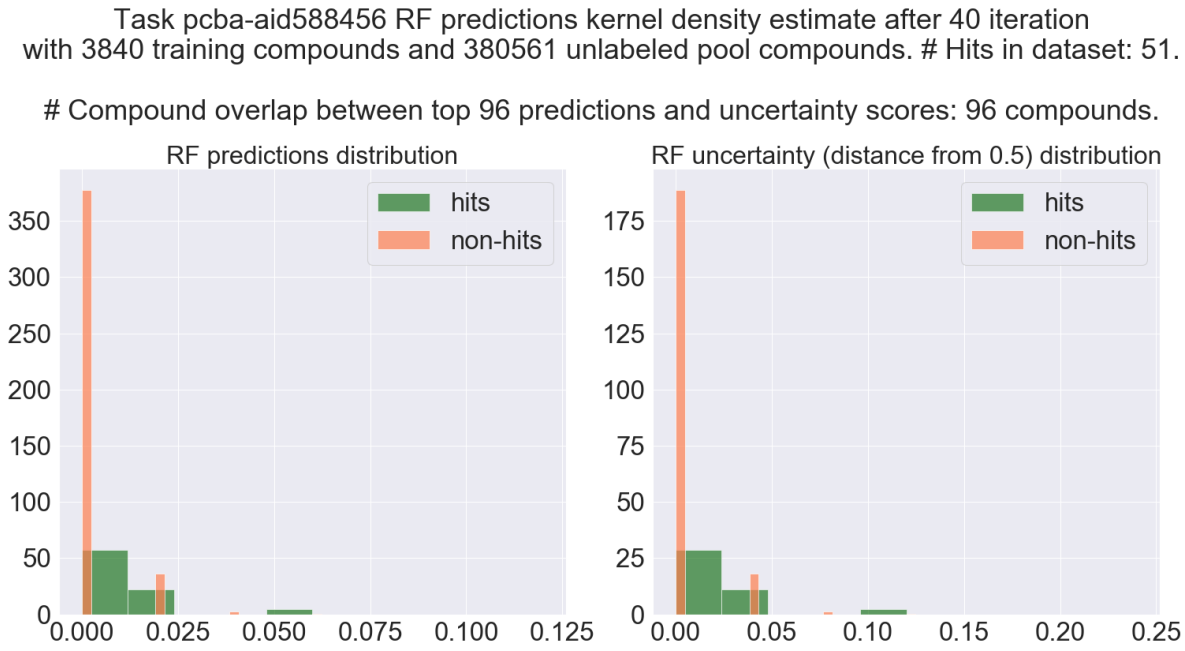


(a.20) Task aid588456 at iteration 20

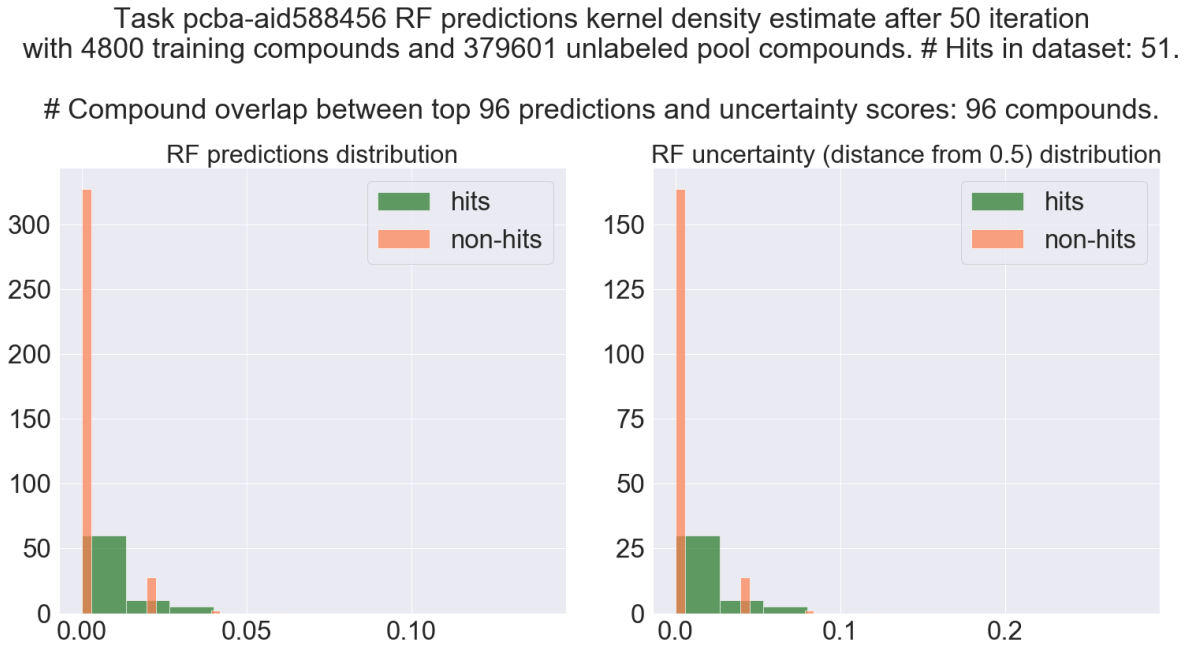


(a.30) Task aid588456 at iteration 30

Figure 5.22: Normalized histogram plots for RF predictions and uncertainty scores for unlabeled compounds at iterations 1, 10, 20, 30, 40, and 50. The RF is trained on the training set at the defined iteration and task with plate #0 in experiment 3.1. The plot sub-caption denotes the task and iteration: (a.x) aid588456, (b.x) aid602310, and (c.x) aid1458. *(2 of 9 cont.)*

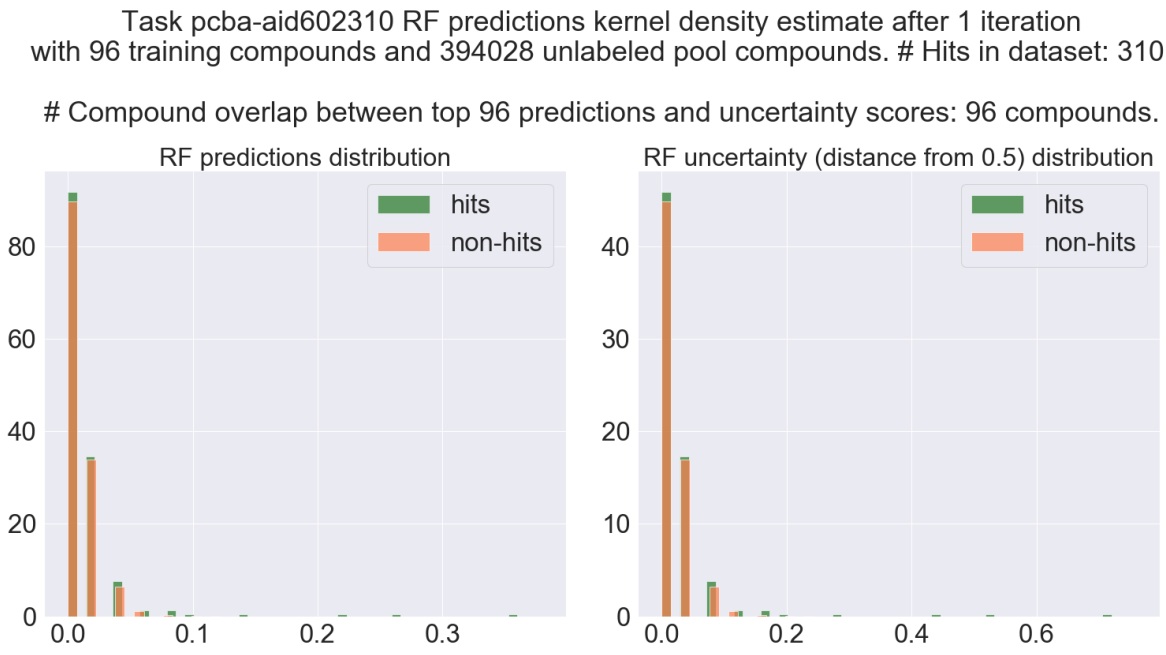


(a.40) Task aid588456 at iteration 40

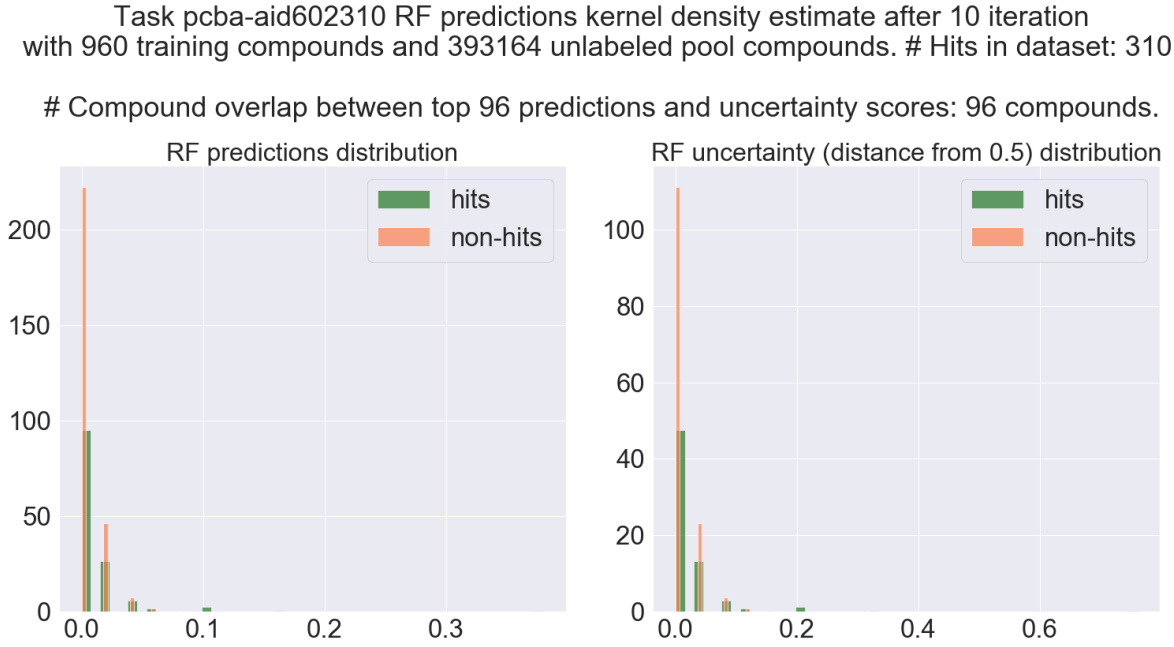


(a.50) Task aid588456 at iteration 50

Figure 5.22: Normalized histogram plots for RF predictions and uncertainty scores for unlabeled compounds at iterations 1, 10, 20, 30, 40, and 50. The RF is trained on the training set at the defined iteration and task with plate #0 in experiment 3.1. The plot sub-caption denotes the task and iteration: (a.x) aid588456, (b.x) aid602310, and (c.x) aid1458. *(3 of 9 cont.)*

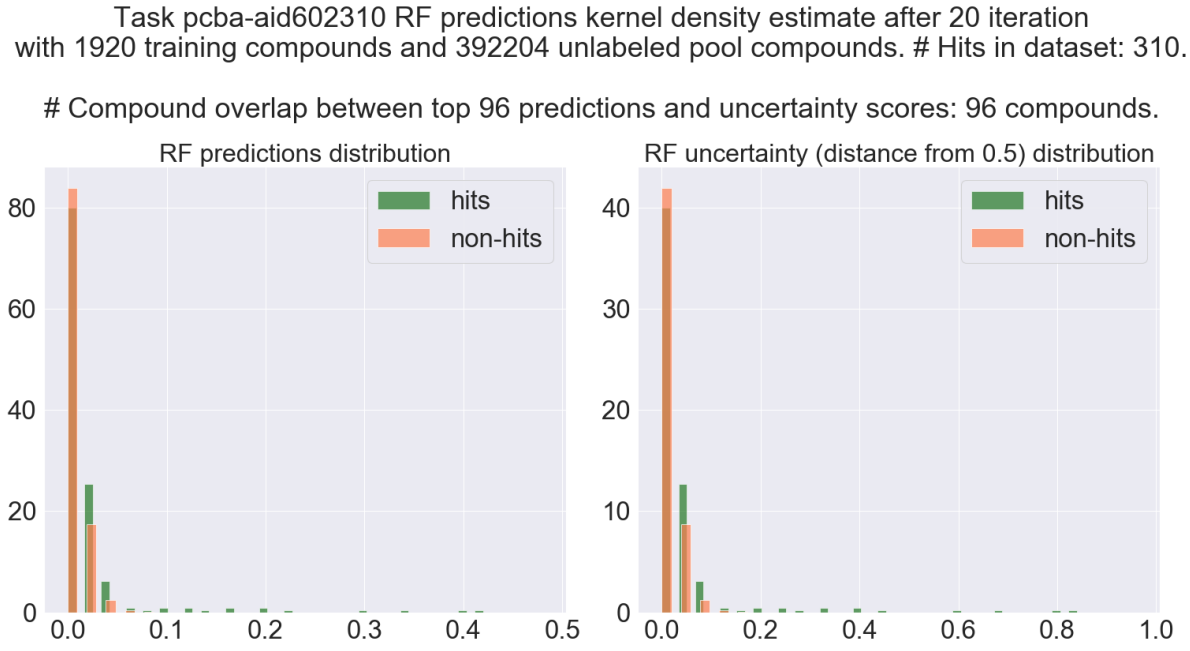


(b.1) Task aid602310 at iteration 1

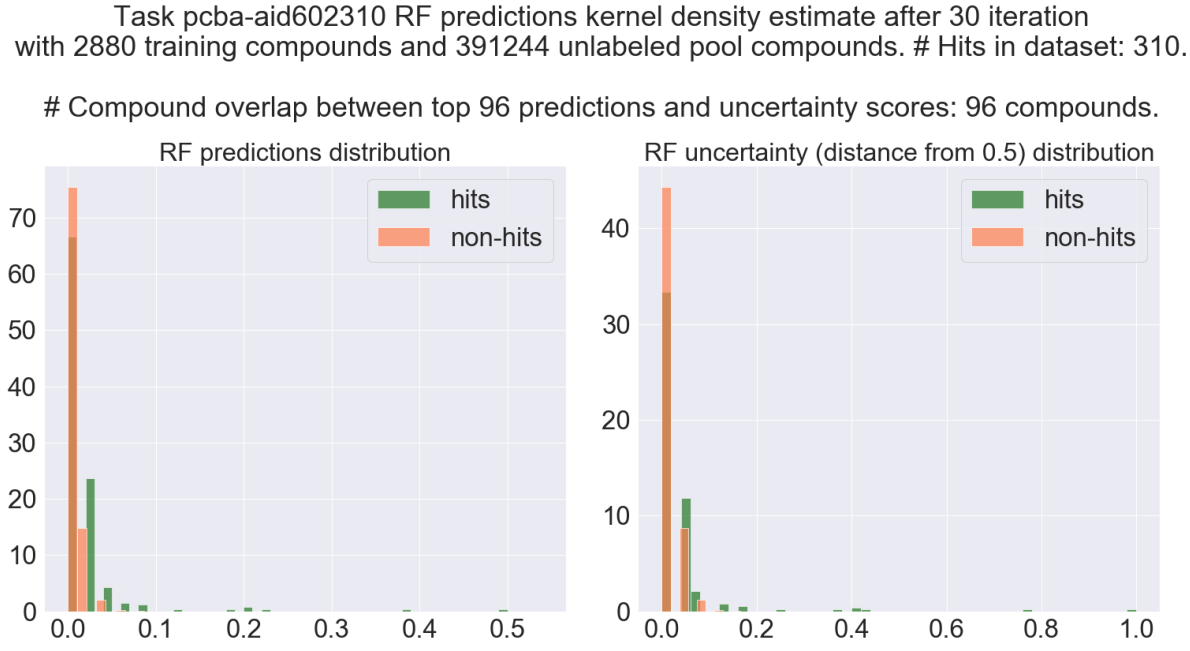


(b.10) Task aid602310 at iteration 10

Figure 5.22: Normalized histogram plots for RF predictions and uncertainty scores for unlabeled compounds at iterations 1, 10, 20, 30, 40, and 50. The RF is trained on the training set at the defined iteration and task with plate #0 in experiment 3.1. The plot sub-caption denotes the task and iteration: (a.x) aid588456, (b.x) aid602310, and (c.x) aid1458. *(4 of 9 cont.)*

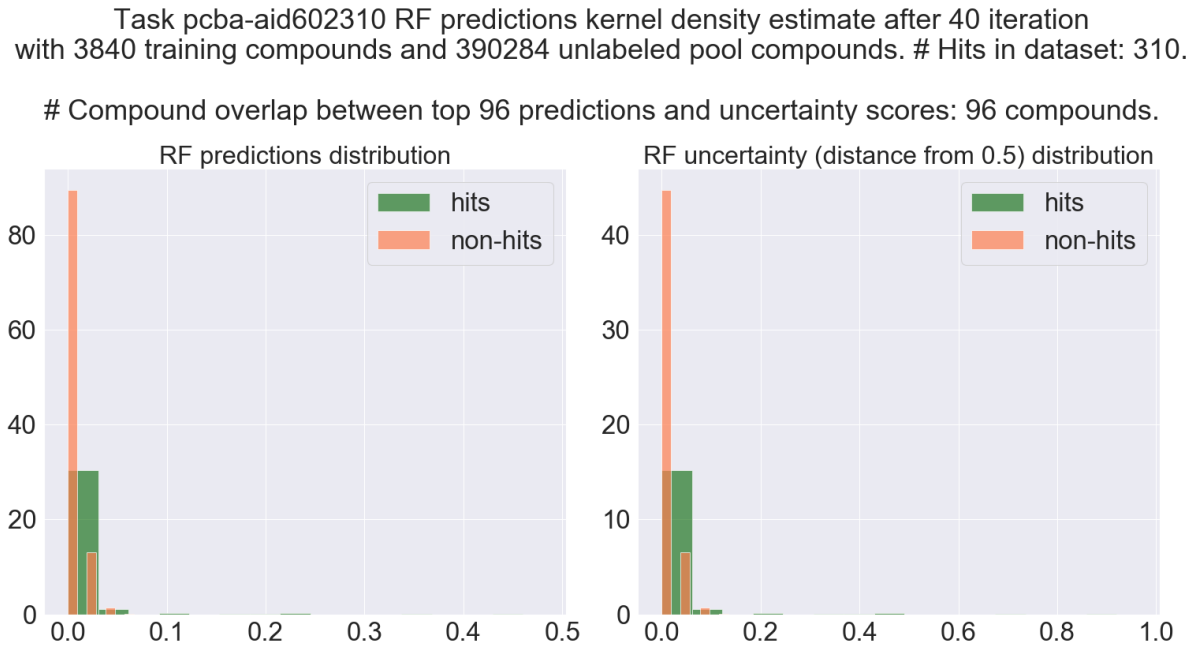


(b.20) Task aid602310 at iteration 20

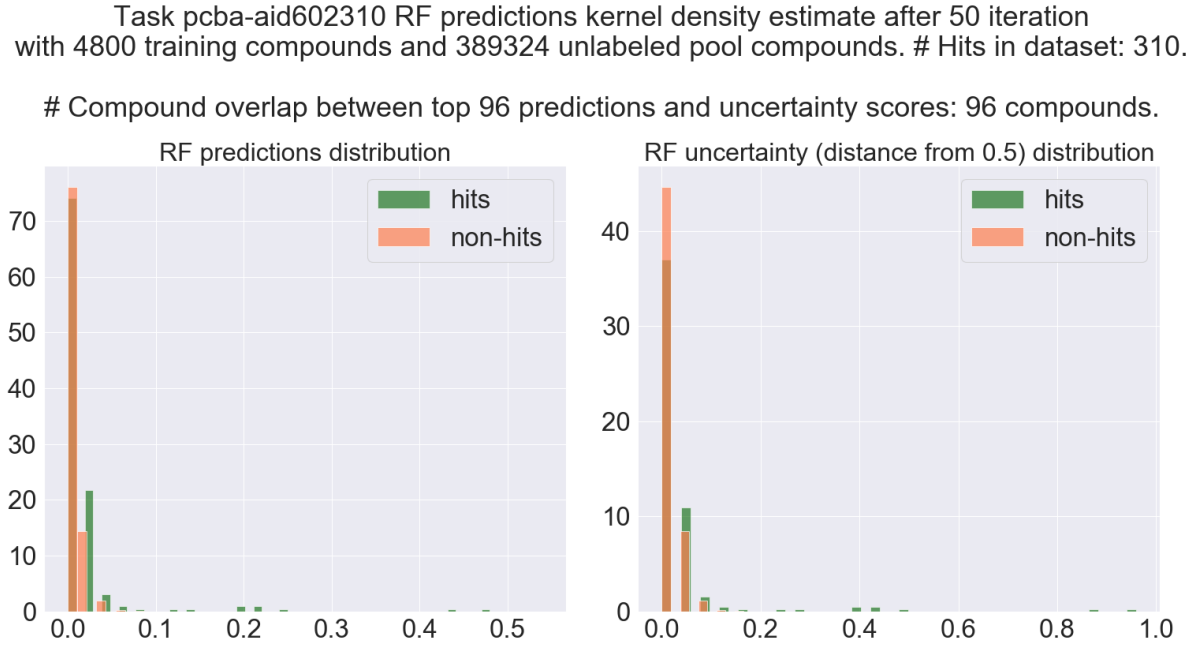


(b.30) Task aid602310 at iteration 30

Figure 5.22: Normalized histogram plots for RF predictions and uncertainty scores for unlabeled compounds at iterations 1, 10, 20, 30, 40, and 50. The RF is trained on the training set at the defined iteration and task with plate #0 in experiment 3.1. The plot sub-caption denotes the task and iteration: (a.x) aid588456, (b.x) aid602310, and (c.x) aid1458. *(5 of 9 cont.)*

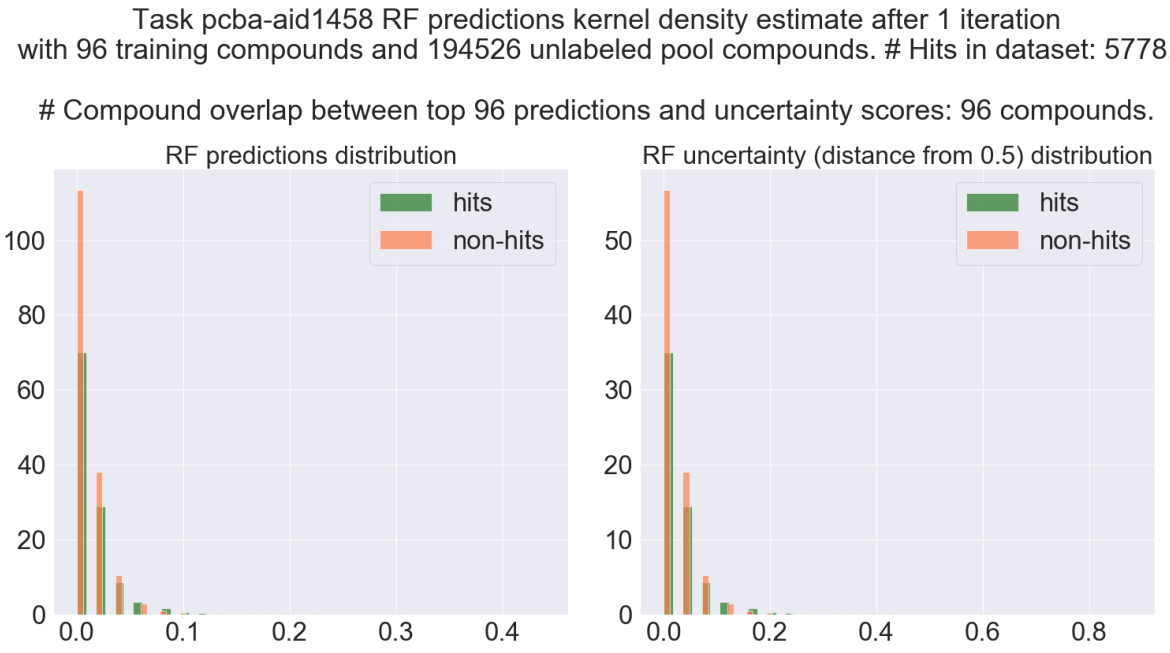


(b.40) Task aid602310 at iteration 40

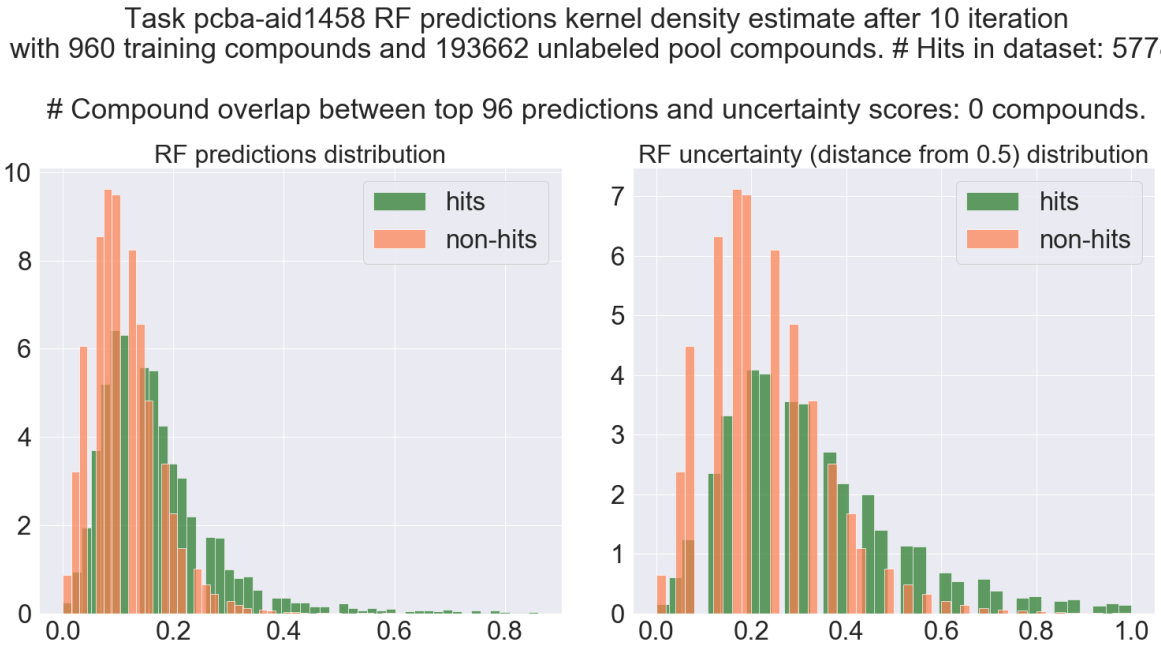


(b.50) Task aid602310 at iteration 50

Figure 5.22: Normalized histogram plots for RF predictions and uncertainty scores for unlabeled compounds at iterations 1, 10, 20, 30, 40, and 50. The RF is trained on the training set at the defined iteration and task with plate #0 in experiment 3.1. The plot sub-caption denotes the task and iteration: (a.x) aid588456, (b.x) aid602310, and (c.x) aid1458. *(6 of 9 cont.)*

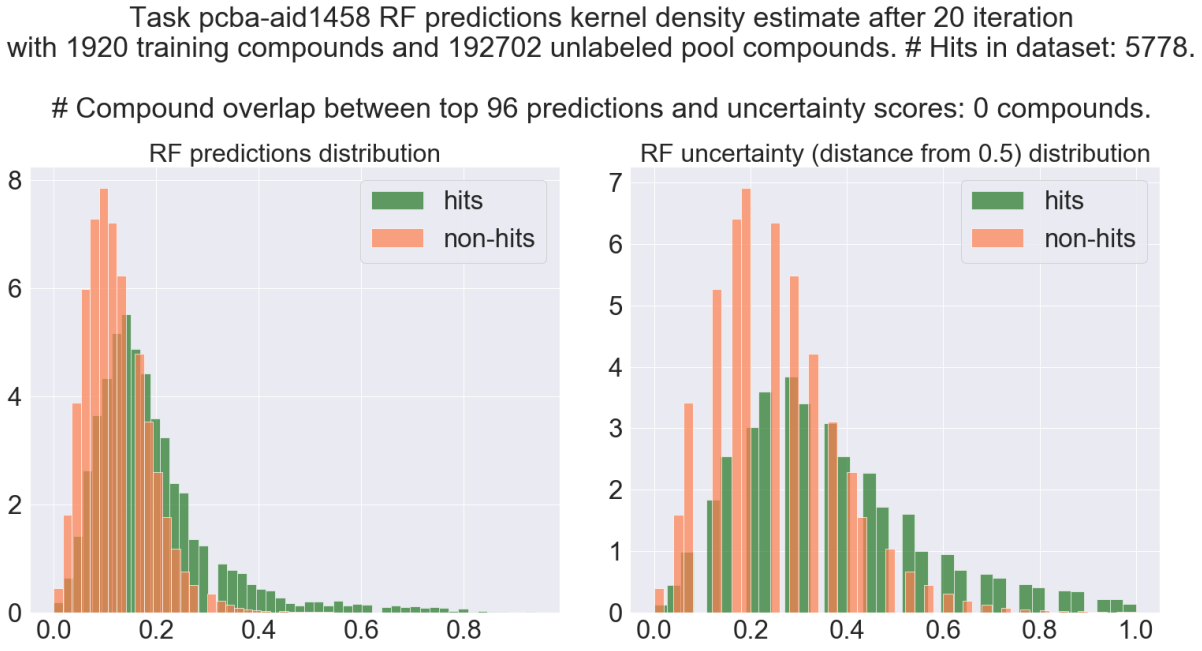


(c.1) Task aid1458 at iteration 1

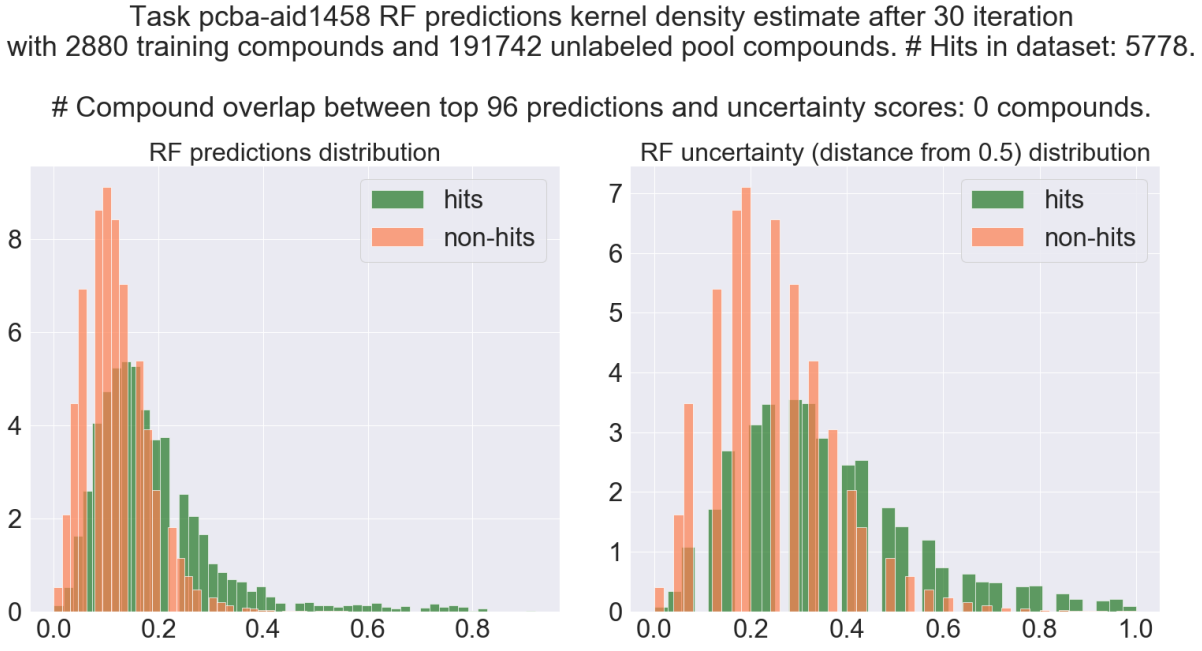


(c.10) Task aid1458 at iteration 10

Figure 5.22: Normalized histogram plots for RF predictions and uncertainty scores for unlabeled compounds at iterations 1, 10, 20, 30, 40, and 50. The RF is trained on the training set at the defined iteration and task with plate #0 in experiment 3.1. The plot sub-caption denotes the task and iteration: (a.x) aid588456, (b.x) aid602310, and (c.x) aid1458. *(7 of 9 cont.)*

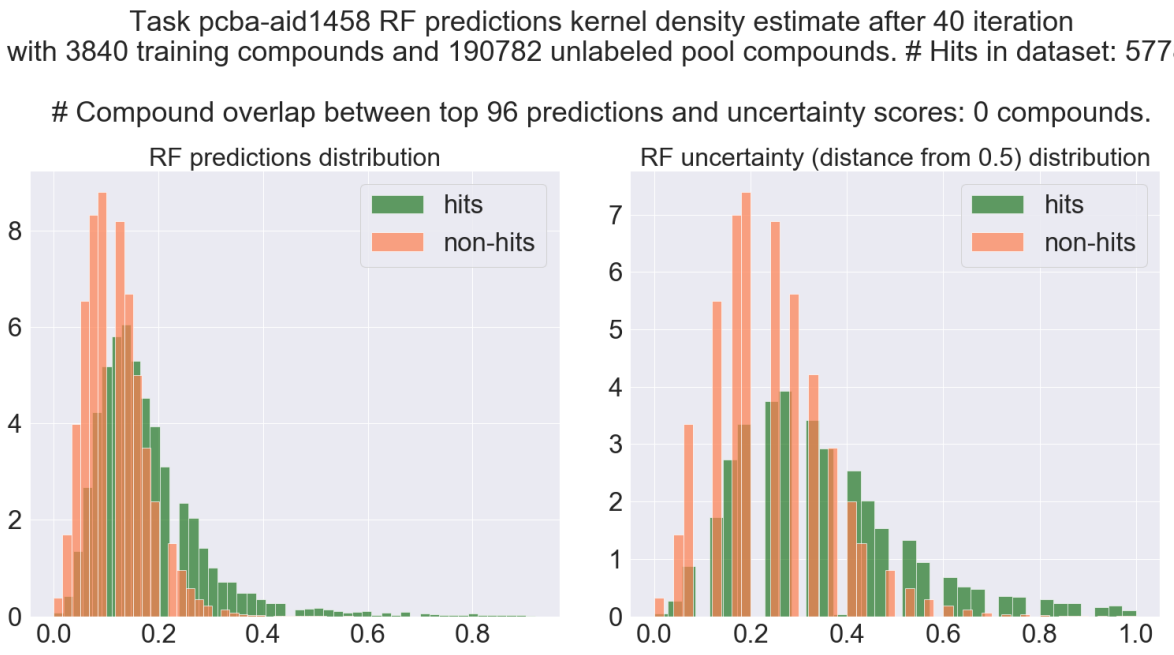


(c.20) Task aid1458 at iteration 20

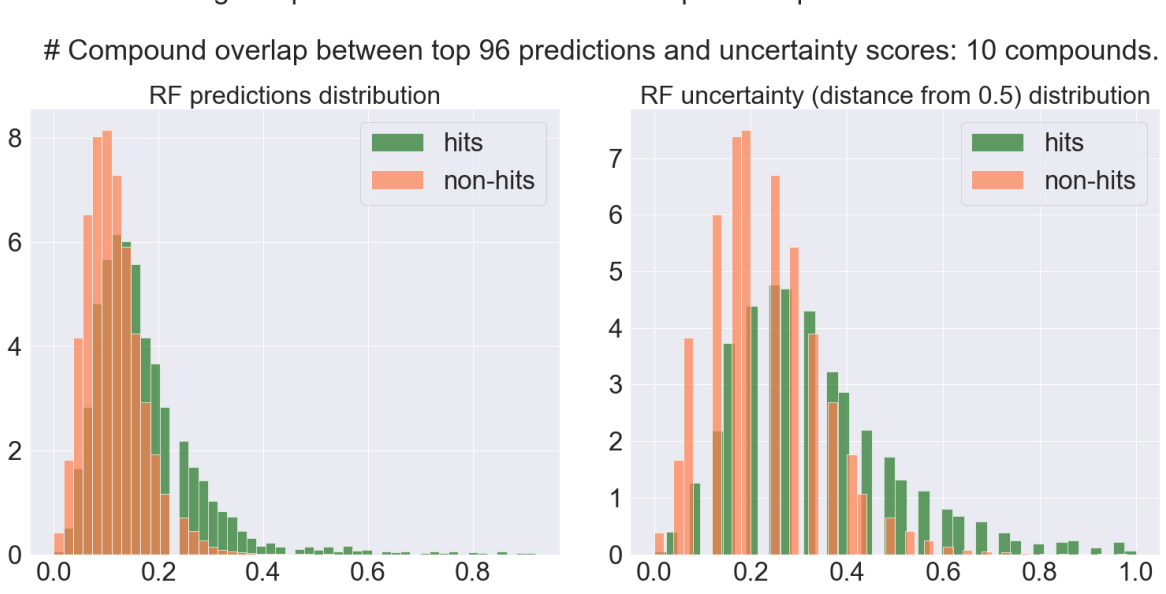


(c.30) Task aid1458 at iteration 30

Figure 5.22: Normalized histogram plots for RF predictions and uncertainty scores for unlabeled compounds at iterations 1, 10, 20, 30, 40, and 50. The RF is trained on the training set at the defined iteration and task with plate #0 in experiment 3.1. The plot sub-caption denotes the task and iteration: (a.x) aid588456, (b.x) aid602310, and (c.x) aid1458. *(8 of 9 cont.)*



(c.40) Task aid1458 at iteration 40



(c.50) Task aid1458 at iteration 50

Figure 5.22: Normalized histogram plots for RF predictions and uncertainty scores for unlabeled compounds at iterations 1, 10, 20, 30, 40, and 50. The RF is trained on the training set at the defined iteration and task with plate #0 in experiment 3.1. The plot sub-caption denotes the task and iteration: (a.x) aid588456, (b.x) aid602310, and (c.x) aid1458. *(9 of 9)*

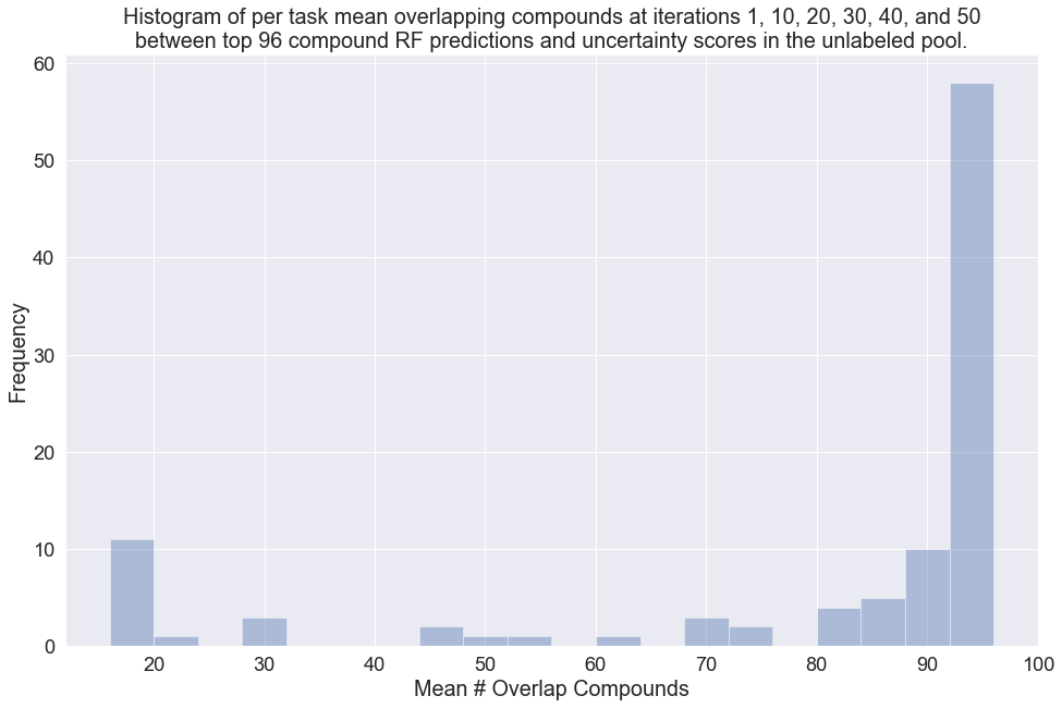


Figure 5.23: Task histogram of mean number of overlapping compounds aggregated across iterations 1, 10, 20, 30, 40, and 50 as seen in Table 5.26.

Table 5.26: Experiment 3.1 task overlap/intersection counts between top 96 unlabeled compound RF predictions and uncertainty scores at iterations 1, 10, 20, 30, 40, and 50. The RF model is trained on the task training set at the designated iterations. CBWS_609 and plate #0 for all tasks from experiment 3.1 are used.

Task ID	Hit %	Overlap 1 iters	Overlap 10 iters	Overlap 20 iters	Overlap 30 iters	Overlap 40 iters	Overlap 50 iters	Mean Overlap
588456	0.01	96	96	96	96	96	96	96.00
463254	0.01	89	96	96	96	96	96	94.83
504891	0.01	96	96	96	96	96	96	96.00
602332	0.02	96	96	96	96	96	96	96.00
492947	0.02	96	96	96	96	96	96	96.00
624246	0.03	96	96	96	96	96	96	96.00
504845	0.03	96	96	96	96	96	96	96.00
504842	0.03	96	96	96	96	96	96	96.00
2662	0.04	96	96	96	96	96	96	96.00
602233	0.04	96	96	96	96	96	96	96.00
2675	0.04	96	96	96	96	96	96	96.00

Continued on next page

Table 5.26: Experiment 3.1 task overlap/intersection counts between top 96 unlabeled compound RF predictions and uncertainty scores at iterations 1, 10, 20, 30, 40, and 50. The RF model is trained on the task training set at the designated iterations. CBWS_609 and plate #0 for all tasks from experiment 3.1 are used.

Task ID	Hit %	Overlap 1 iters	Overlap 10 iters	Overlap 20 iters	Overlap 30 iters	Overlap 40 iters	Overlap 50 iters	Mean Overlap
485294	0.05	96	96	96	96	96	96	96.00
1469	0.06	96	96	96	96	96	96	96.00
1634	0.06	96	96	96	96	96	96	96.00
652025	0.07	96	96	96	96	96	96	96.00
624291	0.07	96	96	96	96	96	96	96.00
720707	0.07	96	96	96	96	96	96	96.00
504706	0.07	96	96	96	96	96	96	96.00
720711	0.08	96	96	96	96	96	96	96.00
743266	0.08	96	96	96	96	96	96	96.00
485281	0.08	96	96	96	96	96	96	96.00
602310	0.08	96	96	96	96	96	96	96.00
2101	0.09	96	96	96	96	96	91	95.17
602179	0.09	96	96	96	96	96	96	96.00
1452	0.12	96	96	96	96	96	96	96.00
1471	0.13	96	96	96	96	96	96	96.00
652106	0.14	96	96	96	96	96	96	96.00
624287	0.14	96	96	96	96	96	96	96.00
720709	0.15	96	96	96	96	96	96	96.00
485367	0.17	96	96	96	96	96	96	96.00
720708	0.18	96	96	96	96	96	96	96.00
485353	0.19	96	96	96	96	96	96	96.00
485349	0.19	96	96	96	96	96	96	96.00
2528	0.19	96	96	96	96	96	80	93.33
602313	0.20	96	96	96	96	96	96	96.00
651644	0.21	96	96	96	96	93	95	95.33
720542	0.21	96	93	96	96	96	96	95.50
504327	0.21	96	96	96	96	96	91	95.17
624170	0.21	96	96	96	96	96	96	96.00
743255	0.24	96	85	72	72	77	78	80.00
1379	0.28	96	92	96	96	96	96	95.33

Continued on next page

Table 5.26: Experiment 3.1 task overlap/intersection counts between top 96 unlabeled compound RF predictions and uncertainty scores at iterations 1, 10, 20, 30, 40, and 50. The RF model is trained on the task training set at the designated iterations. CBWS_609 and plate #0 for all tasks from experiment 3.1 are used.

Task ID	Hit %	Overlap 1 iters	Overlap 10 iters	Overlap 20 iters	Overlap 30 iters	Overlap 40 iters	Overlap 50 iters	Mean Overlap
485290	0.28	96	96	96	88	90	90	92.67
1479	0.29	96	96	96	75	76	79	86.33
2676	0.30	96	96	96	96	96	96	96.00
624171	0.31	96	95	93	94	91	96	94.17
1631	0.34	96	96	96	96	96	96	96.00
2517	0.34	96	87	87	85	84	83	87.00
588795	0.35	96	85	85	82	79	86	85.50
1457	0.35	96	96	96	96	96	96	96.00
720551	0.37	96	96	96	96	96	96	96.00
1721	0.37	96	96	96	87	95	93	93.83
2242	0.39	96	78	82	96	96	96	90.67
2100	0.39	96	96	90	92	81	96	91.83
2326	0.41	96	96	96	96	96	96	96.00
1468	0.41	96	88	92	93	89	85	90.50
624288	0.42	96	96	96	96	96	96	96.00
1454	0.44	96	96	96	96	95	96	95.83
651768	0.47	96	71	79	81	83	80	81.67
720580	0.49	96	67	76	81	85	80	80.83
588579	0.51	96	50	54	34	24	26	47.33
2549	0.52	96	91	79	86	94	92	89.67
485341	0.53	96	96	96	96	96	96	96.00
881	0.57	96	95	96	96	96	95	95.67
540317	0.58	96	84	87	86	89	91	88.83
720579	0.67	96	88	85	86	88	88	88.50
485360	0.68	96	87	86	83	88	85	87.50
2451	0.73	96	87	85	96	83	84	88.50
504847	0.92	96	86	66	56	55	55	69.00
924	0.95	96	92	96	96	91	82	92.17
720553	0.96	96	88	87	89	88	87	89.17
588453	1.06	96	72	83	69	65	49	72.33

Continued on next page

Table 5.26: Experiment 3.1 task overlap/intersection counts between top 96 unlabeled compound RF predictions and uncertainty scores at iterations 1, 10, 20, 30, 40, and 50. The RF model is trained on the task training set at the designated iterations. CBWS_609 and plate #0 for all tasks from experiment 3.1 are used.

Task ID	Hit %	Overlap 1 iters	Overlap 10 iters	Overlap 20 iters	Overlap 30 iters	Overlap 40 iters	Overlap 50 iters	Mean Overlap
624202	1.08	96	96	96	96	96	96	96.00
651635	1.09	96	91	80	50	23	29	61.50
588590	1.10	96	61	31	62	46	26	53.67
1461	1.11	96	96	96	96	96	96	96.00
1688	1.16	96	82	91	91	90	82	88.67
588591	1.26	96	0	0	0	0	0	16.00
652105	1.26	96	0	0	0	0	2	16.33
504466	1.34	96	36	79	88	90	94	80.50
588855	1.39	96	89	75	79	65	50	75.67
485314	1.42	96	0	0	10	27	44	29.50
902	1.57	96	87	92	96	91	87	91.50
686970	1.76	96	84	76	64	65	34	69.83
2147	1.81	96	8	0	0	7	15	21.00
652104	1.90	96	94	92	94	89	96	93.50
651965	1.96	96	96	96	93	86	86	92.17
624417	1.96	96	96	90	96	86	95	93.17
624297	2.02	96	91	85	80	76	81	84.83
540276	2.23	96	66	52	35	21	17	47.83
485313	2.43	96	0	0	0	0	0	16.00
504444	2.54	96	96	71	50	56	59	71.33
1460	2.54	96	0	0	0	0	0	16.00
720504	2.90	96	87	0	0	0	0	30.50
485297	2.94	96	0	0	0	0	0	16.00
1458	2.97	96	0	0	0	0	10	17.67
485364	3.13	96	0	0	0	0	0	16.00
504467	3.14	96	63	35	42	30	38	50.67
624296	3.37	96	28	37	0	12	14	31.17
2546	3.79	96	0	0	0	0	0	16.00
504339	4.74	96	0	0	0	0	0	16.00
504333	4.81	96	0	0	0	0	0	16.00

Continued on next page

Table 5.26: Experiment 3.1 task overlap/intersection counts between top 96 unlabeled compound RF predictions and uncertainty scores at iterations 1, 10, 20, 30, 40, and 50. The RF model is trained on the task training set at the designated iterations. CBWS_609 and plate #0 for all tasks from experiment 3.1 are used.

Task ID	Hit %	Overlap 1 iters	Overlap 10 iters	Overlap 20 iters	Overlap 30 iters	Overlap 40 iters	Overlap 50 iters	Mean Overlap
2551	6.17	96	0	0	0	0	0	16.00

5.20 Lessons Learned and Future Work

Valuable observations and lessons were learned over the experiments. Here we reflect on these and discuss future prospects.

The CBWS strategy has a huge number of parameter settings, even with discretization, there are 600 million possible configurations. In experiment 0, we sampled 800 and 200 of these parameters using prior and random distributions, respectively. There is no guarantee that this manually defined prior is correct or favorable, but rather it was guided by intuition. 1000 samples is not enough to confidently cover the space, but limitations of resources and time led to this decision. A Bayesian optimization approach with multiple starting points may help in searching the space, but even then there are issues to consider when deciding how close two parameter settings are in the parameter space. It is important to note that we set out to find *good* CBWS settings (better than random), rather than optimal. In retrospect, the reason we exposed a large number of parameters in CBWS is because this was a new endeavour and it was not clear which parameters or settings were important. Furthermore, this decision allowed us to express various strategy behaviours with relative ease. After running the experiments, it is apparent that some parameters are more influential than others, and so, CBWS will be revised in future work.

In experiments 0 and 1, we tried to include all three batch sizes (96, 384, and 1536), however, increasing job failures due to time and memory constraints for 384 and 1536 batch sizes dissuaded us from using them in subsequent experiments. Consequently, our promotion criteria in experiments

2 and 3 were based on a batch size of 96. Although we do not believe that a best strategy under a batch size of 96 is the same under a batch size of 1536, we do however intuitively expect that there is overlap in the top set of strategies among different batch sizes. In experiment 4, we opted to use a batch size of 80 due to 16 wells being used for control, but this batch is close to 96 and so we were at ease with using the promoted strategy.

The batch size is important to consider in practice since scheduling and conducting an iteration's physical screen takes more time and cost than running a strategy to computationally select compounds. These costs include purchasing, delivery, scheduling, and physically screening compounds as well as the costs of manual labor. As such, a lab will need to plan an iterative campaign around these cost considerations. For example, if it takes one week to physically screen compounds for batch sizes between 96 and 6144 (4 plates of 1536), then a lab needs to consider how many iterations to run, strategy accuracy, and compound purchase costs. Future work should consider a focused set of experiments to study the effects of batch size on high performing strategies. Another set of experiments should look at incorporating costs in its decision process. Ultimately, keeping a strategy fixed, there are three interacting costs: compound availability and delivery costs, scheduling and screening time, and batch size which determines how many compounds to purchase. The difficulty is optimizing these three costs alongside optimizing a selection strategy. Conceivably, a single strategy that incorporates all these costs in its decision process may be considered.

In experiment 3, we saw that certain tasks are significantly harder and can result in non-random strategies performing similar to random strategies. However it is clear that, on average, certain strategies perform better than others across varying tasks. In experiment 4, we promoted a strategy that promotes novelty, namely CBWS_609. Objectively speaking, the performance of CBWS_609 on the prospective target in experiment 4 was a success; it finds ~4.8 times more hits than expected at random while only screening 4.25% of the unlabelled pool. An interesting observation in experiment 4, is that we see short periods of hit accumulation followed by long periods of exploration with no hits.

The experimental results highlighted two surprises that were counter-intuitive, namely, the

success of MABSelector and exploration heavy CBWS strategies. We expected an exclusively exploitive strategy to struggle with a large unlabelled pool, but this was not the case in a majority of tasks in experiment 3.1. MABSelector_exploitive exclusively exploits based on model hit predictions and performs the best overall on total hits in experiment 3.1. In contrast, we expected an exclusively exploratory strategy to struggle to find hits since it does not rely on hit predictions to guide its selection. CBWS exploratory strategies explore using a combination of model uncertainty scores and dissimilarity of compounds. However, we found that the supposedly exploratory CBWS_609 performs the best overall on total unique hits and performs very well on total hits in experiment 3.1. In experiment 3.2, where the initial plate has no actives, we see that fully exploitative strategies perform much worse than exploratory strategies. However, our followup investigation into exploratory configurations for CBWS revealed that model uncertainty was not performing as intended.

The main culprit for why exploratory strategies like CBWS_609 perform so well is that the max of the random forest hit predictions are less than 0.5 for a majority of the tasks. Thus, exploratory CBWS strategies like CBWS_609 are essentially performing exploitation with dissimilarity. The incorporation of dissimilarity in selection is one of the main differences between CBWS_609 and MABSelector_exploitive, and is likely reason for the unique hits performance boost. Based on these results, it would be beneficial for future work to consider giving more weight to dissimilarity in early iterations to capture novel hits, and then reducing the dissimilarity weight in later iterations to capture total hits.

The unintended exploitation mechanism of exploratory CBWS strategies requires fixing the uncertainty scores. It is unclear at the time of writing this thesis if the main reason is the nature of the random forest model, the small size of the training sets, or the hardness/skew of the learning task. Preliminary research in active learning literature on the nature of skewed datasets reveals that it is not a trivial problem in this domain [148, 149, 150, 151]. Due to this, future work should consider looking into this literature to address the uncertainty scoring.

To summarize, in regards to CBWS as a strategy, the problems and lessons highlighted here

have set forth the adjustments to be done for future work. In retrospect, it is assuring that we did not commit a large amount of resources to cover more of the CBWS configuration space. There were unforeseen issues that were revealed after conducting the various experiments. One glaring issue is the unintended behavior of exploratory CBWS strategies whereby they are in essence doing exploitation with dissimilarity. On the other hand, we were able to confirm that this dissimilarity is helpful for increasing the novelty of hits. Fixing the uncertainty score issue is priority as we want to assess its effectiveness in the IBS setting. Following this, one can explore incorporating cost considerations into the strategy. Finally, any developed strategy should be adapted over time as the developers perform more and more IBS experiments. It is advised that IBS experiments be set up with a goal in mind, and these goals in turn provide insights on how to adjust the strategy.

— 6 —

Conclusions and Future Work

In this thesis, we started by providing an overview of virtual screening and where it is situated in the drug discovery pipeline. This background is intended for new researchers interested in getting up to speed with the terms, application, types, and scenarios of virtual screening. Although we define virtual screening as any computational method used as an aide in the drug discovery pipeline, the focus in this thesis has been on using supervised machine learning methods in the initial hit-finding phase of the pipeline. With more drug discovery results and data being made available online through platforms like PubChem [5], it makes sense to leverage powerful machine learning models. However, it is important to consider the practical application of these models in hit-finding. Researchers need to consider available options according to what data is available, the quality of the data, the availability of prospective drugs, and the costs associated with purchasing and screening. Depending on the availability of data, we provide guidance on which options are applicable. Of particular interest to us is the application of virtual screening when there exists screening data for a target of interest. In such a case, we identify two viable options: one round screening (ORS) and iterative batched screening (IBS). In both these options, a strategy is used to select compounds from a prospective pool subject to a given budget. The main difference is that ORS only performs one round of screening whereas IBS performs multiple rounds. Thus, an effective IBS strategy might be one that dedicates some screening rounds to exploring the chemical space. We explore the practical effectiveness of machine learning models in an ORS case study on a protein-protein interaction target in chapters 2 and 3. In chapter 5, we formalize the IBS option and compare

strategies in a series of practical experiments, including a prospective target. The options when no data is available are even less clear. Here we provide one solution, called informer sets, which leverages screening data from related targets in the hope that it will lead to better selection of drugs for the target of interest. We provide a formal description of the informer set problem and propose various methods to solve it in chapter 4.

The ORS case study on PriA-SSB in chapters 2 and 3 highlighted the success of machine learning models in prioritizing compounds from a prospective pool. In chapter 2, the training data and prospective pool consisted of 72,423 and 22,434 compounds, respectively. A wide selection of machine learning models were compared using the training data to promote one final model for prospective selection. The final model was used to select 250 compounds from the prospective pool, finding 37 of the 54 hits. This prospective pool was smaller than the training set, and is relatively small compared to commercial pools. In chapter 3, we followed up on this success and applied the same process on PriA-SSB with two massive commercial pools: Aldrich Market Select (AMS) and Enamine REAL. By that time, the training data for PriA-SSB grew to 427,300 compounds which was used to promote one final prospective model. This final model was used to select 701 compounds from the 8,187,682 AMS compounds, finding 337 hits. Similarly, the same model was used to select 68 compounds from the 1,077,562,987 Enamine REAL compounds, finding 31 hits. For both prospective pools, the results far exceeded the expectation at random, and confirmed the effectiveness of machine learning models for one round screening. A main issue that we encountered with commercial pools is the large time and effort devoted to correspondence between our contact and the vendor. Future work in this regard should look into automating this process as much as possible.

In chapter 4, we extend the work done by Zhang et al. [2] on the informer set problem. We define an informer set method as performing two steps: selecting k informers to reveal for the missing target, then scoring the remaining non-informers. To select informers, the method makes use of compound-target data from other known targets. In Zhang et al., small datasets with no missing values were used like the PKIS1 dataset of 366 compounds and 224 targets. Followup

work by Yu et al. [86] proposed a method that can deal with missing values but its computational costs restrict it from being applied to large datasets. One goal of chapter 4 is to develop an informer set method that can scale to much larger datasets with missing values like 128-PCBA. We propose 21 methods that were adapted from fields with related problems like matrix completion, machine learning, and imputation algorithms. These methods were then compared on the PKIS1 dataset in a leave-one-target-out fashion. It was evident that making use of target-target information like protein sequence similarity boosted method performance. Furthermore, the CustomMC method showed competitive performance. This method requires no additional information beyond the bioactivity matrix, is amenable to handle missing values, and can be batched to handle large datasets. Thus, we applied CustomMC and two supervised learning methods to 10 PCBA targets of various sizes ranging from 9,000 to 74,000 compounds. The results were opposite to the PKIS1 experiment, with CustomMC performing the worst. This made it clear that PKIS1 and similarly small datasets, although useful for testing methods, should not be relied upon for future performance. Future work should focus on studying the relationship between the performance of a method and the dataset on which it is applied. As an example, establishing the reason why CustomMC performs so well on certain missing targets but not on others can help in deciding when and where to apply it. Furthermore, in our informer set formulation, we restricted ourselves to the two step scenario: select k informers, then score non-informers. The reason behind this was to imitate a two round screen. Perhaps performing multiple steps of screening might be more beneficial than this two step scenario in terms of hit-finding. There was further a focus on utilizing the compound-target data from the other known targets in selecting the informers. It may be the case that the missing target has no relation to the known targets, and it would be better to operate under that assumption by utilizing diverse selection of compounds as the first step.

In chapter 5, we explored the IBS option and provided a formal framework. The exploit-explore dilemma was discussed and the CBWS strategy was proposed to address it with elements of virtual screening in mind. Furthermore, various benchmarks were proposed based on reinforcement learning and random sampling. These strategies were compared and filtered through a series of

diverse experiments. With the final promoted CBWS strategy, an experiment was conducted on a prospective target with no training data for 50 iterations. The results on this prospective experiment were satisfying as it finds ~4.8 times more hits than expected at random while only screening 4.25% of the unlabelled pool. This experiment serves as an alternative to the informer set solution whereby we start with no training data and no knowledge of the target of interest. Lots of insight was gained from these experiments that point to promising avenues for future work. Addressing the unintended behavior of uncertainty in CBWS which led to exploitation should be the main focus. Despite this, we have seen that the addition of a dissimilarity term leads to finding more novel hits. Future work on incorporating costs of all forms into the strategy will be beneficial in practice, especially with massive commercial pools.

Finally, it is important to keep in mind the short term and long term goal of virtual screening. The short term of utilizing computational methods in finding hits based on preliminary screens can help filter down or triage the large pool of compounds. The long term goal is optimizing hits as we go down the drug discovery pipeline in order to find drugs that are safe and effective. With this in mind, one can conceive of employing virtual screening methods throughout the pipeline whenever possible to automate filtering. As an example, supervised learning models can be used to predict compound-target toxicity. In this thesis, we focused on the short term goal and formalized the available options based on the data available. The projects represent virtual screening case studies that involved in vitro screening at UW-Madison's Small Molecule Screening Facility (SMSF). As such, these projects were conducted by a multi-disciplinary team of chemists, statisticians, and computer scientists from various departments at UW-Madison. Much discussion, knowledge-sharing, and decisions took place through multiple focused meetings. Those with machine learning experience, but little chemical knowledge benefited from discussions with chemists, and vice versa. A diverse skill-set of computer science and chemistry can go a long way in understanding when, where, and how to apply virtual screening.

Bibliography

[1] G. Landrum, "Rdkit: Open-source cheminformatics software," 2018. [Online]. Available: https://github.com/rdkit

[2] H. Zhang, S. S. Ericksen, C.-p. Lee, G. E. Ananiev, N. Wlodarchak, P. Yu, J. C. Mitchell, A. Gitter, S. J. Wright, F. M. Hoffmann, S. A. Wildman, and M. A. Newton, "Predicting kinase inhibitors using bioactivity matrix derived informer sets," *PLOS Computational Biology*, vol. 15, no. 8, pp. 1–29, 08 2019. [Online]. Available: https://doi.org/10.1371/journal.pcbi.1006813

[3] D. H. Drewry, T. M. Willson, and W. J. Zuercher, "Seeding collaborations to advance kinase science with the GSK Published Kinase Inhibitor Set (PKIS)," *Curr Top Med Chem*, vol. 14, no. 3, pp. 340–342, 2014.

[4] B. Ramsundar, S. Kearnes, P. Riley, D. Webster, D. Konerding, and V. Pande, "Massively multitask networks for drug discovery," *arXiv preprint arXiv:1502.02072*, 2015.

[5] Y. Wang, S. H. Bryant, T. Cheng, J. Wang, A. Gindulyte, B. A. Shoemaker, P. A. Thiessen, S. He, and J. Zhang, "Pubchem bioassay: 2017 update," *Nucleic Acids Research*, vol. 45, no. D, pp. D955–D963, 2017.

[6] R. C. Mohs and N. H. Greig, "Drug discovery and development: Role of basic biological research," *Alzheimers Dement (N Y)*, vol. 3, no. 4, pp. 651–657, Nov 2017.

[7] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. Pande, "Moleculenet: a benchmark for molecular machine learning," *Chem. Sci.*, vol. 9, pp. 513–530, 2018. [Online]. Available: http://dx.doi.org/10.1039/C7SC02664A

[8] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, November 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[9] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: Lstm cells and network architectures," *Neural Computation*, vol. 31, no. 7, pp. 1235–1270, 2019, pMID: 31113301. [Online]. Available: https://doi.org/10.1162/neco_a_01199

[10] A. Lex, N. Gehlenborg, H. Strobelt, R. Vuillemot, and H. Pfister, "Upset: Visualization of intersecting sets," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 1983–1992, December 2014.

[11] A. Cichonska, B. Ravikumar, E. Parri, S. Timonen, T. Pahikkala, A. Airola, K. Wennerberg, J. Rousu, and T. Aittokallio, "Computational-experimental approach to drug-target interaction mapping: A case study on kinase inhibitors," *PLOS Computational Biology*, vol. 13, no. 8, pp. 1–28, 08 2017. [Online]. Available: https://doi.org/10.1371/journal.pcbi.1005678

[12] UniProt Consortium, "Uniprot: a worldwide hub of protein knowledge," *Nucleic acids research*, vol. 47, no. D1, pp. D506–D515, Jan 2019, 30395287[pmid]. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/30395287

[13] F. Madeira, Y. M. Park, J. Lee, N. Buso, T. Gur, N. Madhusoodanan, P. Basutkar, A. R. N. Tivey, S. C. Potter, R. D. Finn, and R. Lopez, "The embl-ebi search and sequence analysis tools apis in 2019," *Nucleic acids research*, vol. 47, no. W1, p. W636—W641, July 2019. [Online]. Available: https://europepmc.org/articles/PMC6602479

[14] E. Nelson, “Kinetics of drug absorption, distribution, metabolism, and excretion,” *Journal of Pharmaceutical Sciences*, vol. 50, no. 3, pp. 181–192, Mar 1961. [Online]. Available: https://doi.org/10.1002/jps.2600500302

[15] C. A. Lipinski, F. Lombardo, B. W. Dominy, and P. J. Feeney, “Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings,” *Advanced Drug Delivery Reviews*, vol. 23, no. 1, pp. 3 – 25, 1997, in Vitro Models for Selection of Development Candidates. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0169409X96004231

[16] C. A. Lipinski, “Drug-like properties and the causes of poor solubility and poor permeability,” *Journal of Pharmacological and Toxicological Methods*, vol. 44, no. 1, pp. 235 – 249, 2000, current Directions in Drug Discovery:A Review of Modern Techniques. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1056871900001076

[17] M. P. Doogue and T. M. Polasek, “The ABCD of clinical pharmacokinetics,” *Ther Adv Drug Saf*, vol. 4, no. 1, pp. 5–7, Feb 2013.

[18] S. M. Paul, D. S. Mytelka, C. T. Dunwiddie, C. C. Persinger, B. H. Munos, S. R. Lindborg, and A. L. Schacht, “How to improve r&d productivity: the pharmaceutical industry’s grand challenge,” *Nature Reviews Drug Discovery*, vol. 9, pp. 203 EP –, Feb 2010. [Online]. Available: https://doi.org/10.1038/nrd3078

[19] P. Kirkpatrick and C. Ellis, “Chemical space,” *Nature*, vol. 432, no. 7019, pp. 823–823, 2004. [Online]. Available: https://doi.org/10.1038/432823a

[20] T. Sterling and J. J. Irwin, “Zinc 15 - ligand discovery for everyone,” *Journal of Chemical Information and Modeling*, vol. 55, no. 11, pp. 2324–2337, Nov 2015. [Online]. Available: https://doi.org/10.1021/acs.jcim.5b00559

[21] T. Hoffmann and M. Gastreich, “The next level in chemical space navigation: going far beyond enumerable compound libraries,” *Drug Discovery Today*, vol. 24, no. 5, pp.

1148 – 1156, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1359644618304471

[22] "Enamine. enamine real database," accessed: 11 October 2019. [Online]. Available: https://enamine.net/library-synthesis/real-compounds/real-database

[23] R. Kiss, M. Sandor, and F. A. Szalai, "http://mcule.com: a public web service for drug discovery," *Journal of Cheminformatics*, vol. 4, no. Suppl 1, pp. P17–P17, May 2012, pMC3341220[pmcid]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3341220/

[24] "Sigma-aldrich. aldrich market select." [Online]. Available: https://www.sigmaaldrich.com/chemistry/chemistry-services/aldrich-market-select.html

[25] T. Chen, *A Practical Guide to Assay Development and High-Throughput Screening in Drug Discovery*. CRC Press, 12 2009.

[26] J.-H. Zhang, T. D. Y. Chung, and K. R. Oldenburg, "A simple statistical parameter for use in evaluation and validation of high throughput screening assays," *Journal of Biomolecular Screening*, vol. 4, no. 2, pp. 67–73, 1999, pMID: 10838414. [Online]. Available: https://doi.org/10.1177/108705719900400206

[27] N. Malo, J. A. Hanley, S. Cerquozzi, J. Pelletier, and R. Nadon, "Statistical practice in high-throughput screening data analysis," *Nature Biotechnology*, vol. 24, no. 2, pp. 167–175, 2006. [Online]. Available: https://doi.org/10.1038/nbt1186

[28] T. Tony Cai and W. Sun, "Optimal screening and discovery of sparse signals with applications to multistage high throughput studies," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 79, no. 1, pp. 197–223, 2017. [Online]. Available: https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/rssb.12171

[29] Q. S. Hanley, "The distribution of standard deviations applied to high throughput screening," *Scientific Reports*, vol. 9, no. 1, p. 1268, 2019. [Online]. Available: https://doi.org/10.1038/s41598-018-36722-4

[30] R. SNYDER, *Basic Concepts of the Dose-Response Relationship*, 1984, ch. 4, pp. 37–55. [Online]. Available: https://pubs.acs.org/doi/abs/10.1021/bk-1984-0239.ch004

[31] A. C. A. Roque, *Ligand-macromolecular interactions in drug discovery : methods and protocols*, ser. Methods in Molecular Biology. Humana Press, 2010.

[32] M. Ragoza, J. Hochuli, E. Idrobo, J. Sunseri, and D. R. Koes, "Protein–ligand scoring with convolutional neural networks," *Journal of Chemical Information and Modeling*, vol. 57, no. 4, pp. 942–957, 2017, pMID: 28368587. [Online]. Available: https://doi.org/10.1021/acs.jcim.6b00740

[33] P. Szymanski, M. Markowicz, and E. Mikiciuk-Olasik, "Adaptation of high-throughput screening in drug discovery-toxicological screening tests," *International journal of molecular sciences*, vol. 13, no. 1, pp. 427–452, Dec 2011. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/22312262

[34] S. Liu, M. Alnammi, S. S. Ericksen, A. F. Voter, G. E. Ananiev, J. L. Keck, F. M. Hoffmann, S. A. Wildman, and A. Gitter, "Practical model selection for prospective virtual screening," *Journal of Chemical Information and Modeling*, vol. 59, no. 1, pp. 282–293, 2019, pMID: 30500183. [Online]. Available: https://doi.org/10.1021/acs.jcim.8b00363

[35] O. Berger-Tal, J. Nathan, E. Meron, and D. Saltz, "The exploration-exploitation dilemma: A multidisciplinary framework," *PLOS ONE*, vol. 9, no. 4, pp. 1–8, 04 2014. [Online]. Available: https://doi.org/10.1371/journal.pone.0095693

[36] D. Reker and G. Schneider, "Active-learning strategies in computer-assisted drug discovery," *Drug Discovery Today*, vol. 20, no. 4, pp. 458 – 465, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1359644614004735

[37] G. B. Goh, C. Siegel, A. Vishnu, N. O. Hodas, and N. Baker, "Chemception: A deep neural network with minimal chemistry knowledge matches the performance of expert-developed qsar/qspr models," *arXiv preprint arXiv:1706.06689*, 2017.

[38] G. B. Goh, C. Siegel, A. Vishnu, N. Hodas, and N. Baker, "How much chemistry does a deep neural network need to know to make accurate predictions?" in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2018, pp. 1340–1349.

[39] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, pp. 193–202, 1980.

[40] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[41] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[42] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2020.

[43] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley, "Molecular graph convolutions: moving beyond fingerprints," *J. Comput. Aided Mol. Des.*, vol. 30, no. 8, pp. 595–608, 08 2016.

[44] H. Altae-Tran, B. Ramsundar, A. S. Pappu, and V. Pande, "Low data drug discovery with one-shot learning," *ACS central science*, vol. 3, no. 4, pp. 283–293, Apr 2017, pMC5408335[pmcid]. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/28470045

[45] J. Gomes, B. Ramsundar, E. N. Feinberg, and V. S. Pande, "Atomic convolutional networks for predicting protein-ligand binding affinity," *arXiv preprint arXiv:1703.10603*, 2017.

[46] E. N. Feinberg, D. Sur, Z. Wu, B. E. Husic, H. Mai, Y. Li, S. Sun, J. Yang, B. Ramsundar, and V. S. Pande, "Potentialnet for molecular property prediction," *ACS Central Science*, vol. 4, no. 11, pp. 1520–1530, 2018. [Online]. Available: https://doi.org/10.1021/acscentsci.8b00507

[47] D. Weininger, "Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules," *J. Chem. Inf. Comput. Sci.*, vol. 28, no. 1, pp. 31–36, Feb. 1988. [Online]. Available: http://dx.doi.org/10.1021/ci00057a005

[48] P. Ertl, R. Lewis, E. Martin, and V. Polyakov, "In silico generation of novel, drug-like chemical matter using the lstm neural network," *arXiv preprint arXiv:1712.07449*, 2017.

[49] A. Gupta, A. T. Müller, B. J. H. Huisman, J. A. Fuchs, P. Schneider, and G. Schneider, "Generative recurrent networks for de novo drug design," *Molecular Informatics*, vol. 37, no. 1-2, p. 1700111, 2018.

[50] R. Todeschini, V. Consonni, and P. Gramatica, "4.05 - chemometrics in qsar," in *Comprehensive Chemometrics*, S. D. Brown, R. Tauler, and B. Walczak, Eds. Oxford: Elsevier, 2009, pp. 129 – 172. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780444527011000077

[51] N. Salim, J. Holliday, and P. Willett, "Combination of fingerprint-based similarity coefficients using data fusion," *Journal of Chemical Information and Computer Sciences*, vol. 43, no. 2, pp. 435–442, 2003, pMID: 12653506. [Online]. Available: https://doi.org/10.1021/ci025596j

[52] G. M. Johnson, A. M.; Maggiora, *Concepts and applications of molecular similarity*, ser. Adaptive Computation and Machine Learning. John Wiley & Sons, 1990.

[53] Y. C. Martin, J. L. Kofron, and L. M. Traphagen, "Do structurally similar molecules have similar biological activity?" *Journal of Medicinal Chemistry*, vol. 45, no. 19, pp. 4350–4358, 2002, pMID: 12213076. [Online]. Available: https://doi.org/10.1021/jm020155c

[54] G. Maggiora, M. Vogt, D. Stumpfe, and J. Bajorath, "Molecular similarity in medicinal chemistry," *Journal of Medicinal Chemistry*, vol. 57, no. 8, pp. 3186–3204, 2014, pMID: 24151987. [Online]. Available: https://doi.org/10.1021/jm401411z

[55] D. Rogers and M. Hahn, "Extended-connectivity fingerprints," *Journal of Chemical Information and Modeling*, vol. 50, no. 5, pp. 742–754, 2010, pMID: 20426451. [Online]. Available: https://doi.org/10.1021/ci100050t

[56] Y. Wu and G. Wang, "Machine Learning Based Toxicity Prediction: From Chemical Structural Description to Transcriptome Analysis," *Int J Mol Sci*, vol. 19, no. 8, Aug 2018.

[57] A. F. Voter, K. A. Manthei, and J. L. Keck, "A high-throughput screening strategy to identify protein-protein interaction inhibitors that block the fanconi anemia dna repair pathway," *Journal of Biomolecular Screening*, vol. 21, no. 6, pp. 626–633, 2016.

[58] A. F. Voter, M. P. Killoran, G. E. Ananiev, S. A. Wildman, F. M. Hoffmann, and J. L. Keck, "A high-throughput screening strategy to identify inhibitors of ssb protein–protein interactions in an academic screening facility," *SLAS DISCOVERY: Advancing Life Sciences R&D*, pp. 94–101, 2017.

[59] J. B. Baell and G. A. Holloway, "New substructure filters for removal of pan assay interference compounds (pains) from screening libraries and for their exclusion in bioassays," *Journal of Medicinal Chemistry*, vol. 53, no. 7, pp. 2719–2740, 2010.

[60] B. Ramsundar, B. Liu, Z. Wu, A. Verras, M. Tudor, R. P. Sheridan, and V. Pande, "Is multitask deep learning practical for pharma?" *Journal of Chemical Information and Modeling*, vol. 57, no. 8, pp. 2068–2076, 2017.

[61] G. E. Dahl, N. Jaitly, and R. Salakhutdinov, "Multi-task neural networks for qsar predictions," *arXiv preprint arXiv:1406.1231*, 2014.

[62] S. Kearnes, B. Goldman, and V. Pande, "Modeling industrial admet data with multitask networks," *arXiv preprint arXiv:1606.08793*, 2016.

[63] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[64] S. J. Swamidass, C.-A. Azencott, T.-W. Lin, H. Gramajo, S.-C. Tsai, and P. Baldi, "Influence relevance voting: an accurate and interpretable virtual high throughput screening method," *Journal of Chemical Information and Modeling*, vol. 49, no. 4, pp. 756–766, 2009.

[65] S. Jastrzębski, D. Leśniak, and W. M. Czarnecki, "Learning to smile(s)," *arXiv preprint arXiv:1602.06289*, 2016.

[66] S. S. Ericksen, H. Wu, H. Zhang, L. A. Michael, M. A. Newton, F. M. Hoffmann, and S. A. Wildman, "Machine learning consensus scoring improves performance across targets in structure-based virtual screening," *Journal of Chemical Information and Modeling*, vol. 57, no. 7, pp. 1579–1590, 2017.

[67] M. Lau, "Dtk: Dunnett-tukey-kramer pairwise multiple comparison test adjusted for unequal variances and unequal sample sizes," *R package*, 01 2013.

[68] C. W. Dunnett, "Pairwise multiple comparisons in the unequal variance case," *Journal of the American Statistical Association*, vol. 75, no. 372, pp. 796–800, December 1980. [Online]. Available: https://amstat.tandfonline.com/doi/abs/10.1080/01621459.1980.10477552

[69] J. H. W. Jr., "Hierarchical grouping to optimize an objective function," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 236–244, 1963. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/01621459.1963.10500845

[70] A. Nicholls, "What do we know and when do we know it?" *Journal of Computer-Aided Molecular Design*, vol. 22, no. 3-4, pp. 239–255, March 2008. [Online]. Available: https://link.springer.com/article/10.1007/s10822-008-9170-2

[71] E. B. Lenselink, N. t. Dijke, B. Bongers, G. Papadatos, H. W. T. v. Vlijmen, W. Kowalczyk, A. P. IJzerman, and G. J. P. v. Westen, "Beyond the hype: Deep neural

networks outperform established methods using a chembl bioactivity benchmark set," *Journal of Cheminformatics*, vol. 9, no. 1, p. 45, December 2017. [Online]. Available: https://link.springer.com/article/10.1186/s13321-017-0232-0

[72] A. Korotcov, V. Tkachenko, D. P. Russo, and S. Ekins, "Comparison of deep learning with multiple machine learning methods and metrics using diverse drug discovery data sets," *Molecular Pharmaceutics*, vol. 14, no. 12, pp. 4462–4475, December 2017. [Online]. Available: https://doi.org/10.1021/acs.molpharmaceut.7b00578

[73] P. S. Kutchukian, L. Warren, B. C. Magliaro, A. Amoss, J. A. Cassaday, G. O'Donnell, B. Squadroni, P. Zuck, D. Pascarella, J. C. Culberson, A. J. Cooke, D. Hurzy, K.-A. S. Schlegel, F. Thomson, E. N. Johnson, V. N. Uebele, J. D. Hermes, S. Parmentier-Batteur, and M. Finley, "Iterative focused screening with biological fingerprints identifies selective asc-1 inhibitors distinct from traditional high throughput screening," *ACS Chemical Biology*, vol. 12, no. 2, pp. 519–527, Feb 2017. [Online]. Available: https://doi.org/10.1021/acschembio.6b00913

[74] J. Lyu, S. Wang, T. E. Balius, I. Singh, A. Levit, Y. S. Moroz, M. J. O'Meara, T. Che, E. Algaa, K. Tolmachova, A. A. Tolmachev, B. K. Shoichet, B. L. Roth, and J. J. Irwin, "Ultra-large library docking for discovering new chemotypes," *Nature*, vol. 566, no. 7743, pp. 224–229, 2019. [Online]. Available: https://doi.org/10.1038/s41586-019-0917-9

[75] K. McCloskey, E. A. Sigel, S. Kearnes, L. Xue, X. Tian, D. Moccia, D. Gikunju, S. Bazzaz, B. Chan, M. A. Clark, J. W. Cuozzo, M.-A. Guié, J. P. Guilinger, C. Huguet, C. D. Hupp, A. D. Keefe, C. J. Mulhern, Y. Zhang, and P. Riley, "Machine learning on dna-encoded libraries: A new paradigm for hit finding," *Journal of Medicinal Chemistry*, Jun 2020. [Online]. Available: https://doi.org/10.1021/acs.jmedchem.0c00452

[76] M. A. Clark, R. A. Acharya, C. C. Arico-Muendel, S. L. Belyanskaya, D. R. Benjamin, N. R. Carlson, P. A. Centrella, C. H. Chiu, S. P. Creaser, J. W. Cuozzo, C. P. Davie,

Y. Ding, G. J. Franklin, K. D. Franzen, M. L. Gefter, S. P. Hale, N. J. V. Hansen, D. I. Israel, J. Jiang, M. J. Kavarana, M. S. Kelley, C. S. Kollmann, F. Li, K. Lind, S. Mataruse, P. F. Medeiros, J. A. Messer, P. Myers, H. O'Keefe, M. C. Oliff, C. E. Rise, A. L. Satz, S. R. Skinner, J. L. Svendsen, L. Tang, K. van Vloten, R. W. Wagner, G. Yao, B. Zhao, and B. A. Morgan, "Design, synthesis and selection of dna-encoded small-molecule libraries," *Nature Chemical Biology*, vol. 5, no. 9, pp. 647–654, 2009. [Online]. Available: https://doi.org/10.1038/nchembio.211

[77] J. M. Stokes, K. Yang, K. Swanson, W. Jin, A. Cubillos-Ruiz, N. M. Donghia, C. R. MacNair, S. French, L. A. Carfrae, Z. Bloom-Ackerman, V. M. Tran, A. Chiappino-Pepe, A. H. Badran, I. W. Andrews, E. J. Chory, G. M. Church, E. D. Brown, T. S. Jaakkola, R. Barzilay, and J. J. Collins, "A deep learning approach to antibiotic discovery," *Cell*, vol. 180, no. 4, pp. 688–702.e13, Feb 2020. [Online]. Available: https://doi.org/10.1016/j.cell.2020.01.021

[78] C. Gorgulla, A. Boeszoermenyi, Z.-F. Wang, P. D. Fischer, P. W. Coote, K. M. Padmanabha Das, Y. S. Malets, D. S. Radchenko, Y. S. Moroz, D. A. Scott, K. Fackeldey, M. Hoffmann, I. Iavniuk, G. Wagner, and H. Arthanari, "An open-source drug discovery platform enables ultra-large virtual screens," *Nature*, vol. 580, no. 7805, pp. 663–668, Apr 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2117-z

[79] D. Rogers and M. Hahn, "Extended-connectivity fingerprints," *Journal of Chemical Information and Modeling*, vol. 50, no. 5, pp. 742–754, May 2010. [Online]. Available: https://doi.org/10.1021/ci100050t

[80] R. Taylor, "Simulation analysis of experimental design strategies for screening random compounds as potential new drugs and agrochemicals," *Journal of Chemical Information and Computer Sciences*, vol. 35, no. 1, pp. 59–67, 1995. [Online]. Available: https://pubs.acs.org/doi/abs/10.1021/ci00023a009

[81] D. Butina, "Unsupervised data base clustering based on daylight's fingerprint and tanimoto similarity: A fast and automated way to cluster small and large data sets," *Journal of*

Chemical Information and Computer Sciences, vol. 39, no. 4, pp. 747–750, 1999. [Online]. Available: https://doi.org/10.1021/ci9803381

[82] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 785–794. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939785

[83] B. Ramsundar, P. Eastman, P. Walters, V. Pande, K. Leswing, and Z. Wu, *Deep Learning for the Life Sciences*. O'Reilly Media, 2019, https://www.amazon.com/Deep-Learning-Life-Sciences-Microscopy/dp/1492039837.

[84] P. Walters, "rd_filters," Apr. 2020. [Online]. Available: https://github.com/PatWalters/rd_filters

[85] D. Mendez, A. Gaulton, A. P. Bento, J. Chambers, M. De Veij, E. Felix, M. P. Magarinos, J. F. Mosquera, P. Mutowo, M. Nowotka, M. Gordillo-Maranon, F. Hunter, L. Junco, G. Mugumbate, M. Rodriguez-Lopez, F. Atkinson, N. Bosc, C. J. Radoux, A. Segura-Cabrera, A. Hersey, and A. R. Leach, "Chembl: towards direct deposition of bioassay data," *Nucleic Acids Res.*, vol. 47, no. D1, pp. D930–D940, Jan 2019.

[86] P. Yu, S. S. Ericksen, A. Gitter, and M. A. Newton, "Bayes optimal informer sets for early-stage drug discovery," *arXiv preprint arXiv:2011.06122*, 2020.

[87] S. Paricharak, A. P. IJzerman, J. L. Jenkins, A. Bender, and F. Nigsch, "Data-driven derivation of an "informer compound set" for improved selection of active compounds in high-throughput screening," *Journal of Chemical Information and Modeling*, vol. 56, no. 9, pp. 1622–1630, 2016, pMID: 27487177. [Online]. Available: https://doi.org/10.1021/acs.jcim.6b00244

[88] T. Qiu, J. Qiu, J. Feng, D. Wu, Y. Yang, K. Tang, Z. Cao, and R. Zhu, "The recent progress in proteochemometric modelling: focusing on target descriptors, cross-term descriptors

and application scope," *Briefings in Bioinformatics*, vol. 18, no. 1, pp. 125–136, 02 2016. [Online]. Available: https://doi.org/10.1093/bib/bbw004

[89] M. Wójcikowski, M. Kukiełka, M. M. Stepniewska-Dziubinska, and P. Siedlecki, "Development of a protein–ligand extended connectivity (PLEC) fingerprint and its application for binding affinity predictions," *Bioinformatics*, vol. 35, no. 8, pp. 1334–1341, 09 2018. [Online]. Available: https://doi.org/10.1093/bioinformatics/bty757

[90] A. Cichonska, B. Ravikumar, R. J. Allaway, S. Park, F. Wan, O. Isayev, S. Li, M. Mason, A. Lamb, Z. Tanoli, M. Jeon, S. Kim, M. Popova, S. Capuzzi, J. Zeng, K. Dang, G. Koytiger, J. Kang, C. I. Wells, T. M. Willson, T. I. Oprea, A. Schlessinger, D. H. Drewry, G. Stolovitzky, K. Wennerberg, J. Guinney, and T. Aittokallio, "Crowdsourced mapping extends the target space of kinase inhibitors," *bioRxiv*, 2020. [Online]. Available: https://www.biorxiv.org/content/early/2020/02/11/2019.12.31.891812

[91] D. H. Drewry, C. I. Wells, D. M. Andrews, R. Angell, H. Al-Ali, A. D. Axtman, S. J. Capuzzi, J. M. Elkins, P. Ettmayer, M. Frederiksen, O. Gileadi, N. Gray, A. Hooper, S. Knapp, S. Laufer, U. Luecking, M. Michaelides, S. Muller, E. Muratov, R. A. Denny, K. S. Saikatendu, D. K. Treiber, W. J. Zuercher, and T. M. Willson, "Progress towards a public chemogenomic set for protein kinases and a call for contributions," *PLoS ONE*, vol. 12, no. 8, 2017.

[92] W. Yang, J. Soares, P. Greninger, E. J. Edelman, H. Lightfoot, S. Forbes, N. Bindal, D. Beare, J. A. Smith, I. R. Thompson, S. Ramaswamy, P. A. Futreal, D. A. Haber, M. R. Stratton, C. Benes, U. McDermott, and M. J. Garnett, "Genomics of drug sensitivity in cancer (gdsc): a resource for therapeutic biomarker discovery in cancer cells," *Nucleic acids research*, vol. 41, no. Database issue, pp. D955–D961, Jan 2013, 23180760[pmid]. [Online]. Available: https://doi.org/10.1093/nar/gks1111

[93] X. Zhu, A. Singla, S. Zilles, and A. N. Rafferty, "An overview of machine teaching," *CoRR*, vol. abs/1801.05927, 2018. [Online]. Available: http://arxiv.org/abs/1801.05927

[94] S. Mei and X. Zhu, "Some submodular data-poisoning attacks on machine learners," *Computer Science Tech Report 1822*, 2015. [Online]. Available: https://minds.wisconsin.edu/handle/1793/76118

[95] C. Coleman, C. Yeh, S. Mussmann, B. Mirzasoleiman, P. Bailis, P. Liang, J. Leskovec, and M. Zaharia, "Selection via proxy: Efficient data selection for deep learning," *CoRR*, vol. abs/1906.11829, 2019. [Online]. Available: http://arxiv.org/abs/1906.11829

[96] A. D. Prjibelski, A. I. Korobeynikov, and A. L. Lapidus, "Sequence analysis," in *Encyclopedia of Bioinformatics and Computational Biology*, S. Ranganathan, M. Gribskov, K. Nakai, and C. Schönbach, Eds. Oxford: Academic Press, 2019, pp. 292 – 322. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128096338201064

[97] M. Gönen, "Predicting drug–target interactions from chemical and genomic kernels using Bayesian matrix factorization," *Bioinformatics*, vol. 28, no. 18, pp. 2304–2310, 06 2012. [Online]. Available: https://doi.org/10.1093/bioinformatics/bts360

[98] T. van Laarhoven and E. Marchiori, "Predicting drug-target interactions for new drug compounds using a weighted nearest neighbor profile," *PloS one*, vol. 8, no. 6, pp. e66 952–e66 952, Jun 2013, pMC3694117[pmcid]. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/23840562

[99] D. Karasev, B. Sobolev, A. Lagunin, D. Filimonov, and V. Poroikov, "Prediction of protein-ligand interaction based on the positional similarity scores derived from amino acid sequences," *International journal of molecular sciences*, vol. 21, no. 1, p. 24, Dec 2019, pMC6981593[pmcid]. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/31861473

[100] A. Rubinsteyn, S. Feldman, T. O'Donnell, and B. Beaulieu-Jones, "hammerlab/fancyimpute: Version 0.2.0," Sep. 2017. [Online]. Available: https://doi.org/10.5281/zenodo.886614

[101] E. J. Candès and B. Recht, "Exact matrix completion via convex optimization," *Foundations of Computational Mathematics*, vol. 9, no. 6, p. 717, Apr 2009. [Online]. Available: https://doi.org/10.1007/s10208-009-9045-5

[102] J. Yang, L. You, K. Li, and J. Yang, "Low-rank matrix completion against missing rows and columns with separable 2-d sparsity priors," in *2017 IEEE International Conference on Image Processing (ICIP)*, 2017, pp. 3520–3524.

[103] R. Mazumder, T. Hastie, and R. Tibshirani, "Spectral regularization algorithms for learning large incomplete matrices," *J. Mach. Learn. Res.*, vol. 11, p. 2287–2322, Aug. 2010.

[104] R. Mazumder and T. Hastie, "softimpute: Matrix completion via iterative softthresholded svd," *R package*, 2013. [Online]. Available: http://CRAN.R-project.org/package=softImpute

[105] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[106] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[107] I. Tsang, J. Kwok, and P.-M. Cheung, "Core vector machines: Fast svm training on very large data sets." *Journal of Machine Learning Research*, vol. 6, pp. 363–392, 04 2005.

[108] S. Har-Peled and A. Kushal, "Smaller coresets for k-median and k-means clustering," *Discrete & Computational Geometry*, vol. 37, no. 1, pp. 3–19, Jan 2007. [Online]. Available: https://doi.org/10.1007/s00454-006-1271-x

[109] O. Sener and S. Savarese, "Active learning for convolutional neural networks: A core-set approach," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=H1aIuk-RW

[110] B. Settles, "Active learning literature survey," 2010. [Online]. Available: http://burrsettles.com/pub/settles.activelearning.pdf

[111] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[112] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[113] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[114] G. W. Bemis and M. A. Murcko, "The properties of known drugs. 1. molecular frameworks," *Journal of Medicinal Chemistry*, vol. 39, no. 15, pp. 2887–2893, 1996, pMID: 8709122. [Online]. Available: https://doi.org/10.1021/jm9602928

[115] S. Paricharak, A. P. IJzerman, A. Bender, and F. Nigsch, "Analysis of iterative screening with stepwise compound selection based on novartis in-house hts data," *ACS Chemical Biology*, vol. 11, no. 5, pp. 1255–1264, 2016, pMID: 26878899. [Online]. Available: https://doi.org/10.1021/acschembio.6b00029

[116] M. K. Warmuth, G. Rätsch, M. Mathieson, J. Liao, and C. Lemmen, "Active learning in the drug discovery process," in *Advances in Neural information processing systems*, 2002, pp. 1449–1456.

[117] Y. Fujiwara, Y. Yamashita, T. Osoda, M. Asogawa, C. Fukushima, M. Asao, H. Shimadzu, K. Nakao, and R. Shimizu, "Virtual screening system for finding structurally diverse hits

by active learning," *Journal of Chemical Information and Modeling*, vol. 48, no. 4, pp. 930–940, 2008, pMID: 18351729. [Online]. Available: https://doi.org/10.1021/ci700085q

[118] K. De Grave, J. Ramon, and L. De Raedt, *Active Learning for High Throughput Screening*, ser. Lecture Notes in Artificial Intelligence. Springer-Verlag Berlin Heidelberg, 10 2008.

[119] M. Ahmadi, M. Vogt, P. Iyer, J. Bajorath, and H. Fröhlich, "Predicting potent compounds via model-based global optimization," *Journal of Chemical Information and Modeling*, vol. 53, no. 3, pp. 553–559, 2013, pMID: 23363236. [Online]. Available: https://doi.org/10.1021/ci3004682

[120] B. Desai, K. Dixon, E. Farrant, Q. Feng, K. R. Gibson, W. P. van Hoorn, J. Mills, T. Morgan, D. M. Parry, M. K. Ramjee, C. N. Selway, G. J. Tarver, G. Whitlock, and A. G. Wright, "Rapid discovery of a novel series of abl kinase inhibitors by application of an integrated microfluidic synthesis and screening platform," *Journal of Medicinal Chemistry*, vol. 56, no. 7, pp. 3033–3047, 2013, pMID: 23441572. [Online]. Available: https://doi.org/10.1021/jm400099d

[121] F. Svensson, U. Norinder, and A. Bender, "Improving screening efficiency through iterative screening using docking and conformal prediction," *Journal of Chemical Information and Modeling*, vol. 57, no. 3, pp. 439–444, 2017, pMID: 28195474. [Online]. Available: https://doi.org/10.1021/acs.jcim.6b00532

[122] U. Norinder, L. Carlsson, S. Boyer, and M. Eklund, "Introducing conformal prediction in predictive modeling. a transparent and flexible alternative to applicability domain determination," *Journal of Chemical Information and Modeling*, vol. 54, no. 6, pp. 1596–1603, 2014, pMID: 24797111. [Online]. Available: https://doi.org/10.1021/ci5001168

[123] I. Cortés-Ciriano, N. C. Firth, A. Bender, and O. Watson, "Discovering highly potent molecules from an initial set of inactives using iterative screening," *Journal of Chemical*

Information and Modeling, vol. 58, no. 9, pp. 2000–2014, 2018, pMID: 30130102. [Online]. Available: https://doi.org/10.1021/acs.jcim.8b00376

[124] T. Miyao and K. Funatsu, "Iterative screening methods for identification of chemical compounds with specific values of various properties," *Journal of Chemical Information and Modeling*, vol. 0, no. 0, p. null, 2019. [Online]. Available: https://doi.org/10.1021/acs.jcim.9b00093

[125] S. Wold, M. Sjöström, and L. Eriksson, "Pls-regression: a basic tool of chemometrics," *Chemometrics and Intelligent Laboratory Systems*, vol. 58, no. 2, pp. 109 – 130, 2001, pLS Methods. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0169743901001551

[126] K. Hasegawa, Y. Miyashita, and K. Funatsu, "Ga strategy for variable selection in qsar studies: Ga-based pls analysis of calcium channel antagonists," *Journal of Chemical Information and Computer Sciences*, vol. 37, no. 2, pp. 306–310, 1997, pMID: 9157101. [Online]. Available: https://doi.org/10.1021/ci960047x

[127] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *J. of Global Optimization*, vol. 13, no. 4, pp. 455–492, Dec. 1998. [Online]. Available: https://doi.org/10.1023/A:1008306431147

[128] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*, ser. Adaptive Computation and Machine Learning. Cambridge, MA, USA: MIT Press, Jan. 2006.

[129] J. González, Z. Dai, P. Hennig, and N. Lawrence, "Batch bayesian optimization via local penalization," in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (AISTATS)*, ser. JMLR Workshop and Conference Proceedings, vol. 51, May 2016, pp. 648–657. [Online]. Available: http://jmlr.org/proceedings/papers/v51/gonzalez16a.pdf

[130] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, no. 3, pp. 199–222, Aug. 2004. [Online]. Available: https://doi.org/10.1023/B:STCO.0000035301.49549.88

[131] R. Buendia, T. Kogej, O. Engkvist, L. Carlsson, H. Linusson, U. Johansson, P. Toccaceli, and E. Ahlberg, "Accurate hit estimation for iterative screening using venn–abers predictors," *Journal of Chemical Information and Modeling*, vol. 59, no. 3, pp. 1230–1237, 2019, pMID: 30726080. [Online]. Available: https://doi.org/10.1021/acs.jcim.8b00724

[132] V. Vovk and I. Petej, "Venn-abers predictors," in *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, ser. UAI'14. Arlington, Virginia, USA: AUAI Press, 2014, p. 829–838.

[133] M. Maciejewski, A. M. Wassermann, M. Glick, and E. Lounkine, "Experimental design strategy: Weak reinforcement leads to increased hit rates and enhanced chemical diversity," *Journal of Chemical Information and Modeling*, vol. 55, no. 5, pp. 956–962, 2015, pMID: 25915687. [Online]. Available: https://doi.org/10.1021/acs.jcim.5b00054

[134] J. M. Jansen, G. De Pascale, S. Fong, M. Lindvall, H. E. Moser, K. Pfister, B. Warne, and C. Wartchow, "Biased complement diversity selection for effective exploration of chemical space in hit-finding campaigns," *Journal of Chemical Information and Modeling*, vol. 59, no. 5, pp. 1709–1714, 2019, pMID: 30943027. [Online]. Available: https://doi.org/10.1021/acs.jcim.9b00048

[135] W. R. Thompson, "ON THE LIKELIHOOD THAT ONE UNKNOWN PROBABILITY EXCEEDS ANOTHER IN VIEW OF THE EVIDENCE OF TWO SAMPLES," *Biometrika*, vol. 25, no. 3-4, pp. 285–294, 12 1933.

[136] A. Slivkins, "Introduction to multi-armed bandits," *Foundations and Trends® in Machine Learning*, vol. 12, no. 1-2, pp. 1–286, 2019. [Online]. Available: http://dx.doi.org/10.1561/2200000068

[137] K.-S. Jun, K. Jamieson, R. Nowak, and X. Zhu, "Top arm identification in multi-armed bandits with batch arm pulls," ser. Proceedings of Machine Learning Research, A. Gretton and C. C. Robert, Eds., vol. 51. Cadiz, Spain: PMLR, 09–11 May 2016, pp. 139–148.

[138] K. Madhawa and T. Murata, "A multi-armed bandit approach for exploring partially observed networks," *Applied Network Science*, vol. 4, no. 1, p. 26, May 2019. [Online]. Available: https://doi.org/10.1007/s41109-019-0145-0

[139] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, p. 1–30, Dec. 2006.

[140] S. García and F. Herrera, "An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons," *Journal of Machine Learning Research - JMLR*, vol. 9, 12 2008.

[141] S. García, A. Fernández, J. Luengo, and F. Herrera, "Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power," *Inf. Sci.*, vol. 180, no. 10, p. 2044–2064, May 2010. [Online]. Available: https://doi.org/10.1016/j.ins.2009.12.010

[142] B. Trawiński, M. Smundefinedtek, Z. Telec, and T. Lasota, "Nonparametric statistical analysis for multiple comparison of machine learning regression algorithms," *Int. J. Appl. Math. Comput. Sci.*, vol. 22, no. 4, p. 867–881, Dec. 2012. [Online]. Available: https://doi.org/10.2478/v10006-012-0064-z

[143] G. Chao and S. Sun, "Consensus and complementarity based maximum entropy discrimination for multi-view classification," *Information Sciences*, vol. 367-368, pp. 296 – 310, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S002002551630411X

[144] M. De Gregorio and M. Giordano, "An experimental evaluation of weightless neural networks for multi-class classification," *Applied Soft Computing*, vol. 72, pp. 338 – 354, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S156849461830440X

[145] F. Hafiz, A. Swain, C. Naik, and N. Patel, "Efficient feature selection of power quality events using two dimensional (2d) particle swarms," *Applied Soft Computing*, vol. 81, p. 105498, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1568494619302686

[146] Z. Liu and J. Liu, "A robust time series prediction method based on empirical mode decomposition and high-order fuzzy cognitive maps," *Knowledge-Based Systems*, vol. 203, p. 106105, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950705120303713

[147] T.-t. Liu, T. Yang, M.-n. Gao, K.-x. Chen, S. Yang, K.-q. Yu, and H.-l. Jiang, "The inhibitory mechanism of aurintricarboxylic acid targeting serine/threonine phosphatase stp1 in staphylococcus aureus: insights from molecular dynamics simulations," *Acta Pharmacologica Sinica*, vol. 40, no. 6, pp. 850–858, Jun 2019. [Online]. Available: https://doi.org/10.1038/s41401-019-0216-x

[148] J. Attenberg and S. Ertekin, *Class Imbalance and Active Learning*. John Wiley & Sons, Ltd, 2013, ch. 6, pp. 101–149.

[149] H. Yu, X. Yang, S. Zheng, and C. Sun, "Active learning from imbalanced data: A solution of online weighted extreme learning machine," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 4, pp. 1088–1103, 2019.

[150] C. H. Lin, Mausam, and D. S. Weld, "Active learning with unbalanced classes and example-generation queries," in *HCOMP*, 2018.

[151] U. Aggarwal, A. Popescu, and C. Hudelot, "Active learning for imbalanced datasets," in *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2020, pp. 1417–1426.

Appendix A

Experiment 3.1: Per Task Boxplots

A.1 Experiment 3.1: Per Task Total Hits Boxplots

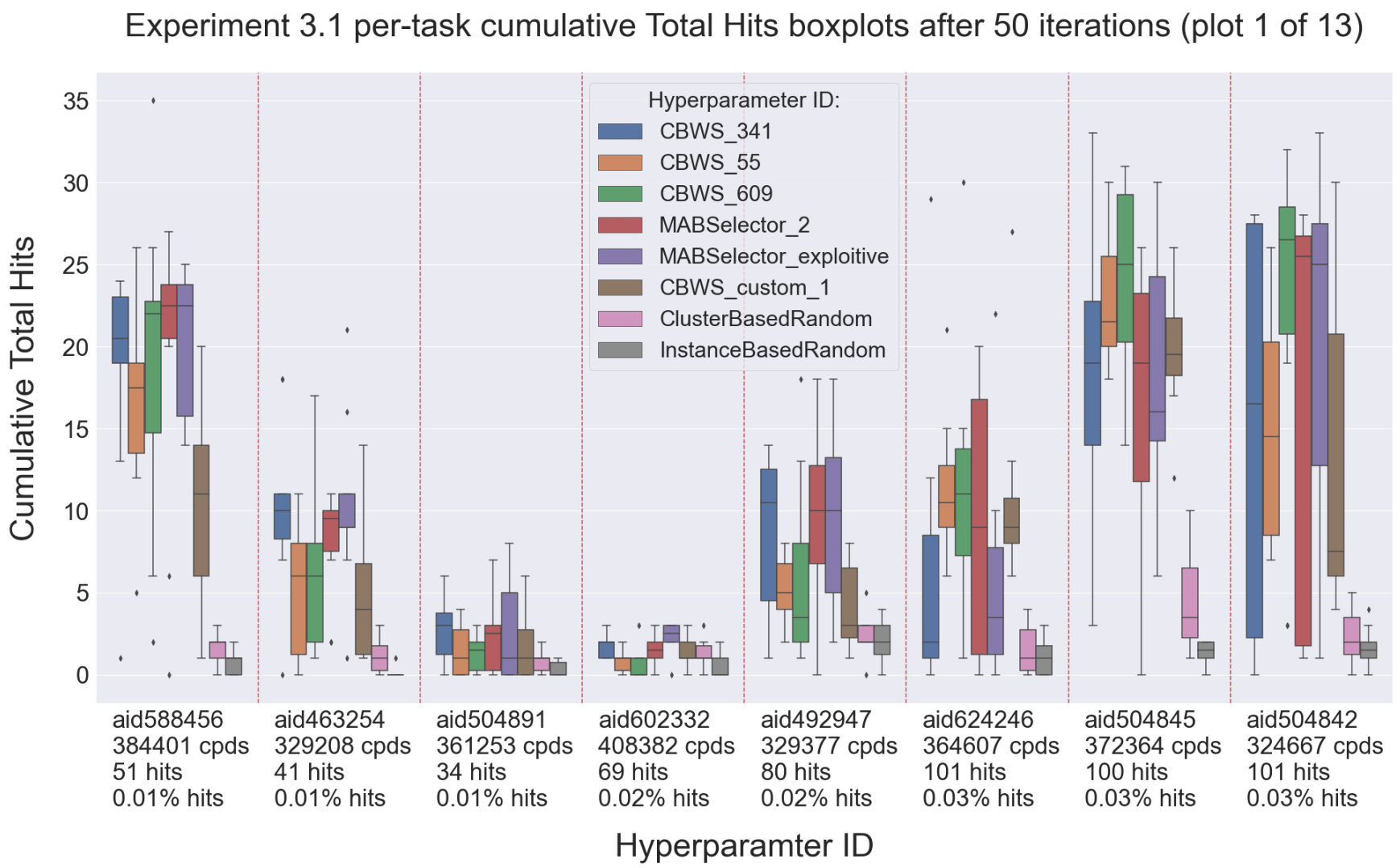


Figure A.1: Experiment 3.1 per task **Total Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(1 of 13 cont.)*

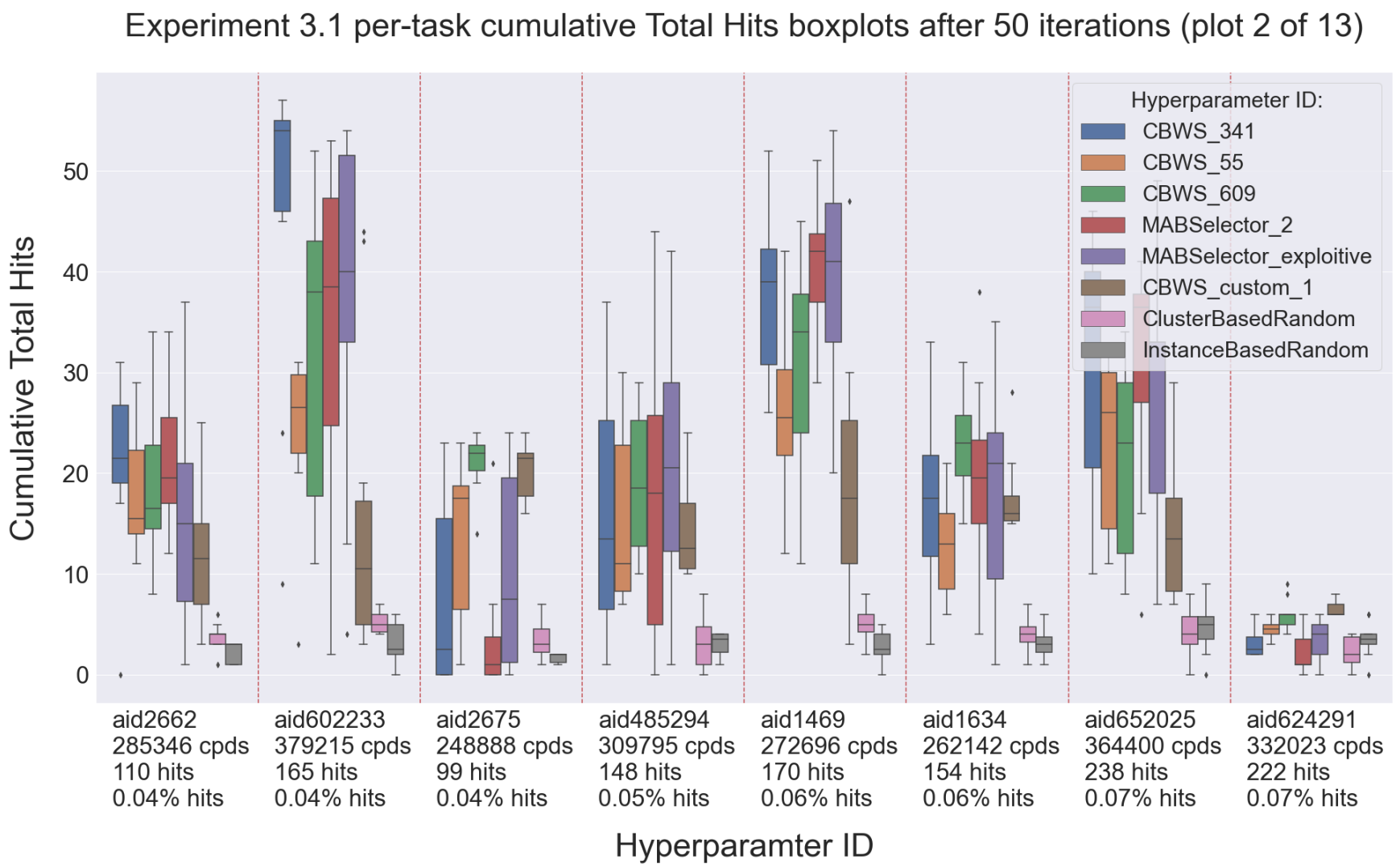


Figure A.1: Experiment 3.1 per task **Total Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(2 of 13 cont.)*

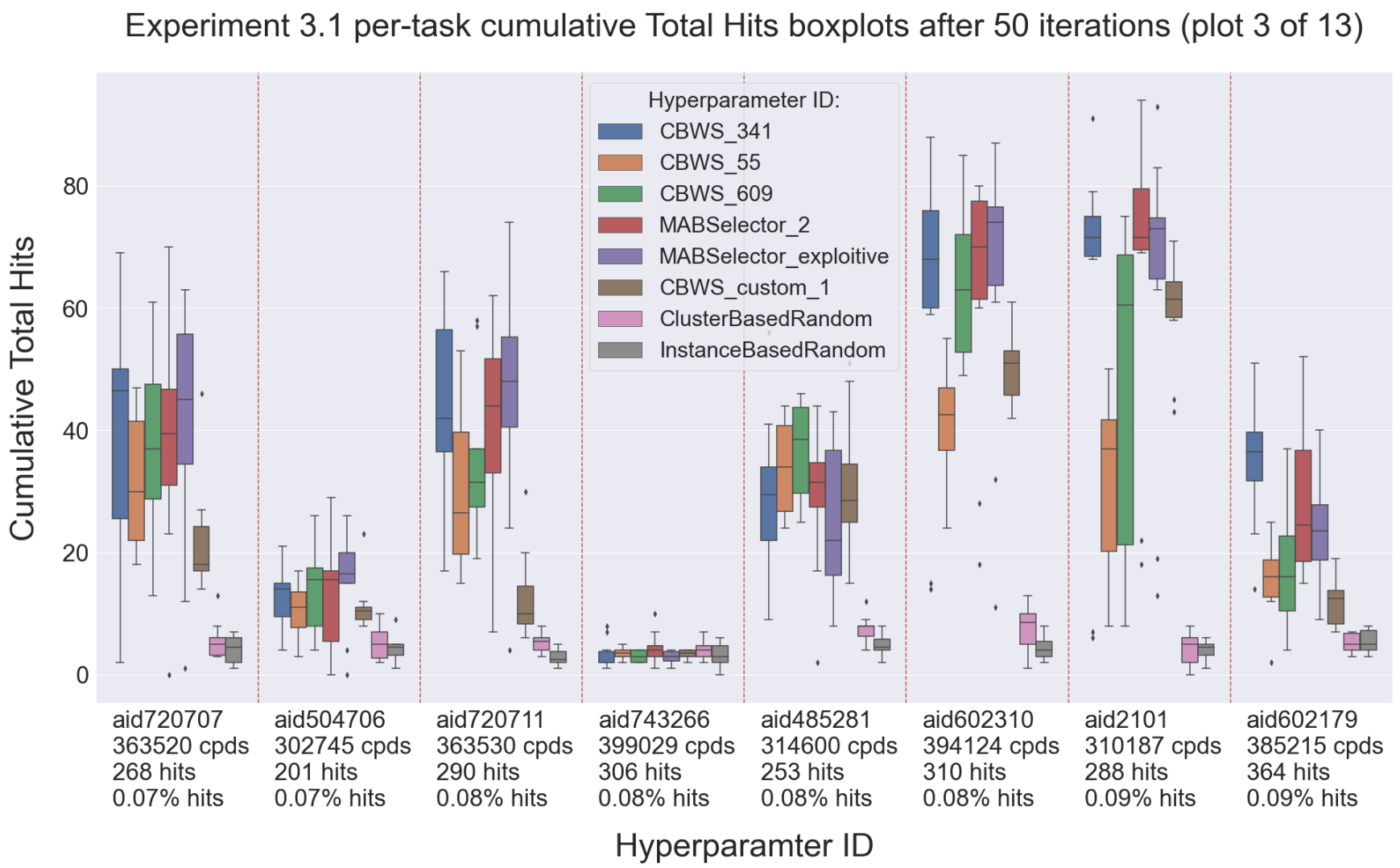


Figure A.1: Experiment 3.1 per task **Total Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(3 of 13 cont.)*

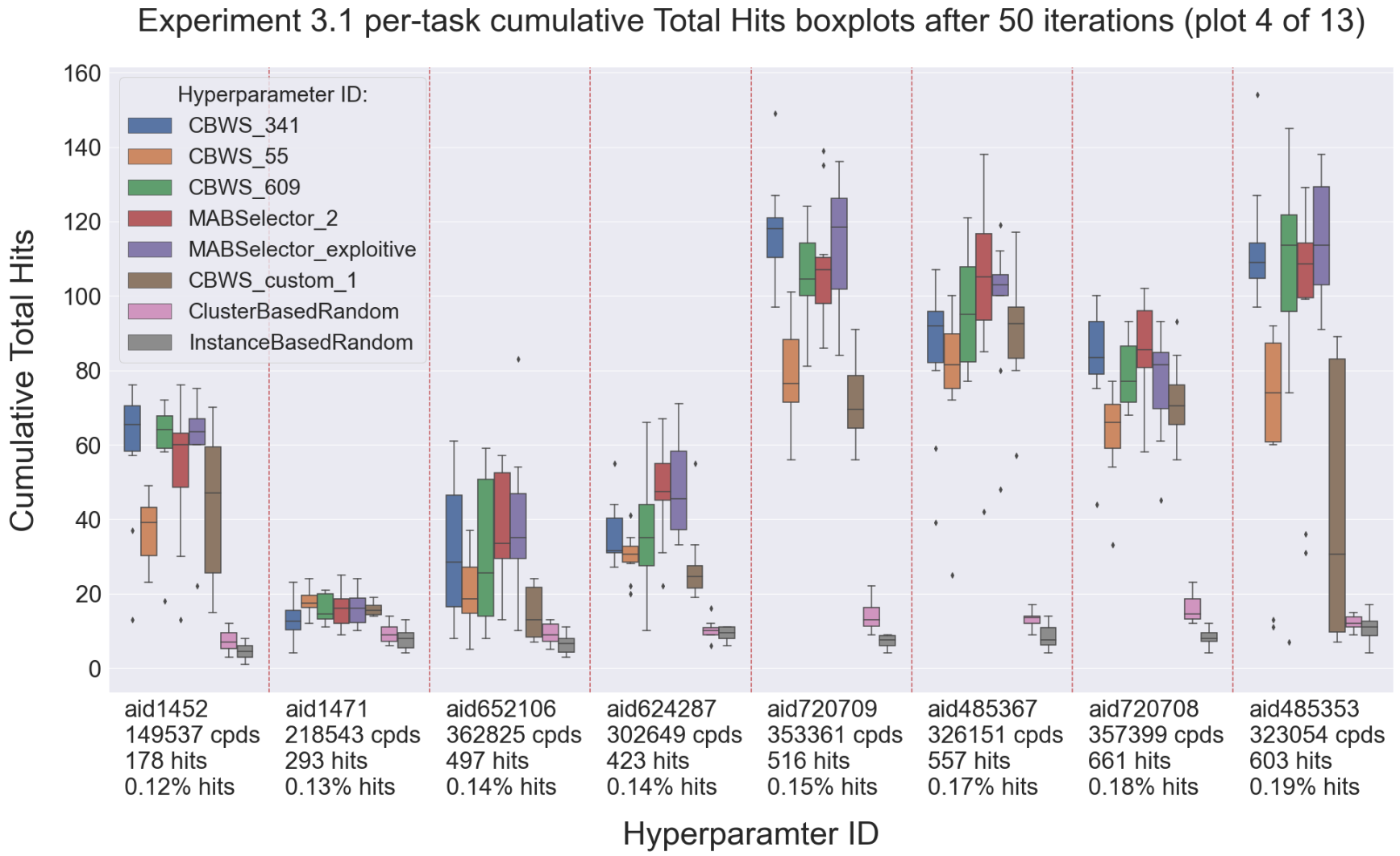


Figure A.1: Experiment 3.1 per task **Total Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(4 of 13 cont.)*

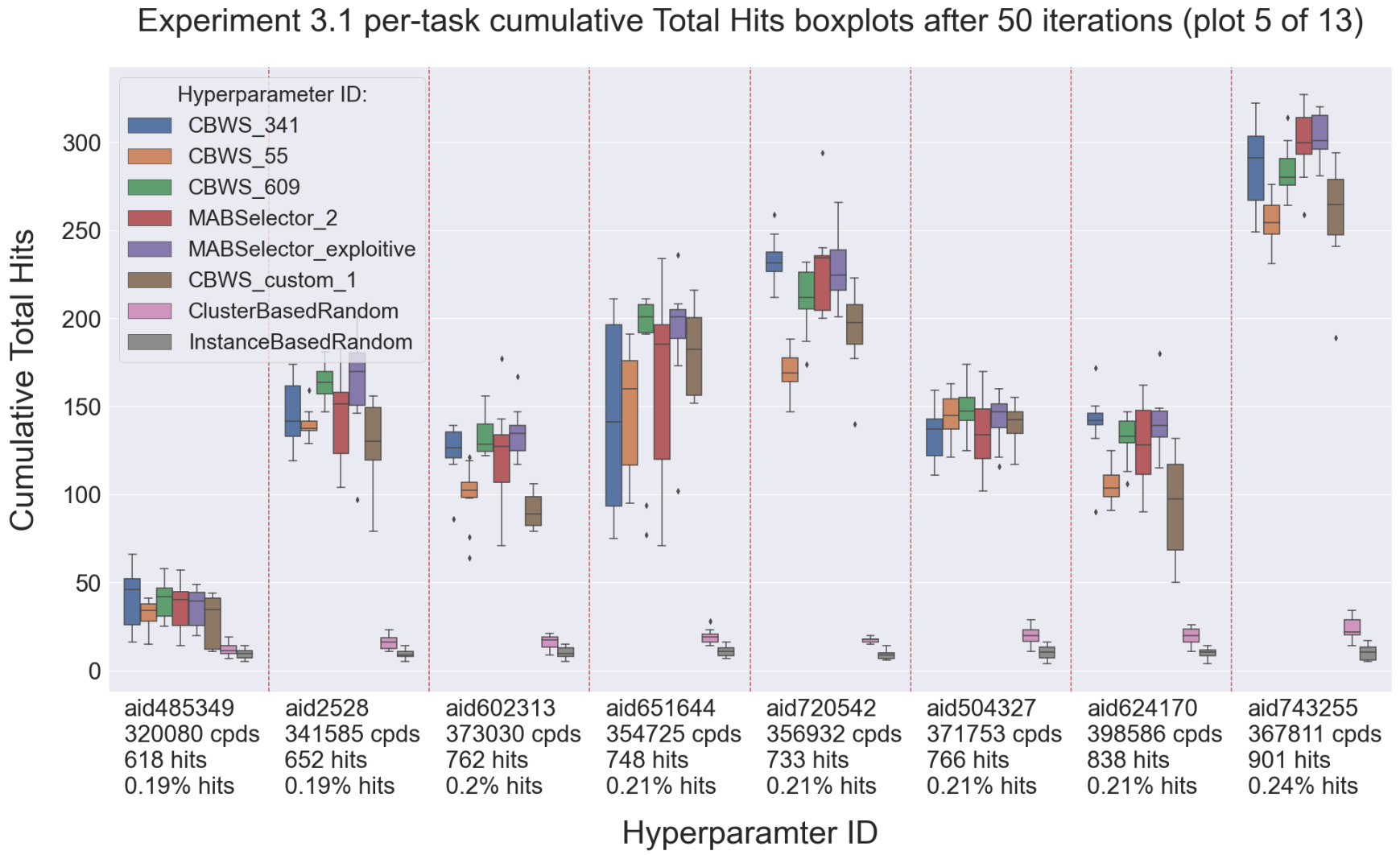


Figure A.1: Experiment 3.1 per task **Total Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(5 of 13 cont.)*

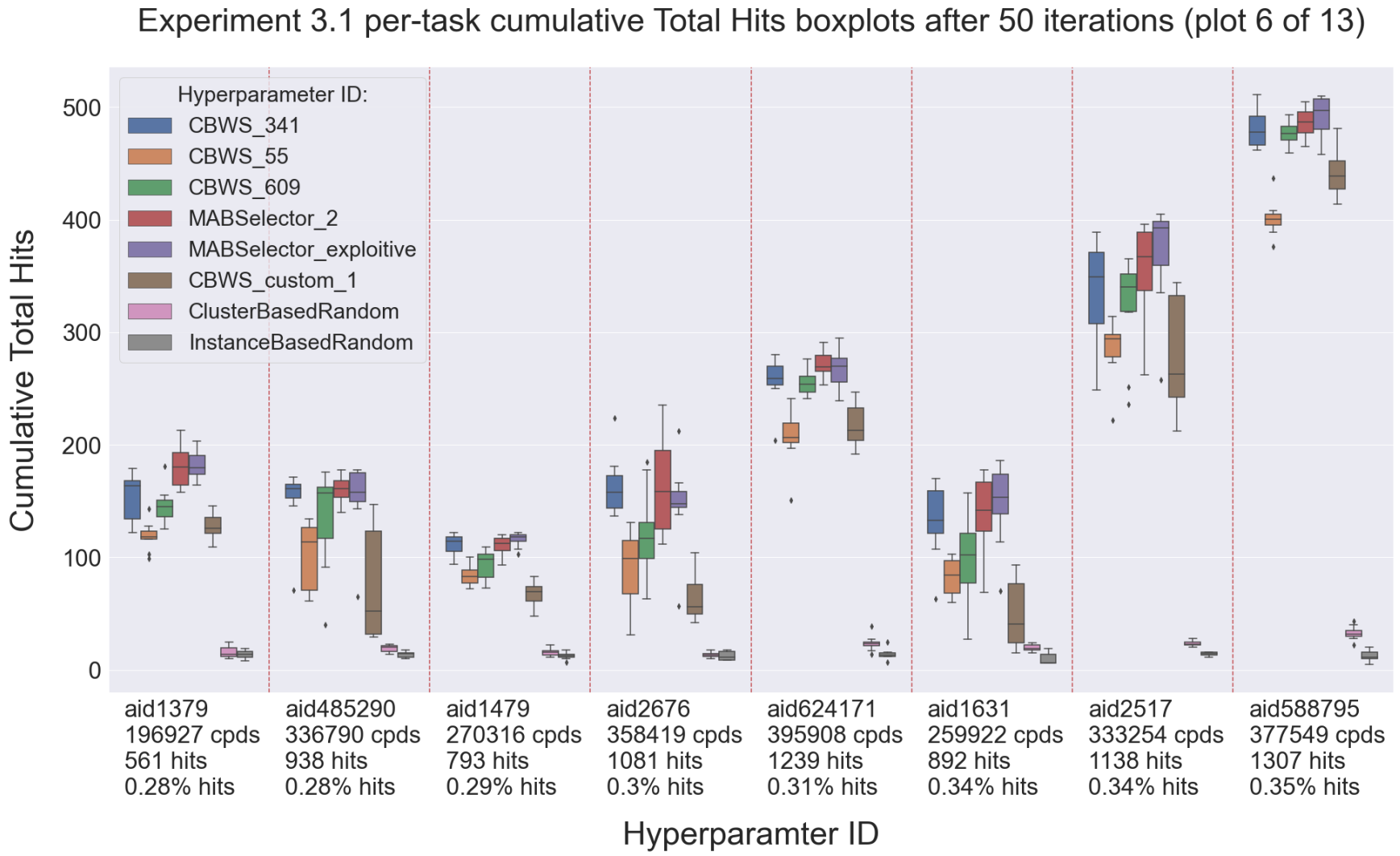


Figure A.1: Experiment 3.1 per task **Total Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(6 of 13 cont.)*

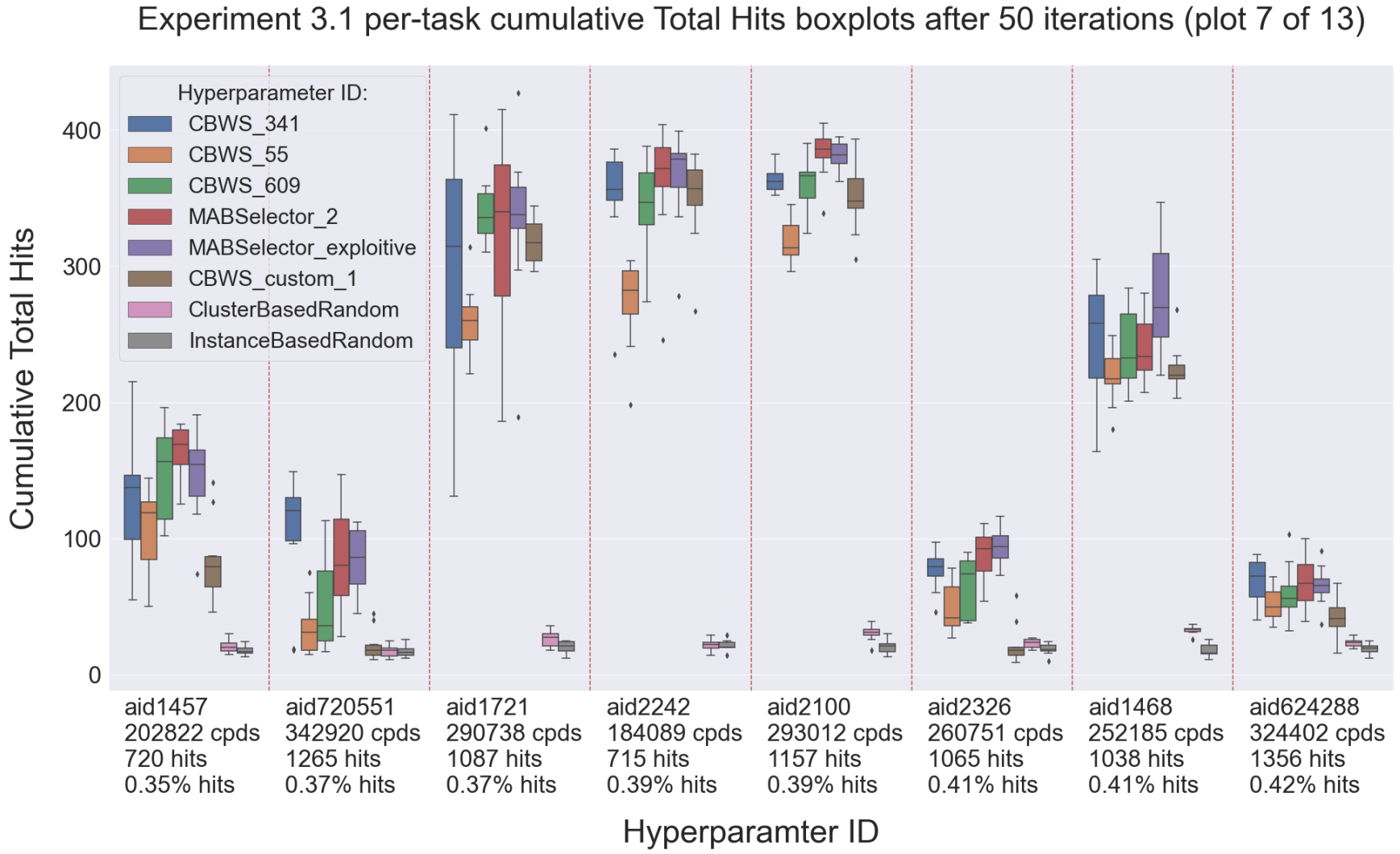


Figure A.1: Experiment 3.1 per task **Total Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(7 of 13 cont.)*

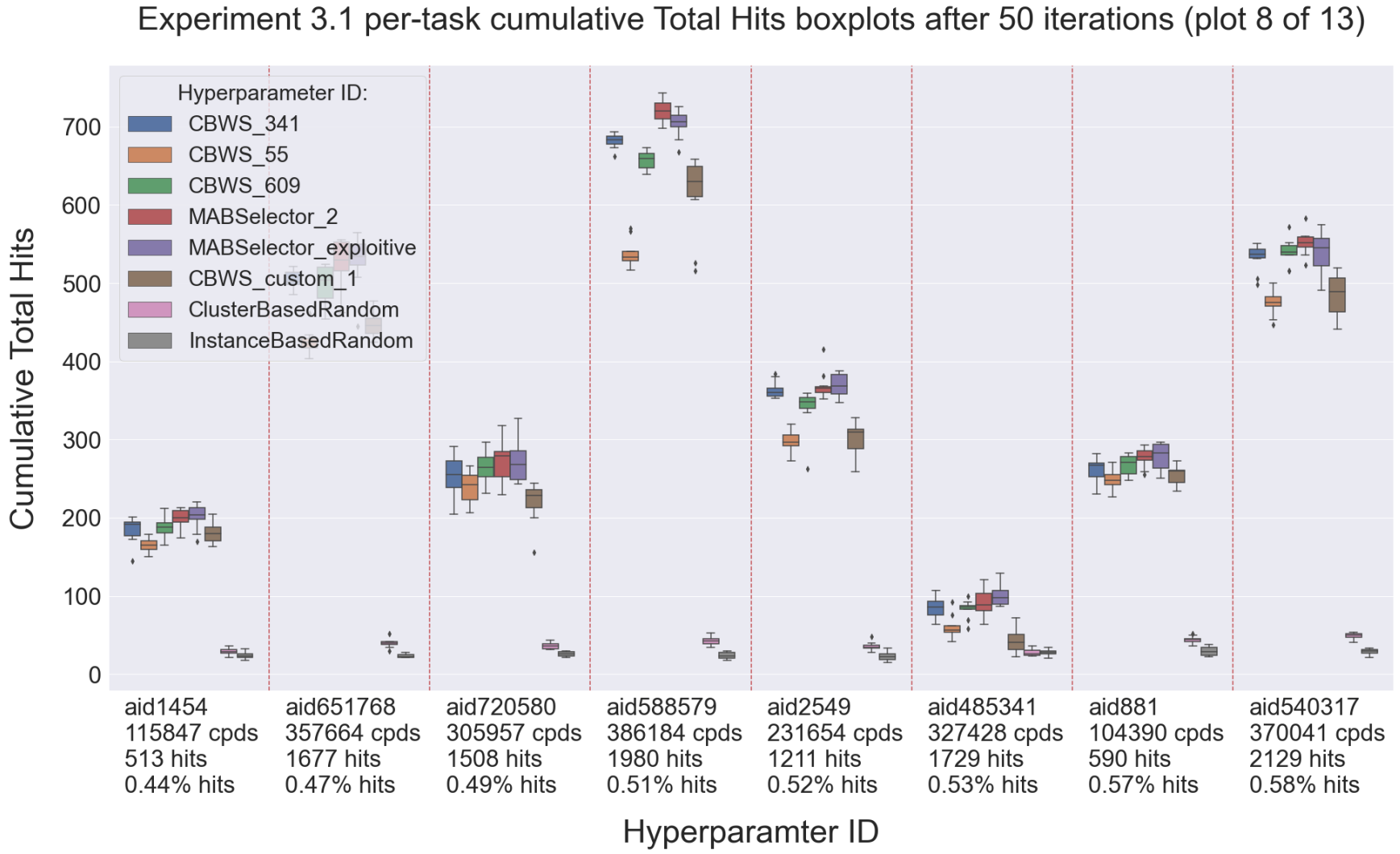


Figure A.1: Experiment 3.1 per task **Total Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(8 of 13 cont.)*

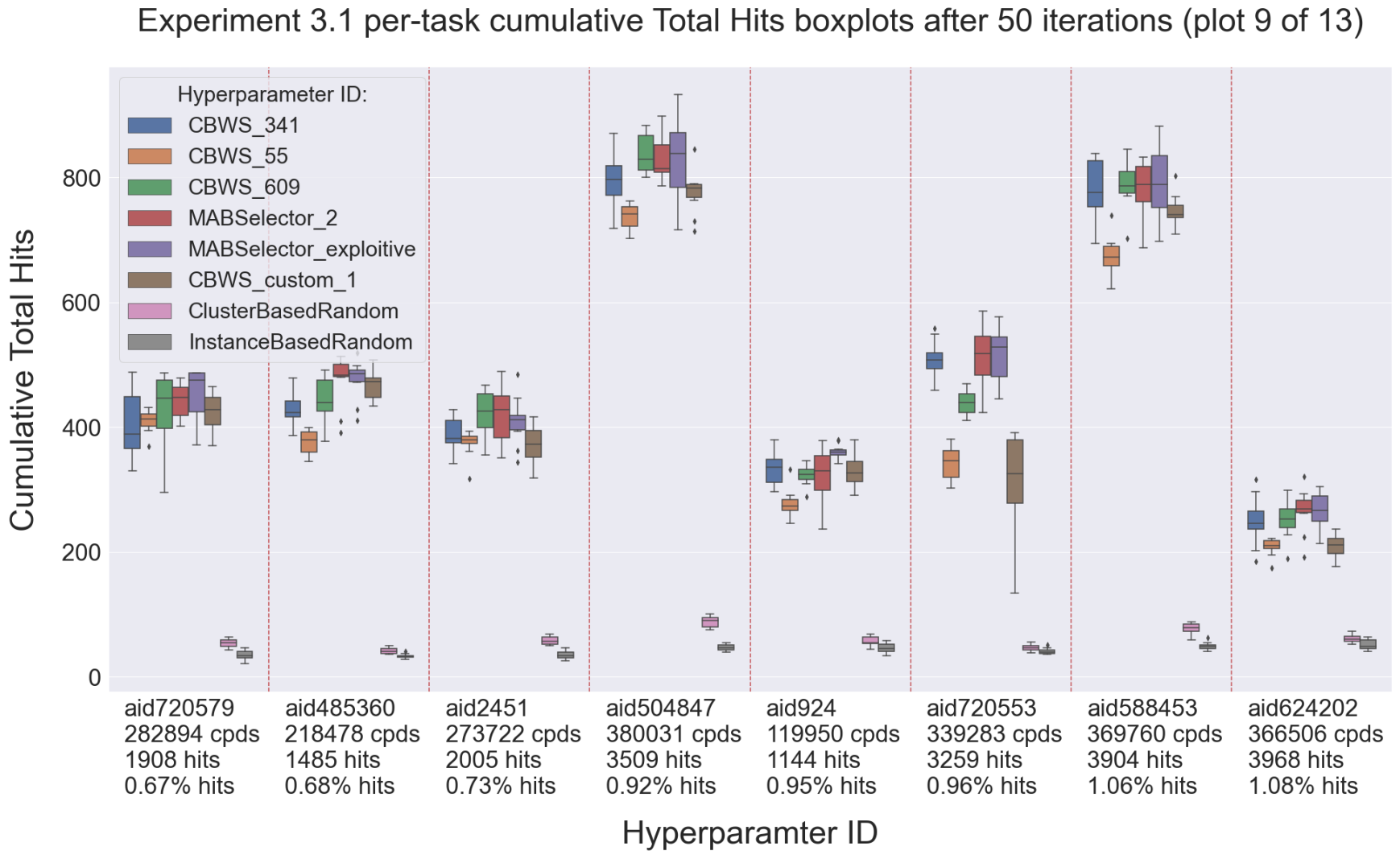


Figure A.1: Experiment 3.1 per task **Total Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(9 of 13 cont.)*

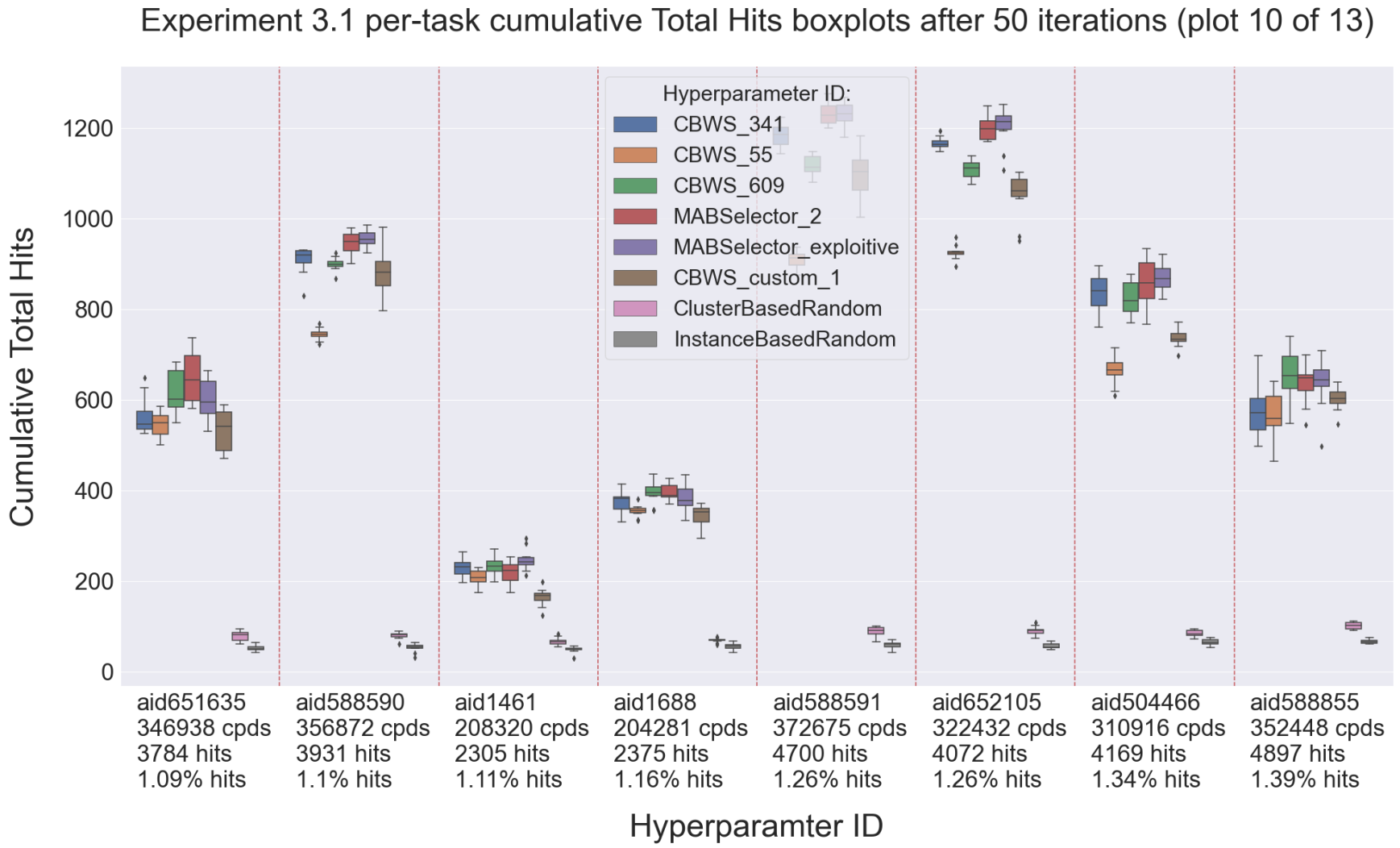


Figure A.1: Experiment 3.1 per task **Total Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(10 of 13 cont.)*

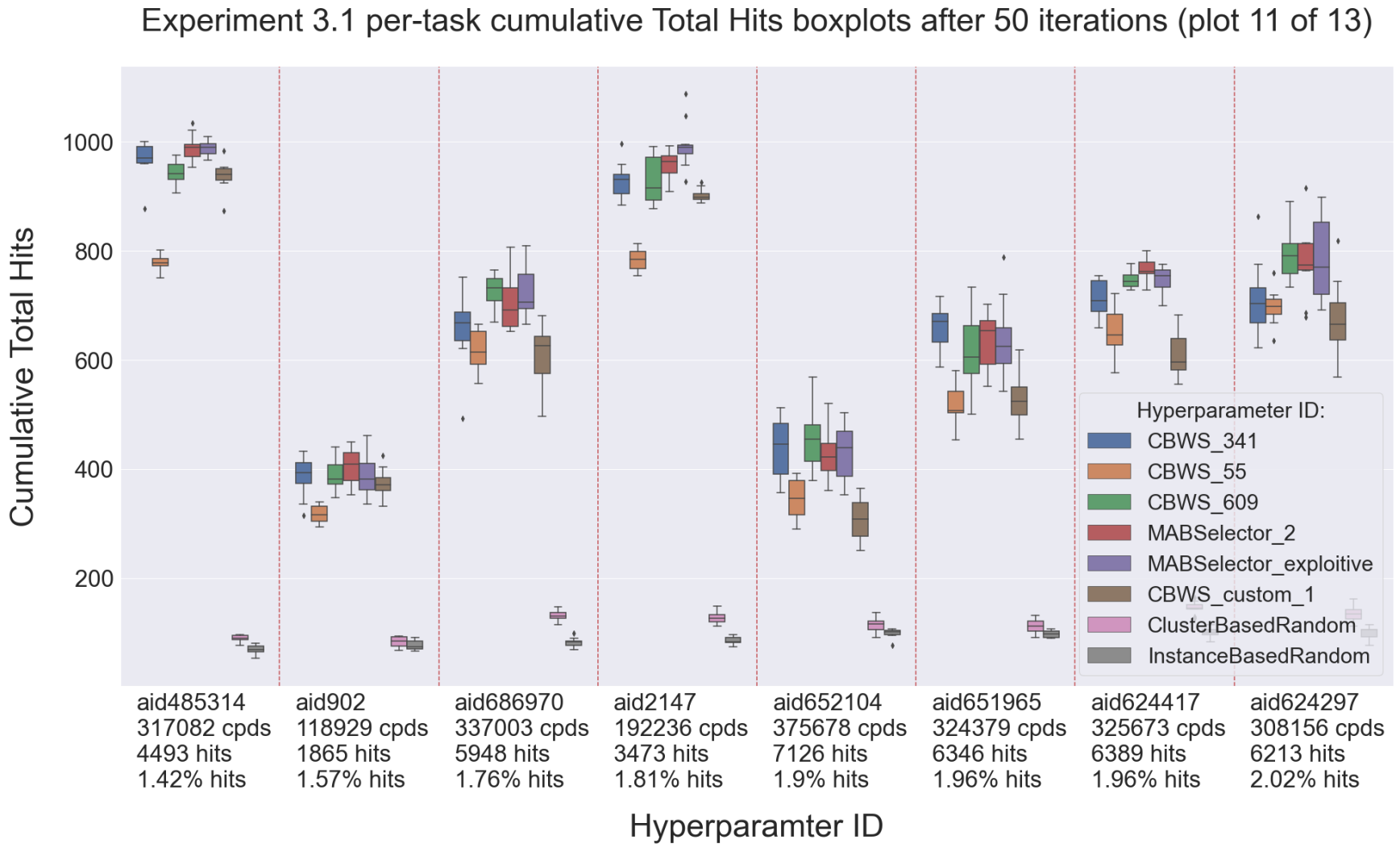


Figure A.1: Experiment 3.1 per task **Total Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(11 of 13 cont.)*

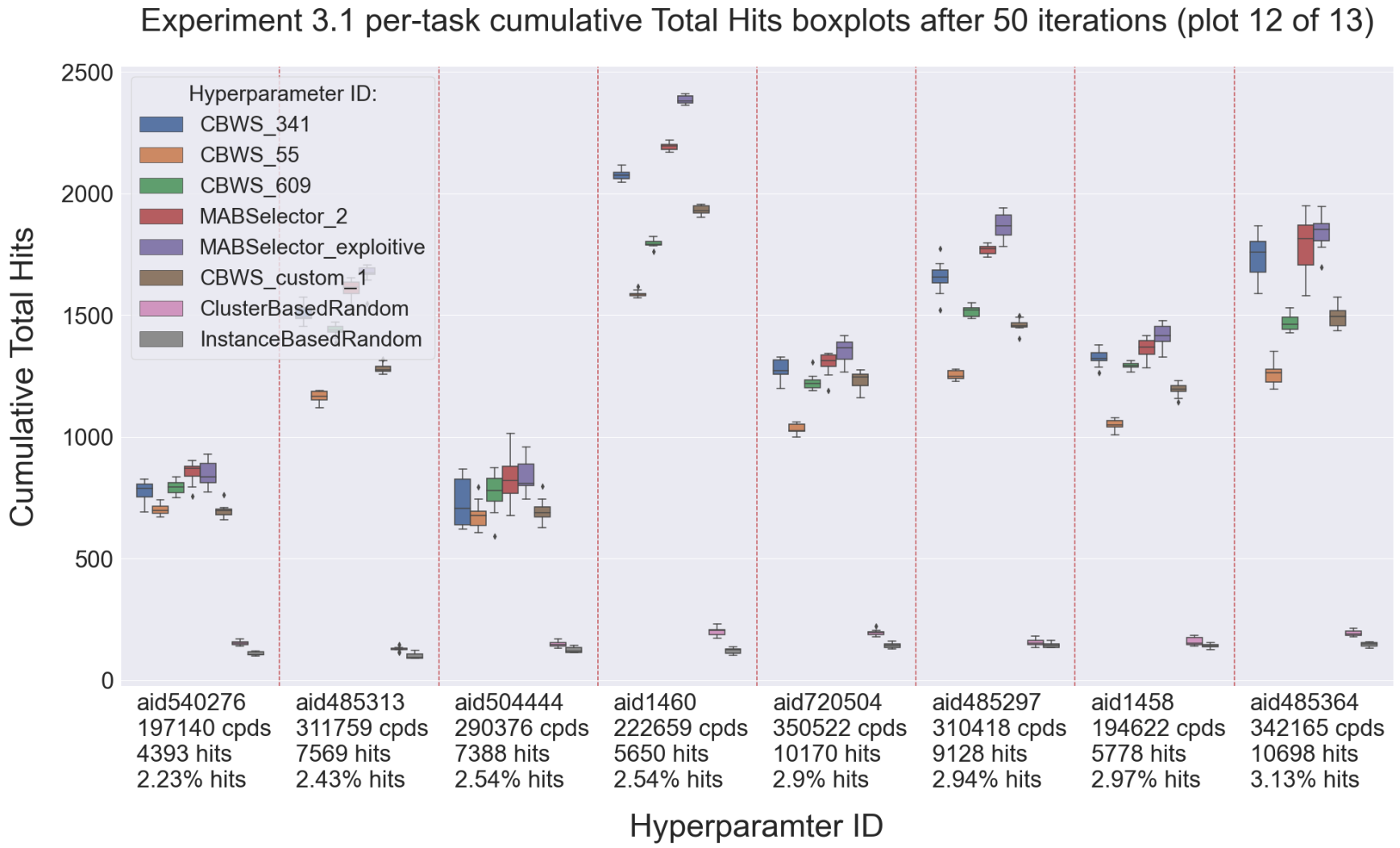


Figure A.1: Experiment 3.1 per task **Total Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(12 of 13 cont.)*

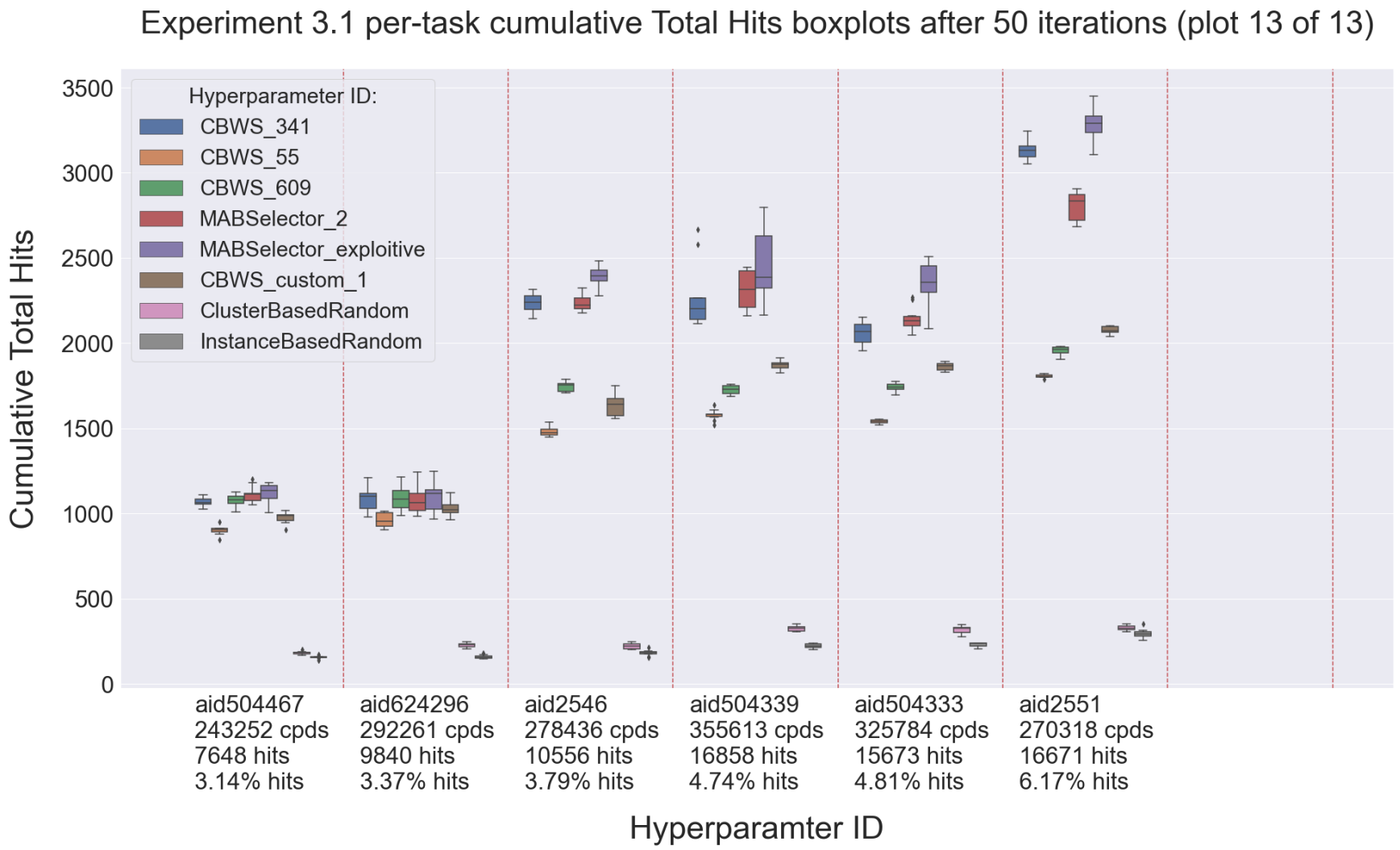


Figure A.1: Experiment 3.1 per task **Total Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(13 of 13)*

A.2 Experiment 3.1: Per Task Total Unique Hits Boxplots

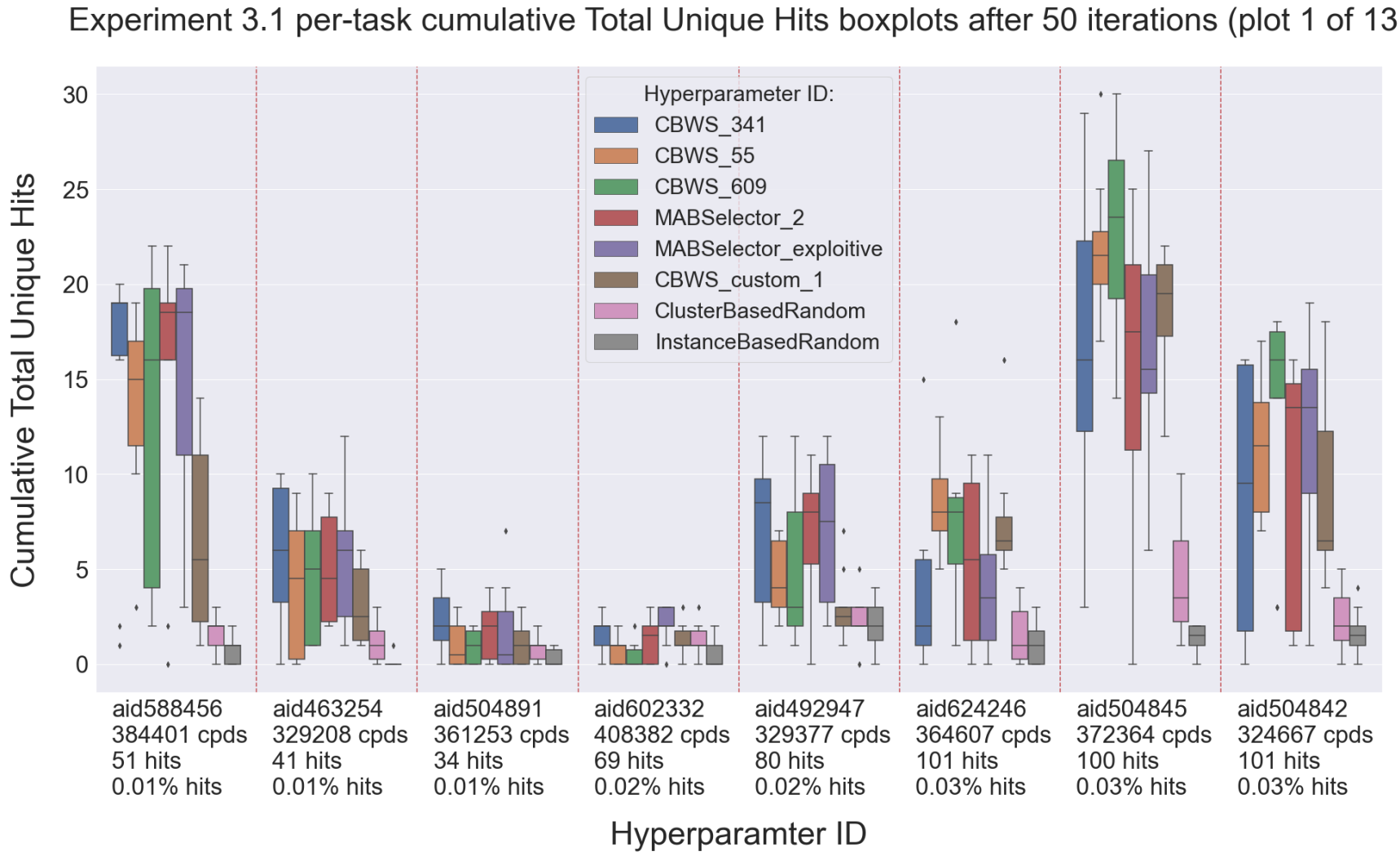


Figure A.2: Experiment 3.1 per task **Total Unique Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(1 of 13 cont.)*

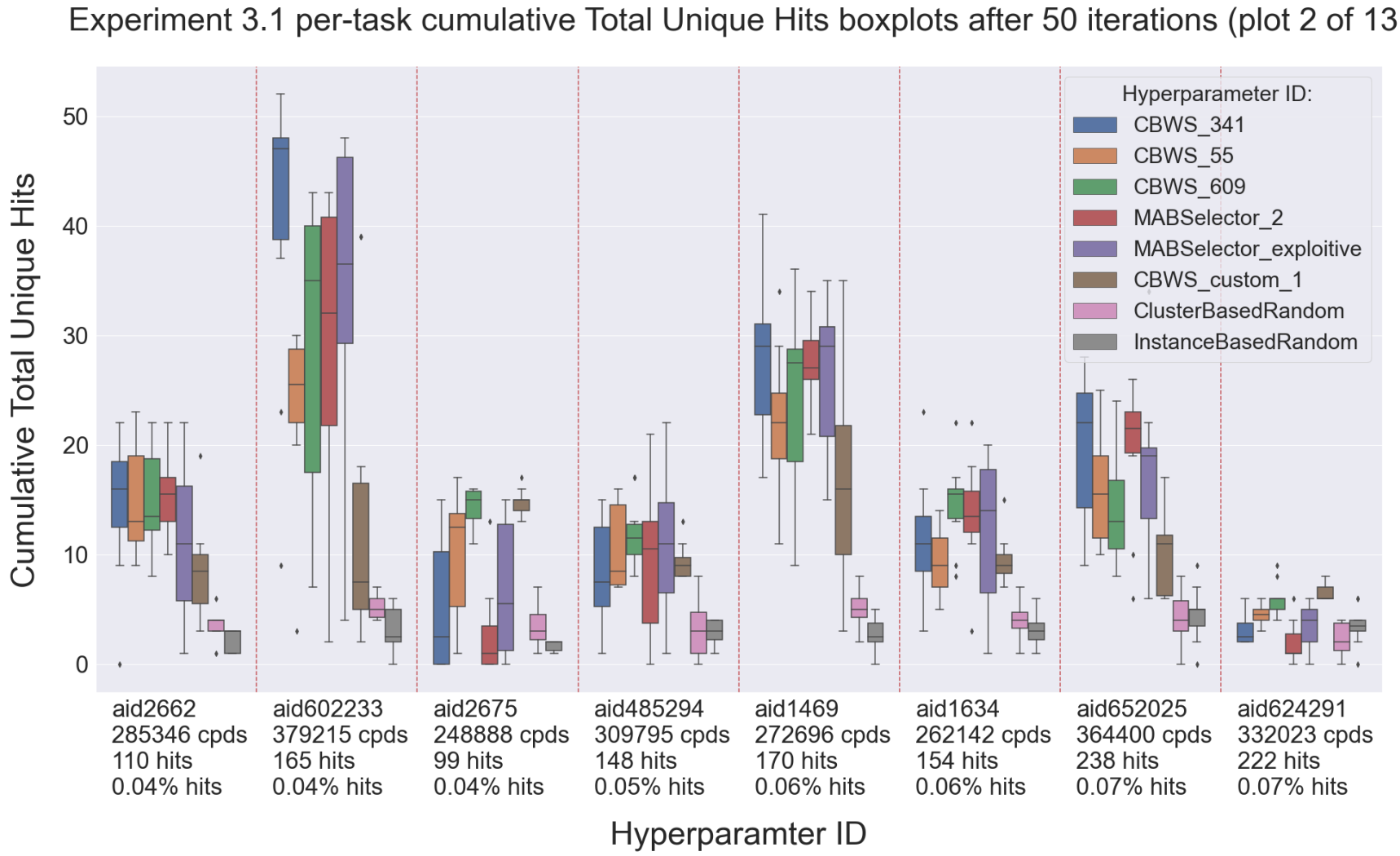


Figure A.2: Experiment 3.1 per task **Total Unique Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(2 of 13 cont.)*

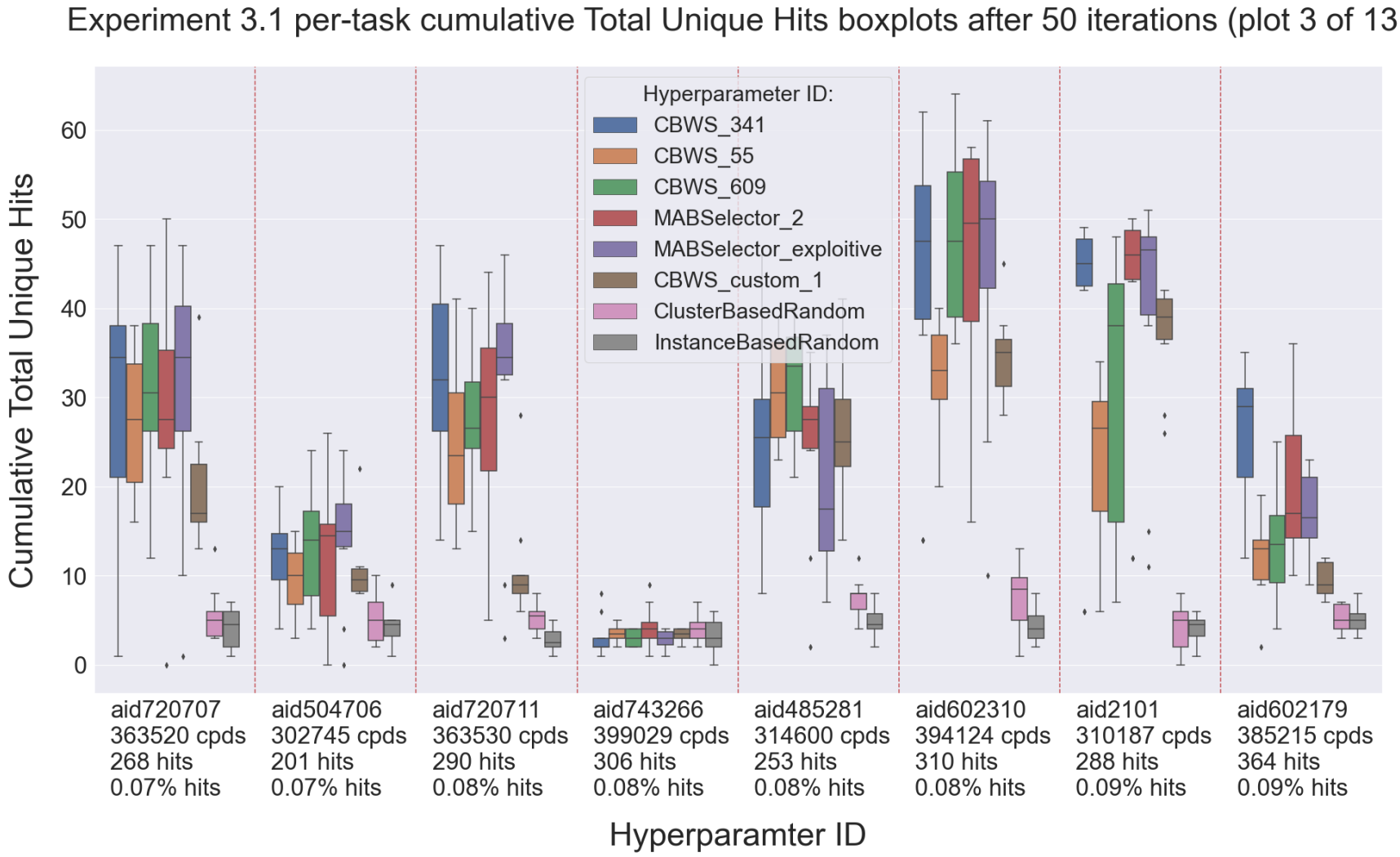


Figure A.2: Experiment 3.1 per task **Total Unique Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(3 of 13 cont.)*

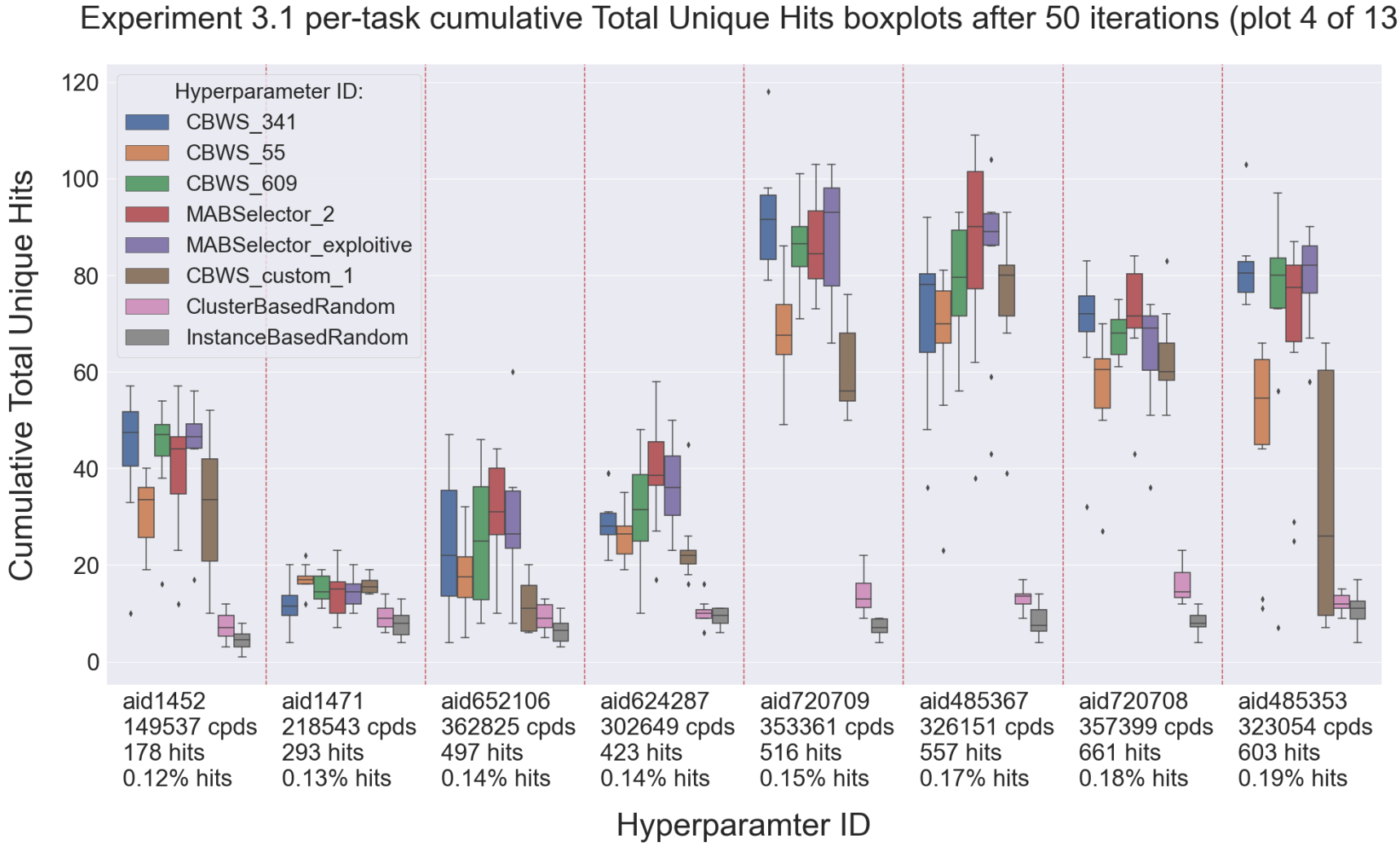


Figure A.2: Experiment 3.1 per task **Total Unique Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(4 of 13 cont.)*

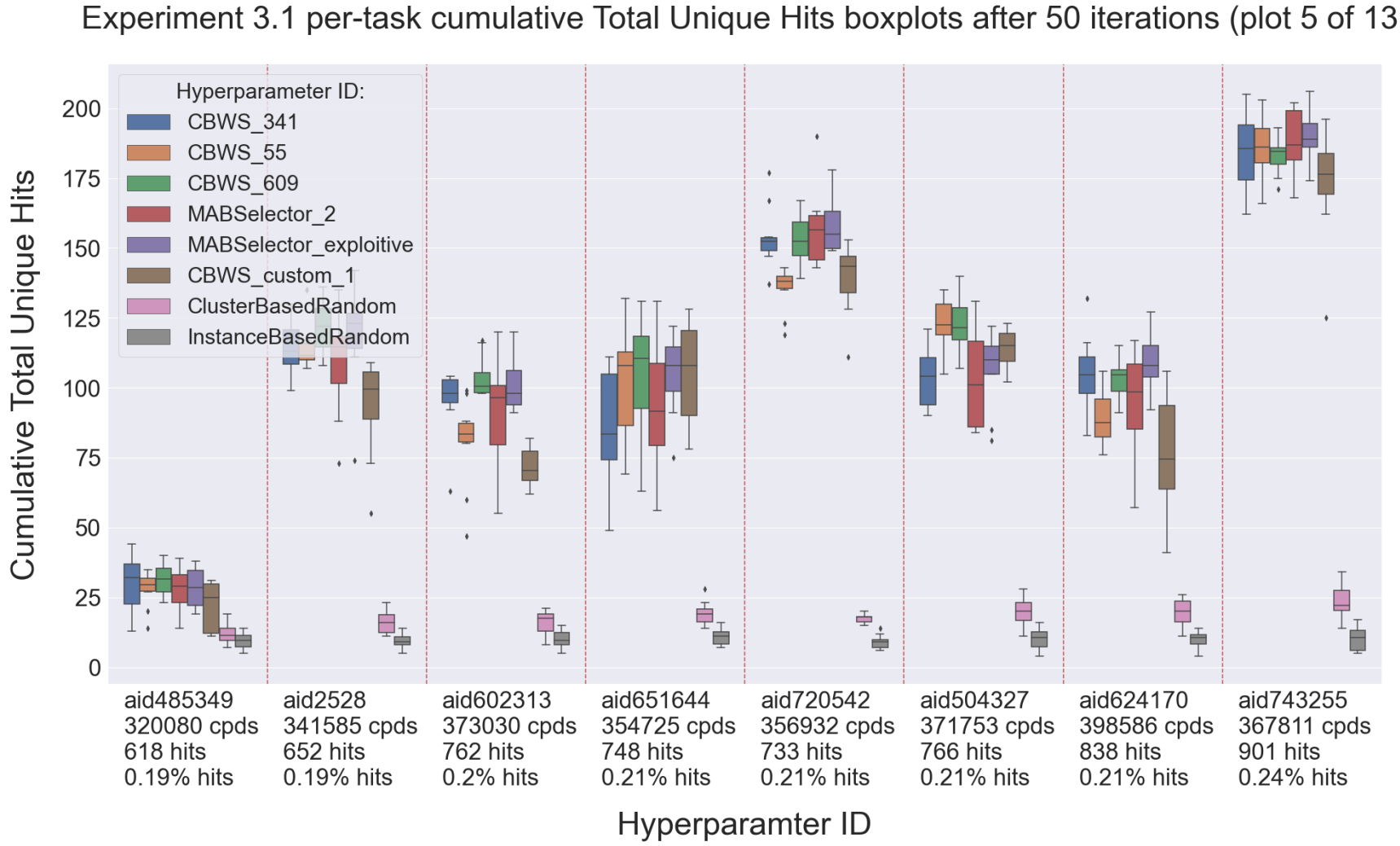


Figure A.2: Experiment 3.1 per task **Total Unique Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(5 of 13 cont.)*

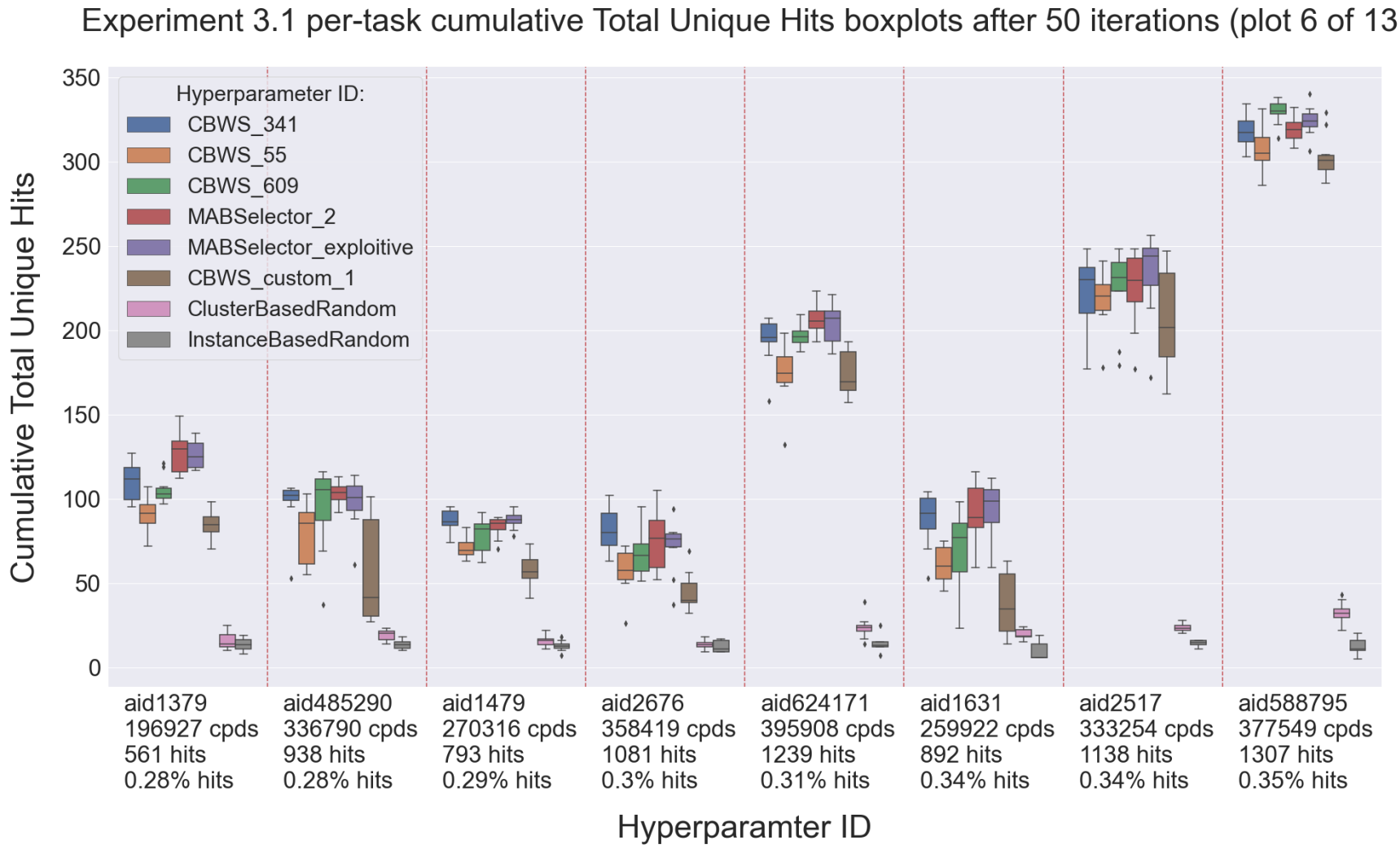


Figure A.2: Experiment 3.1 per task **Total Unique Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(6 of 13 cont.)*

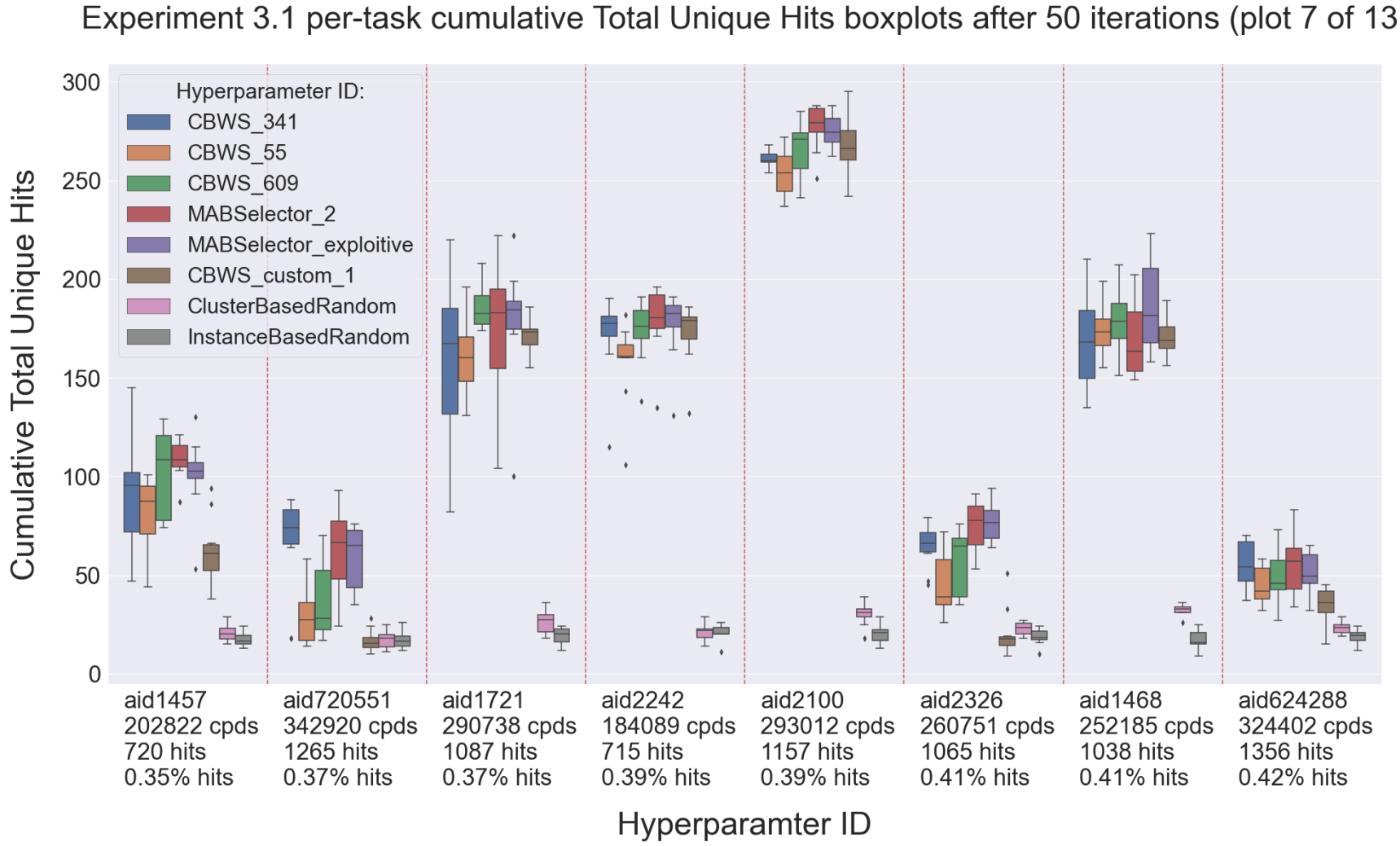


Figure A.2: Experiment 3.1 per task **Total Unique Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(7 of 13 cont.)*

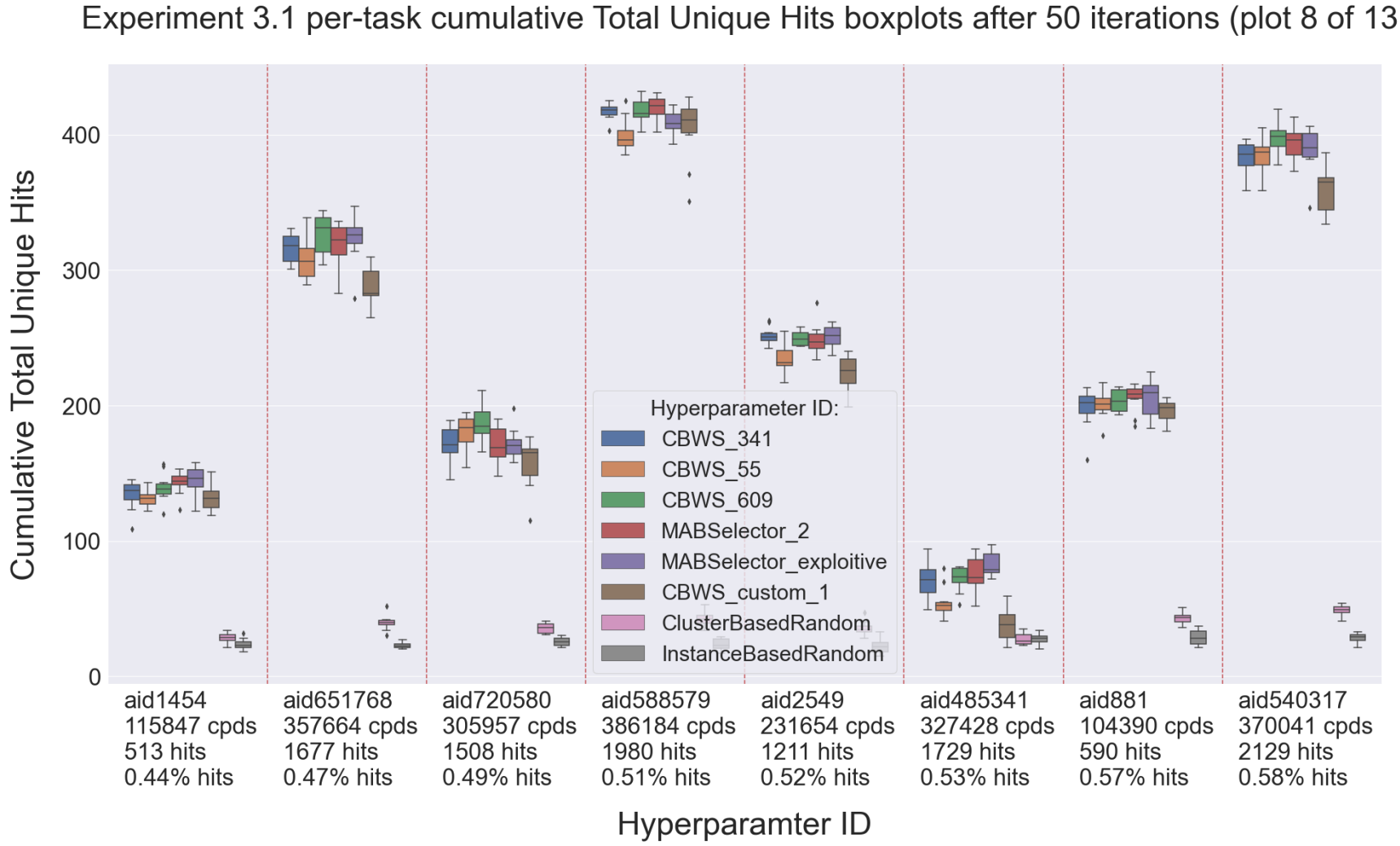


Figure A.2: Experiment 3.1 per task **Total Unique Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(8 of 13 cont.)*

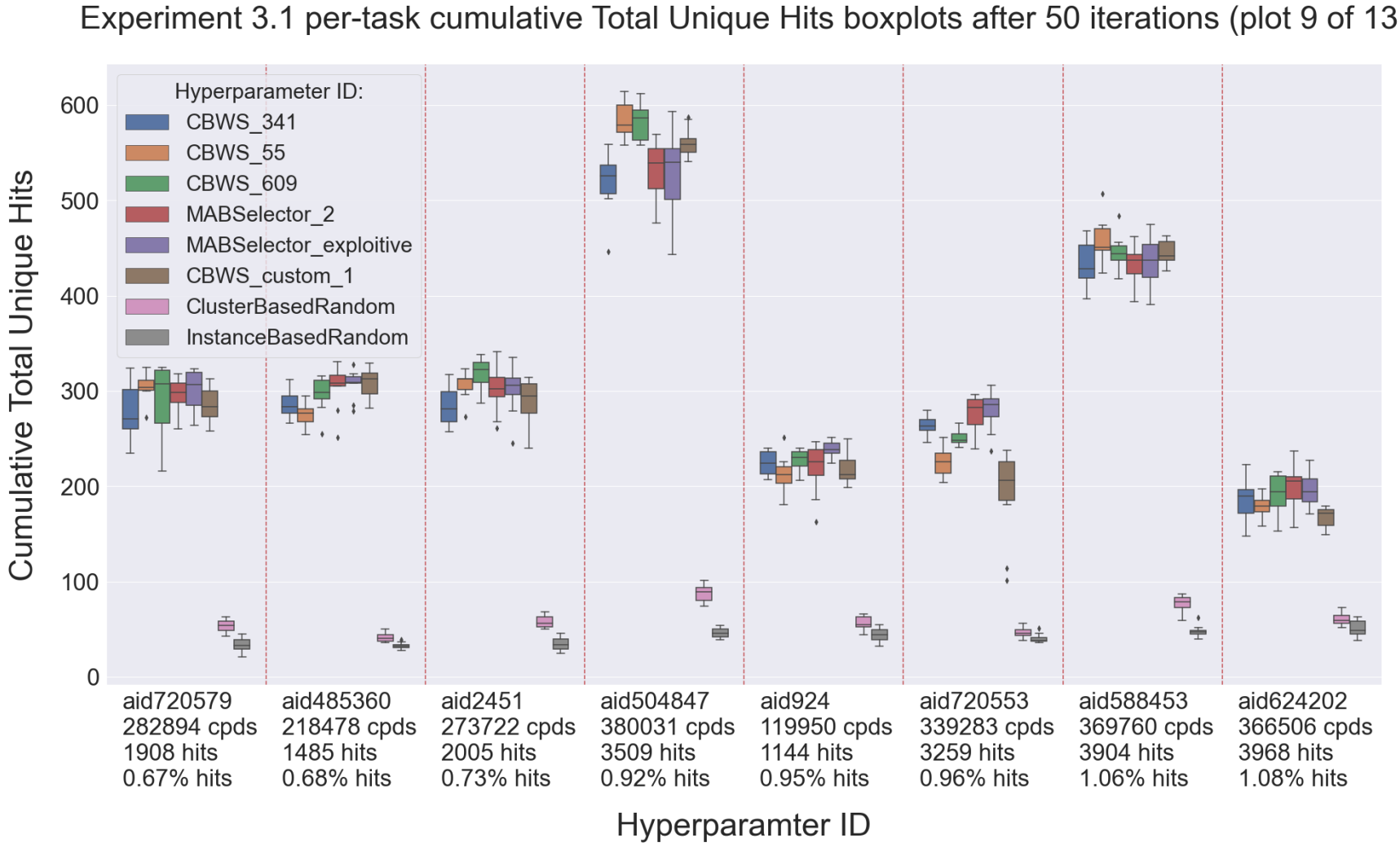


Figure A.2: Experiment 3.1 per task **Total Unique Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(9 of 13 cont.)*

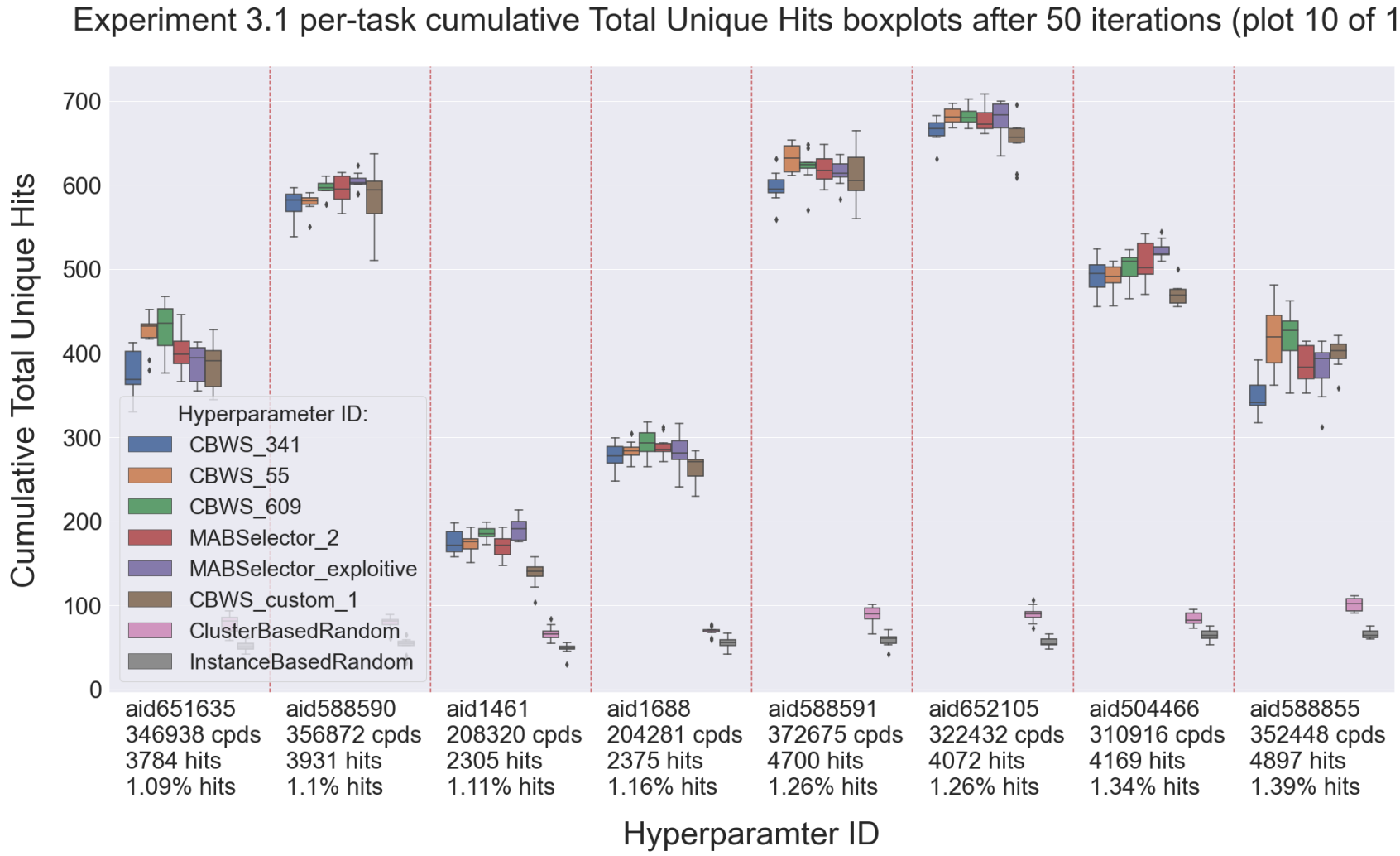


Figure A.2: Experiment 3.1 per task **Total Unique Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(10 of 13 cont.)*

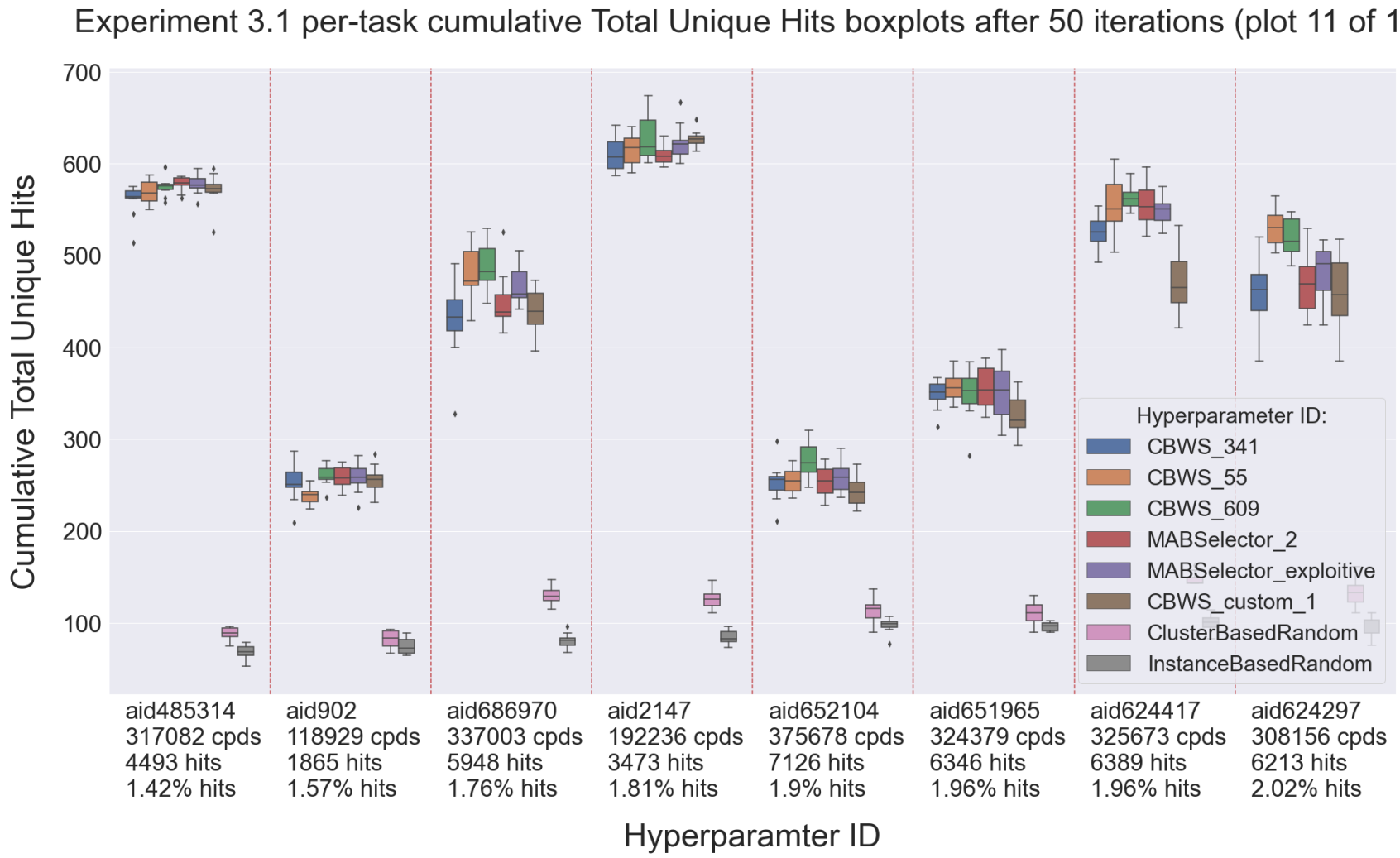


Figure A.2: Experiment 3.1 per task **Total Unique Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(11 of 13 cont.)*

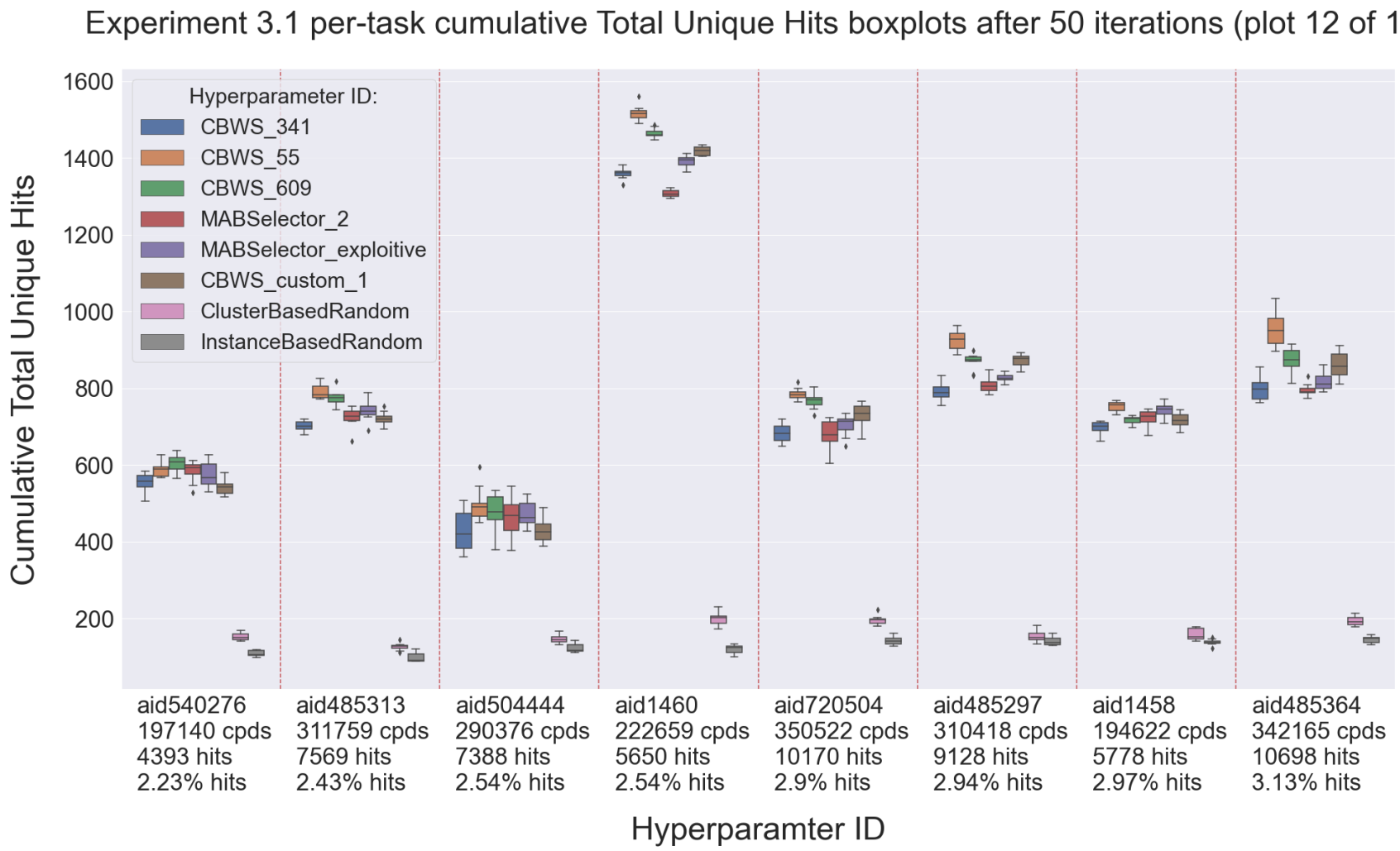


Figure A.2: Experiment 3.1 per task **Total Unique Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(12 of 13 cont.)*

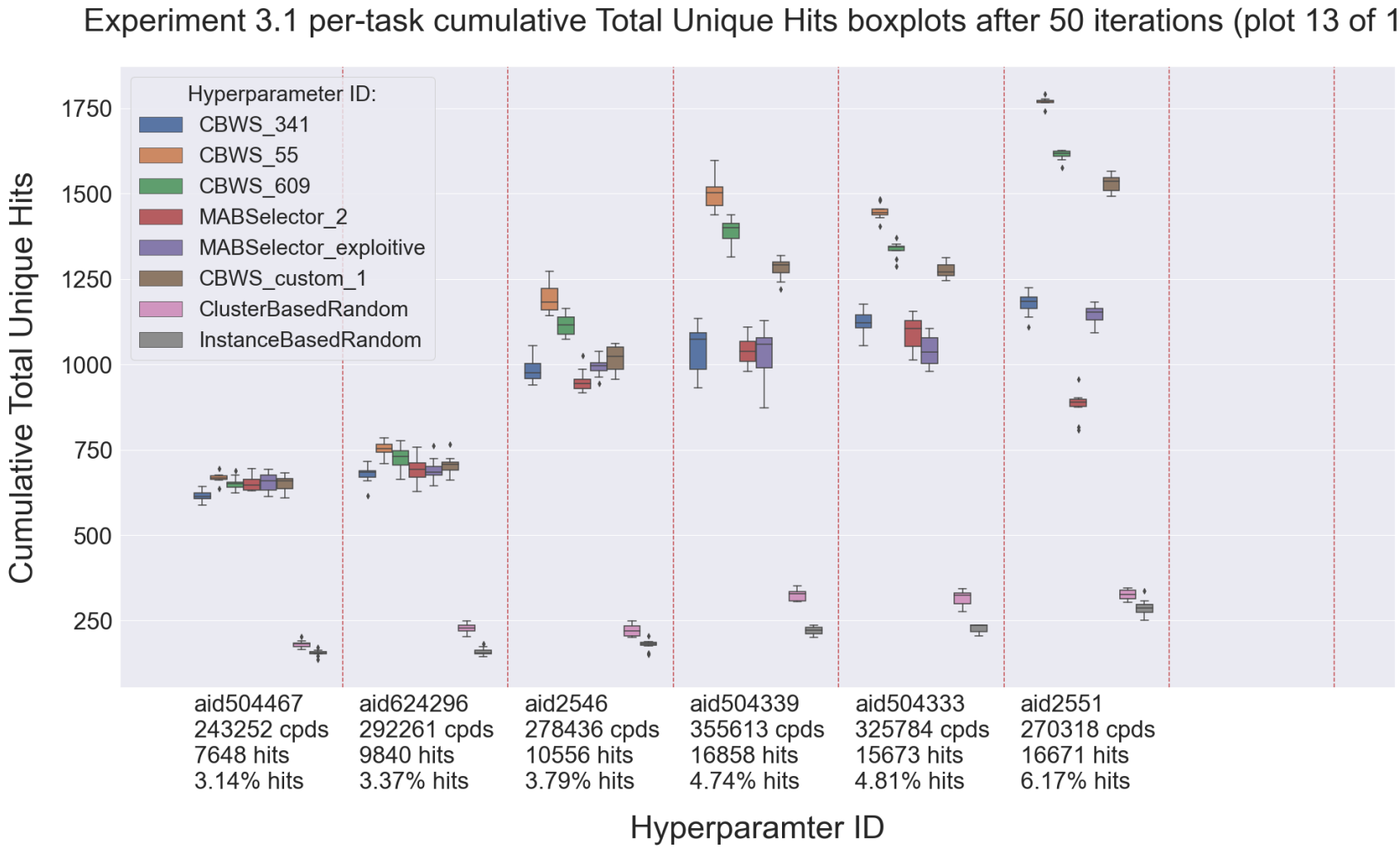


Figure A.2: Experiment 3.1 per task **Total Unique Hits** boxplots after 50 iterations (102 tasks). The x-tick labels for each task include number of compounds, number of hits, and hit %. *(13 of 13)*

Appendix B

Experiment 3.1: Per Task Contrast Estimate Based on Medians (CEM)

B.1 Experiment 3.1: Per Task Total Hits CEM

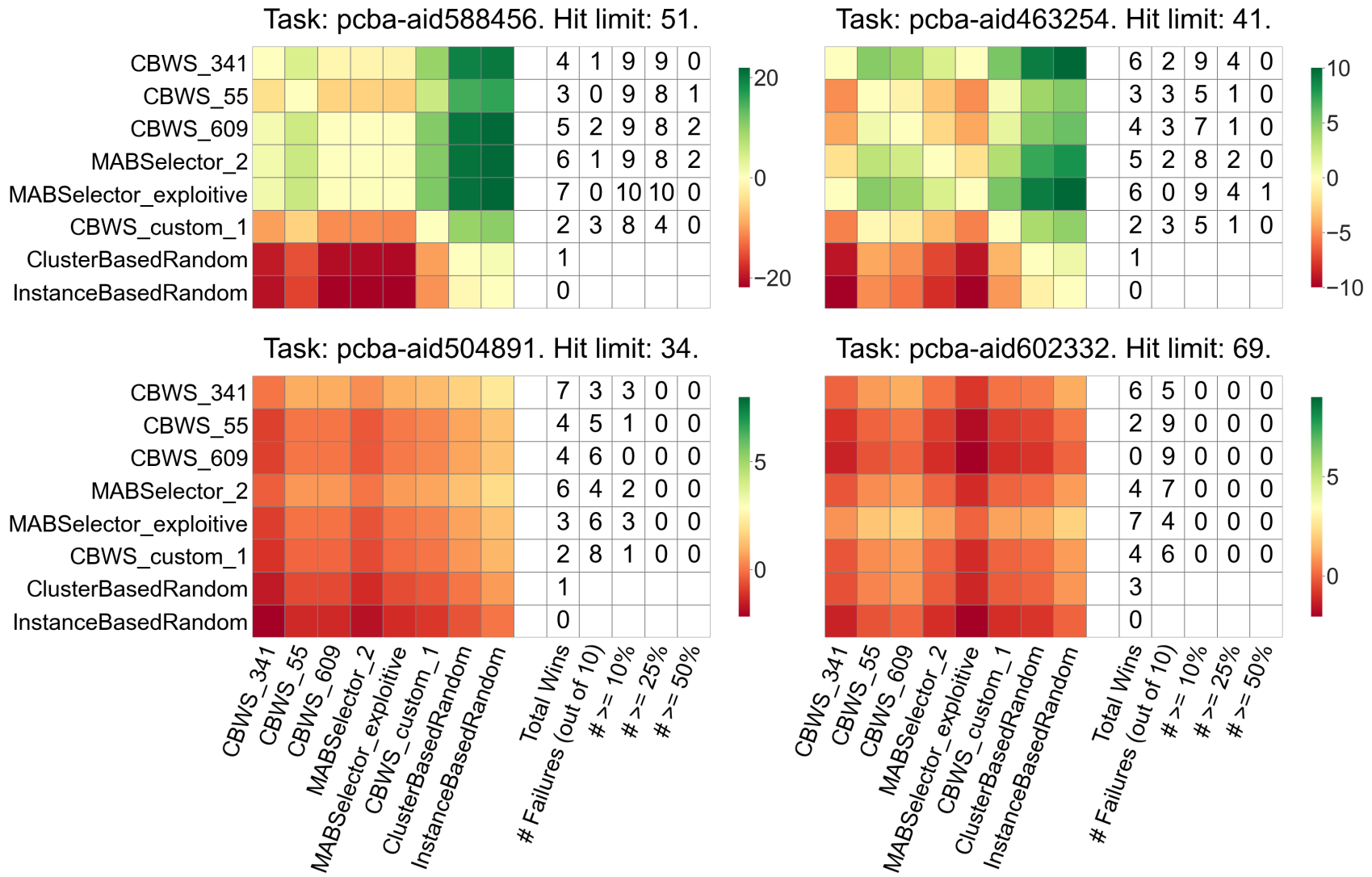


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(1 of 26 cont.)*

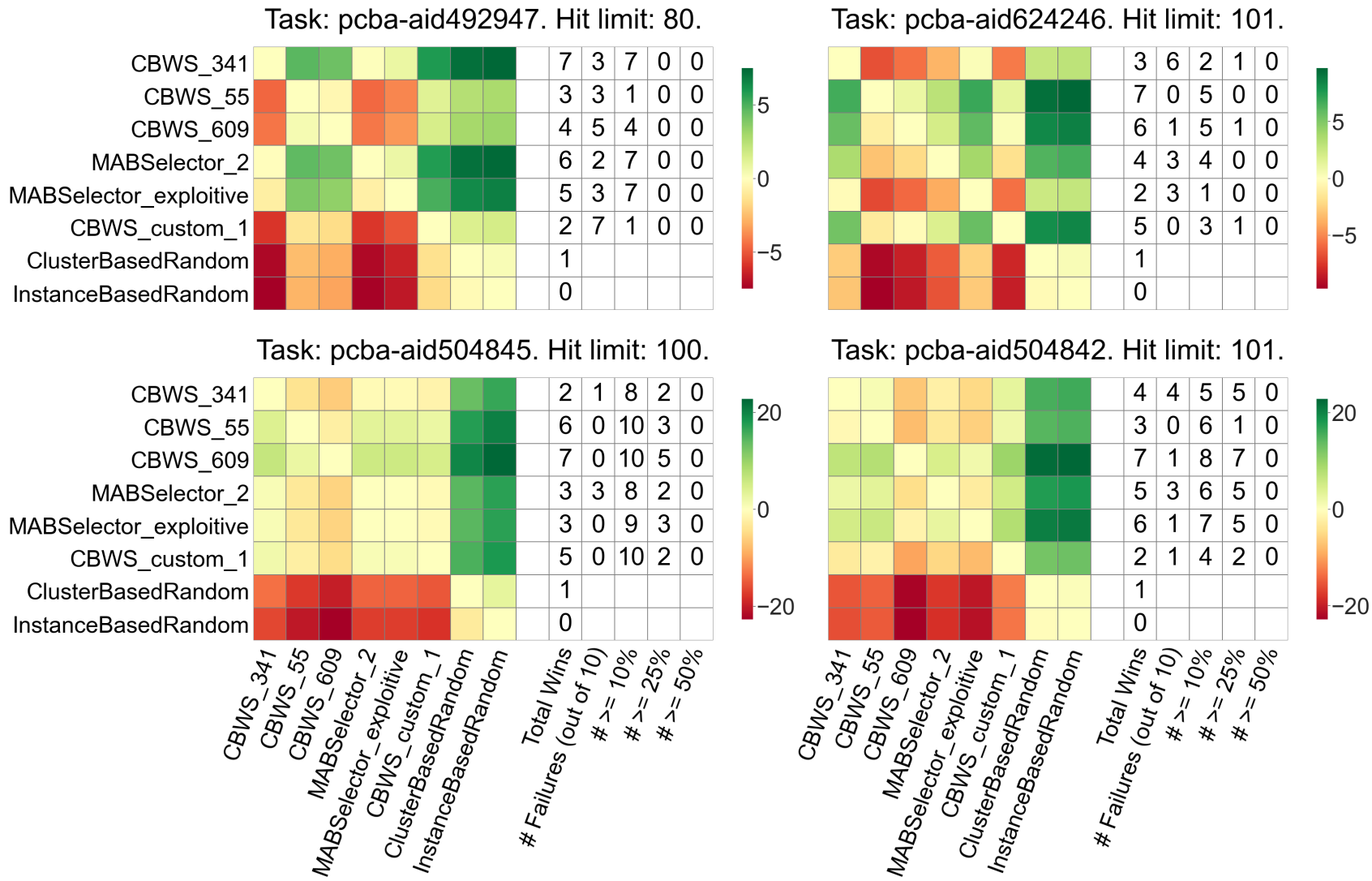


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(2 of 26 cont.)*

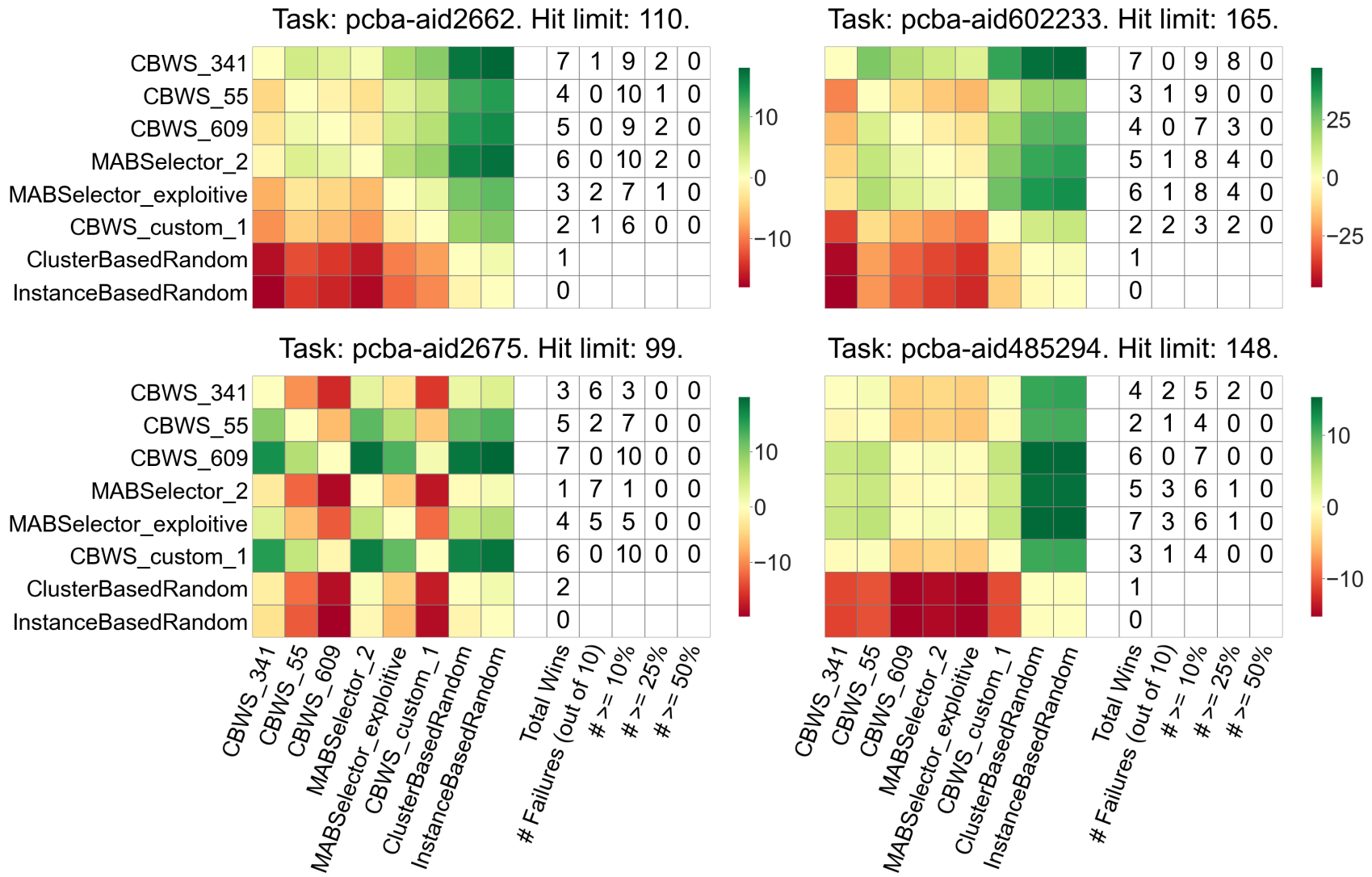


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(3 of 26 cont.)*

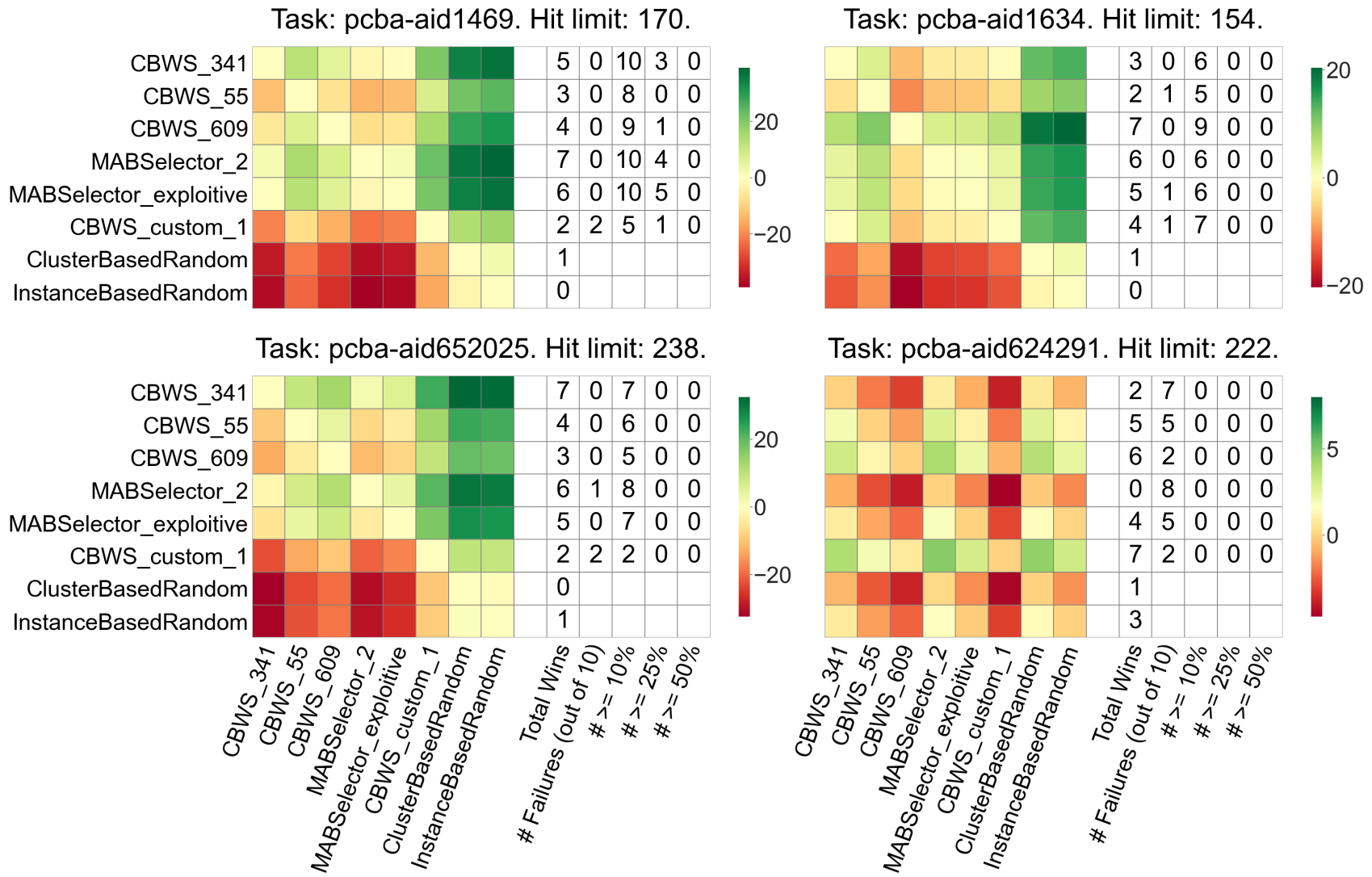


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(4 of 26 cont.)*

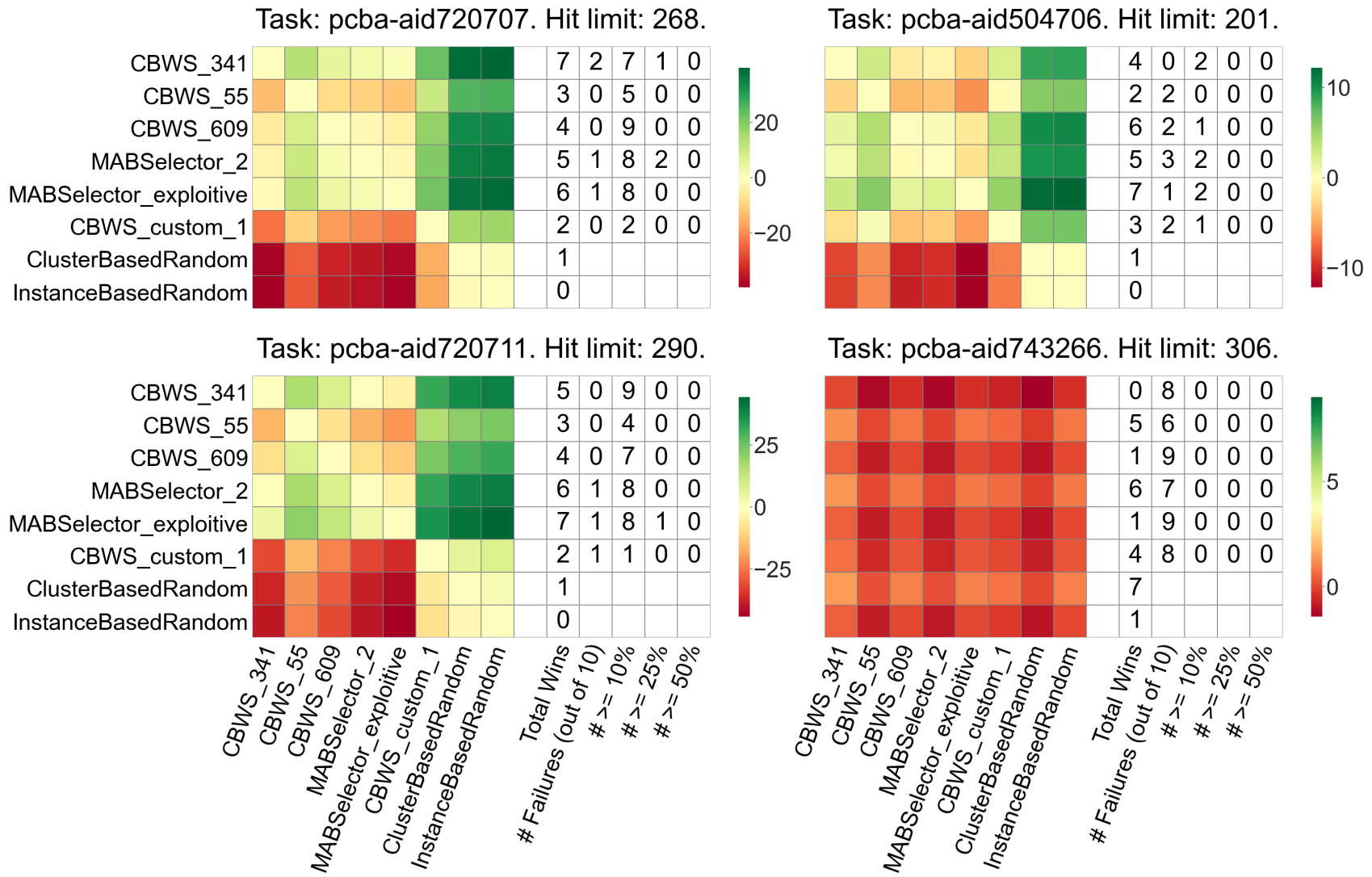


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(5 of 26 cont.)*

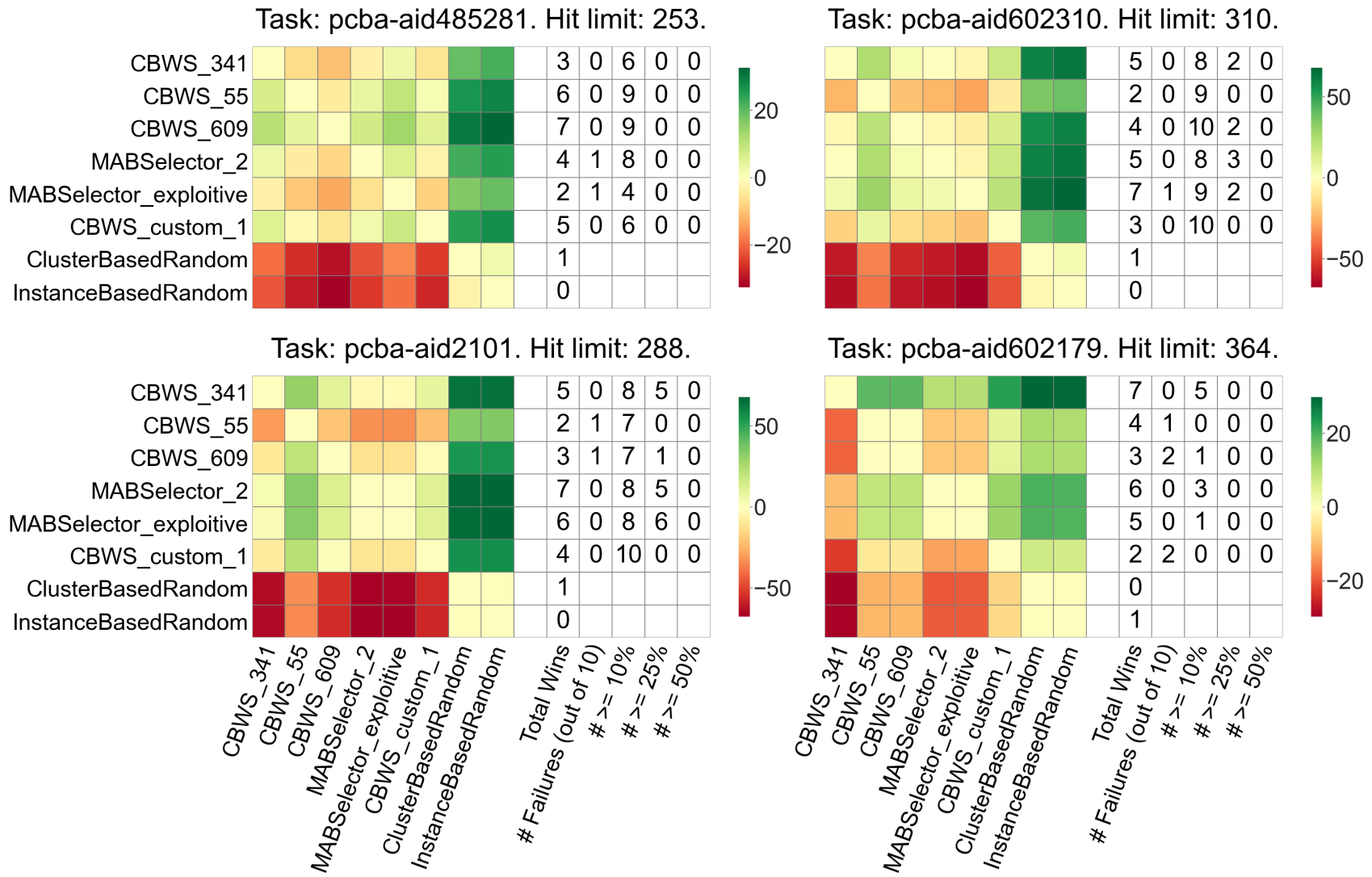


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(6 of 26 cont.)*

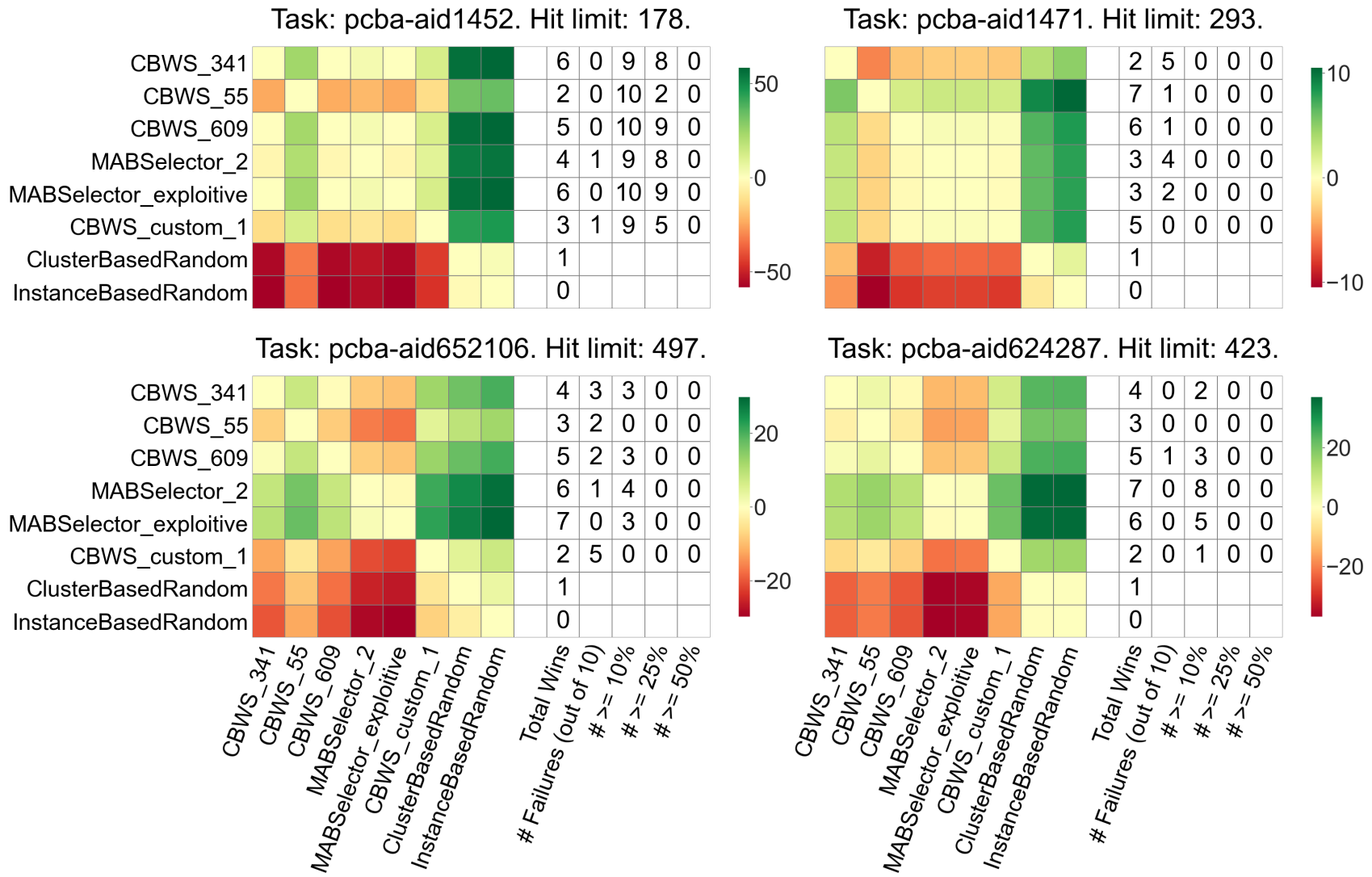


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(7 of 26 cont.)*

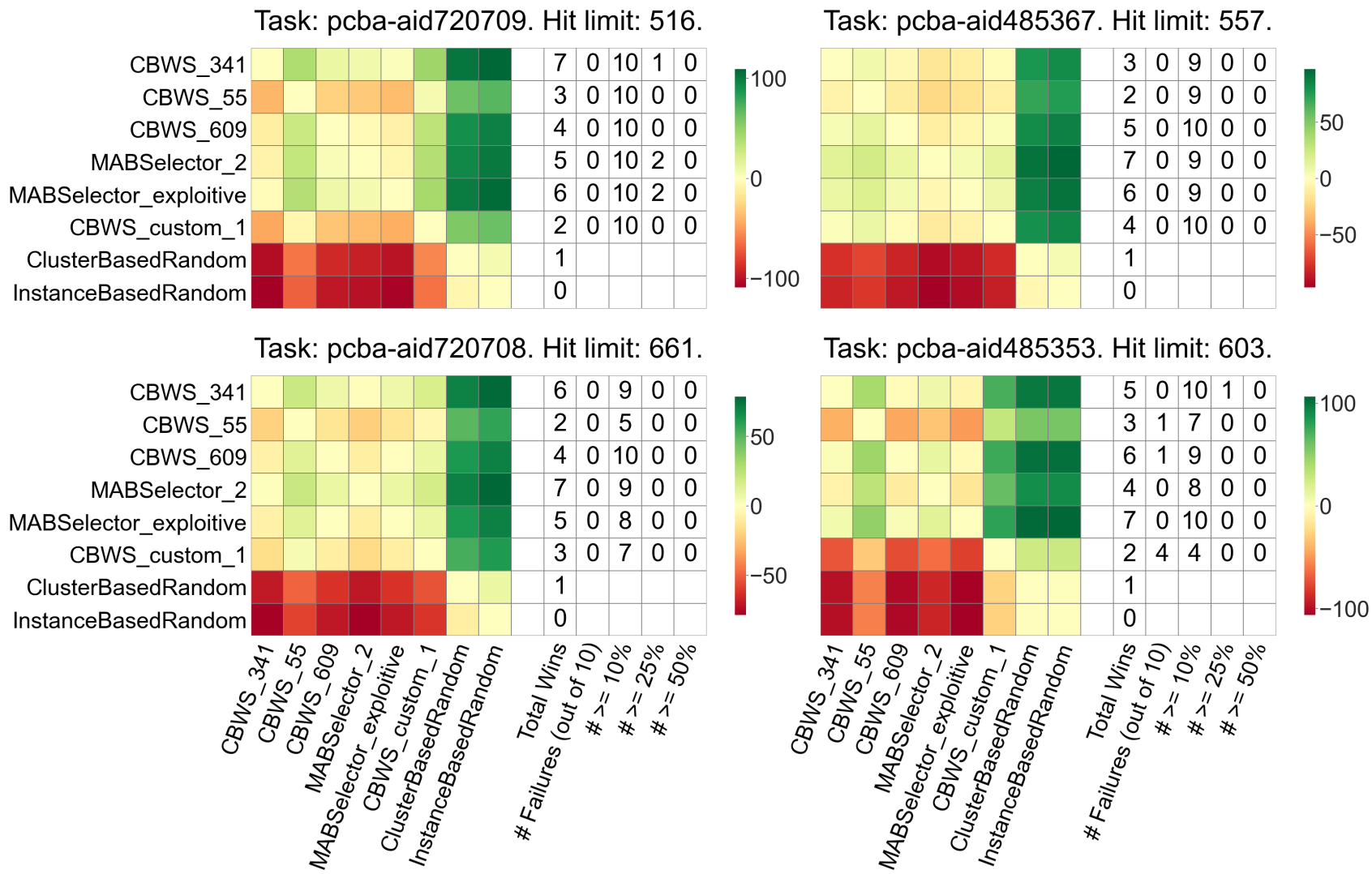


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(8 of 26 cont.)*

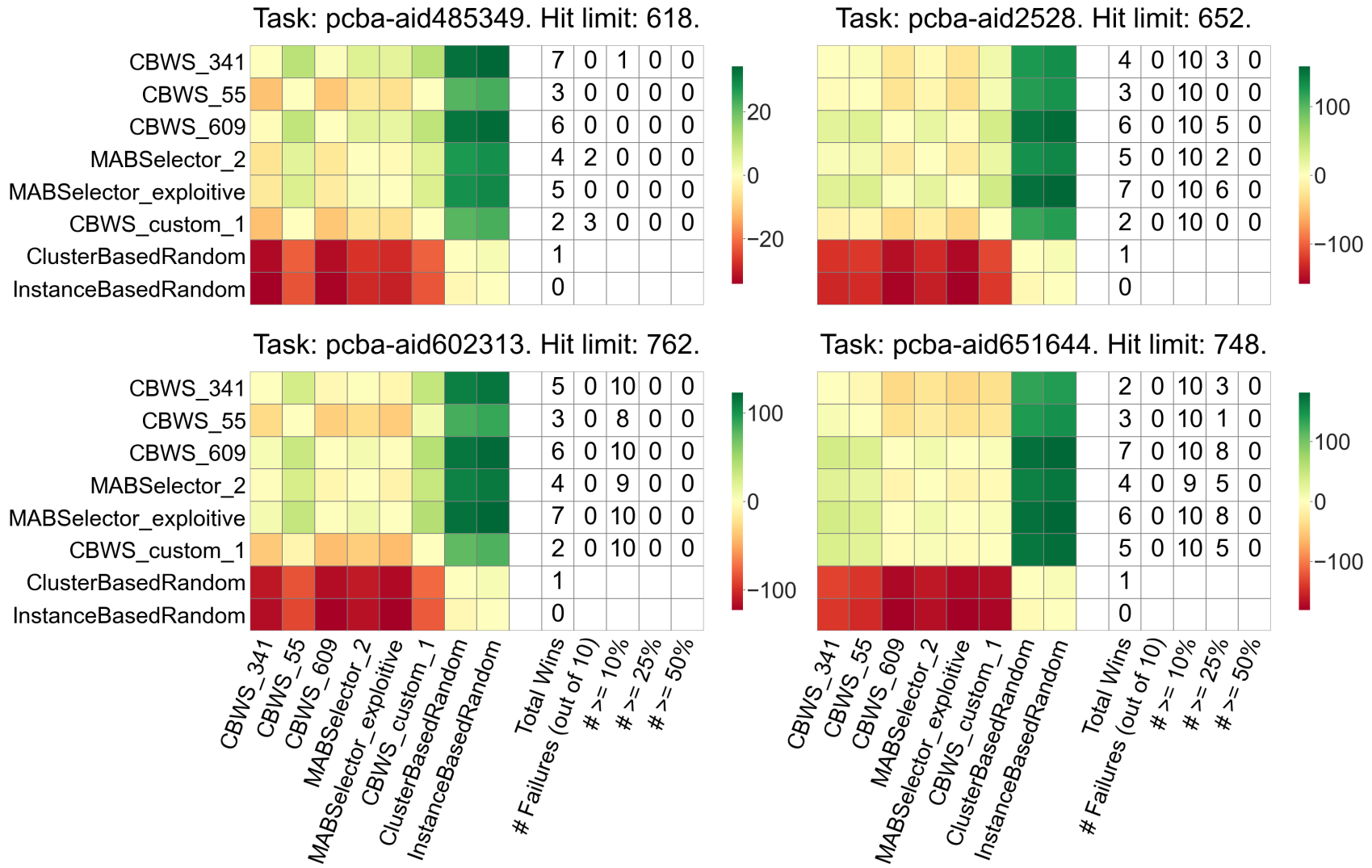


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(9 of 26 cont.)*

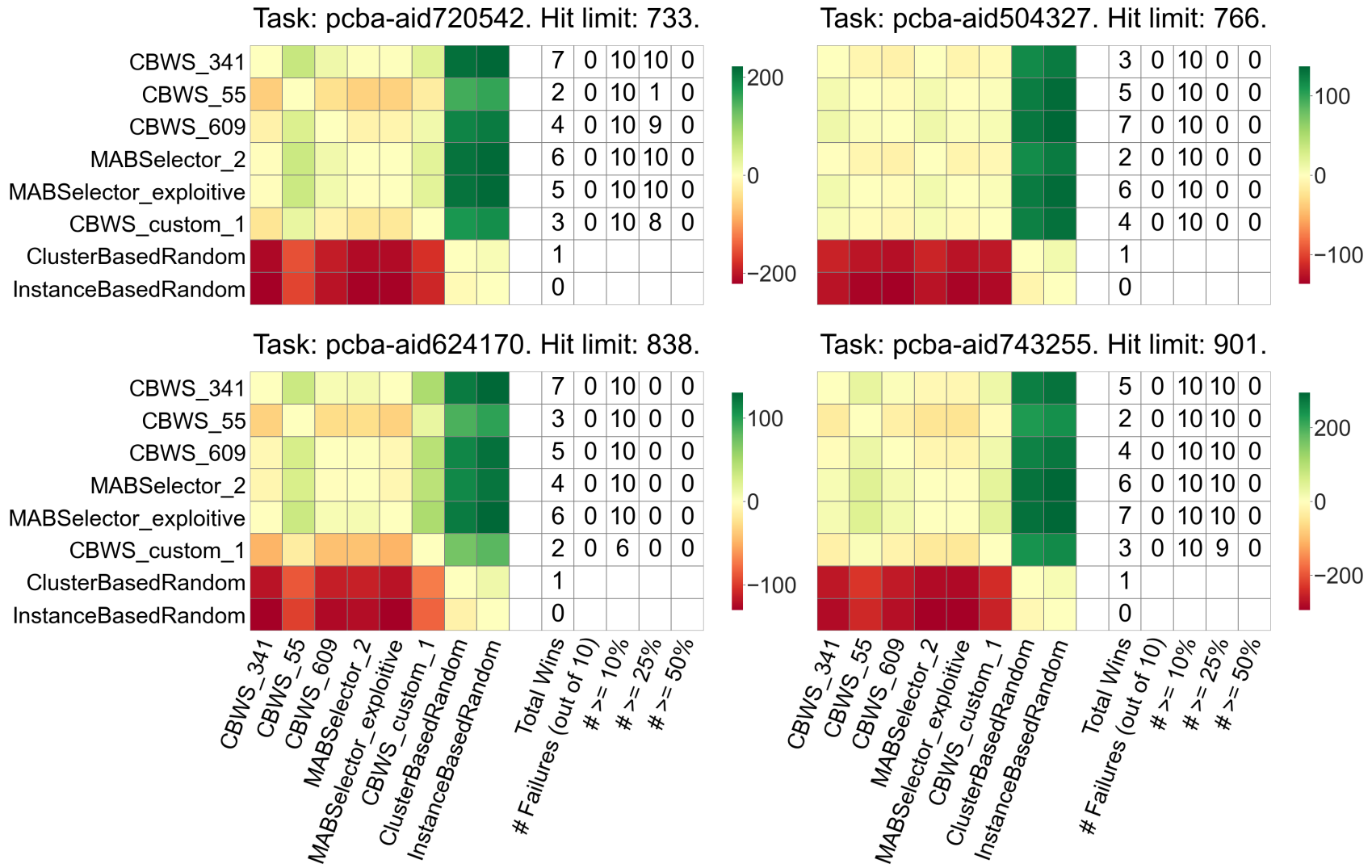


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(10 of 26 cont.)*

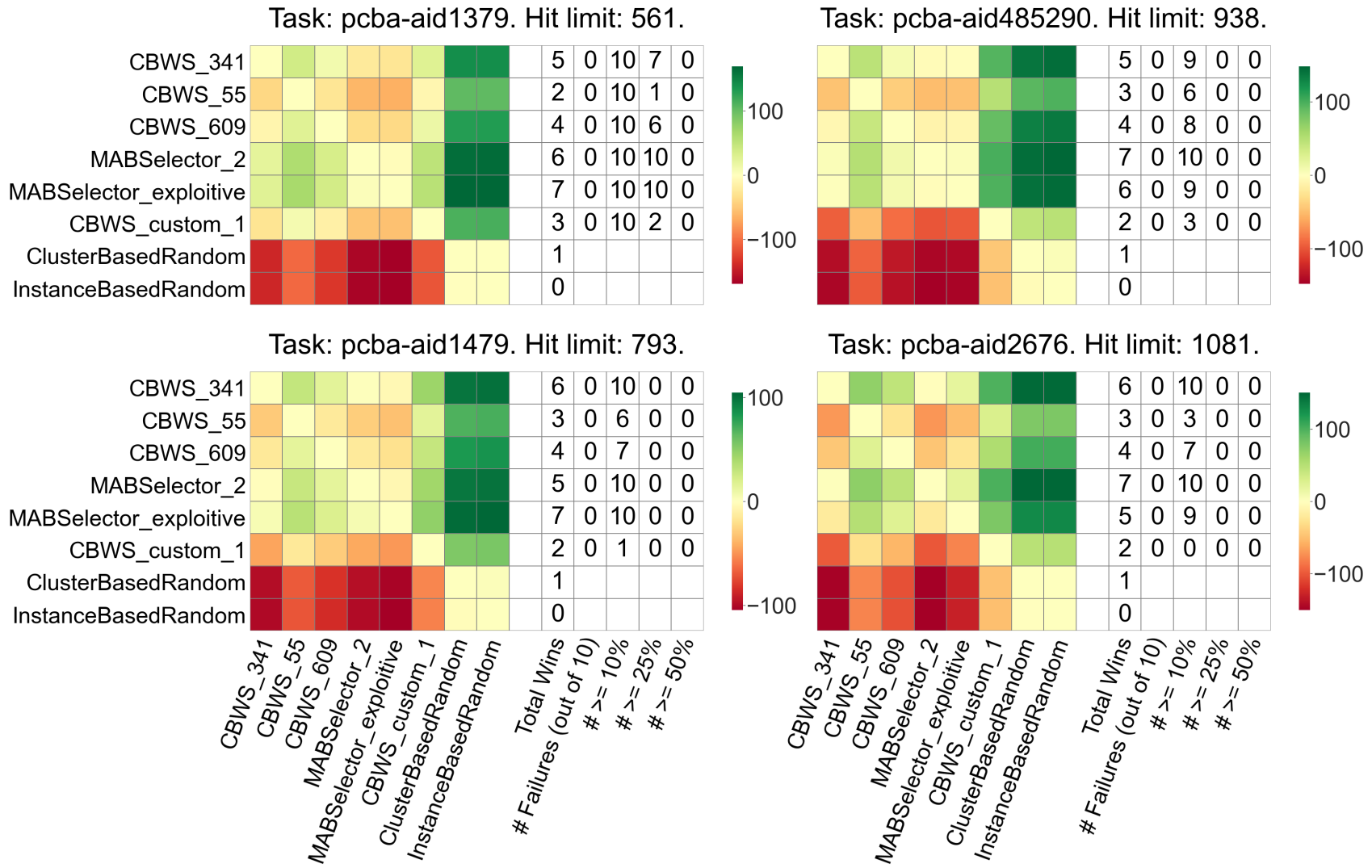


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(11 of 26 cont.)*

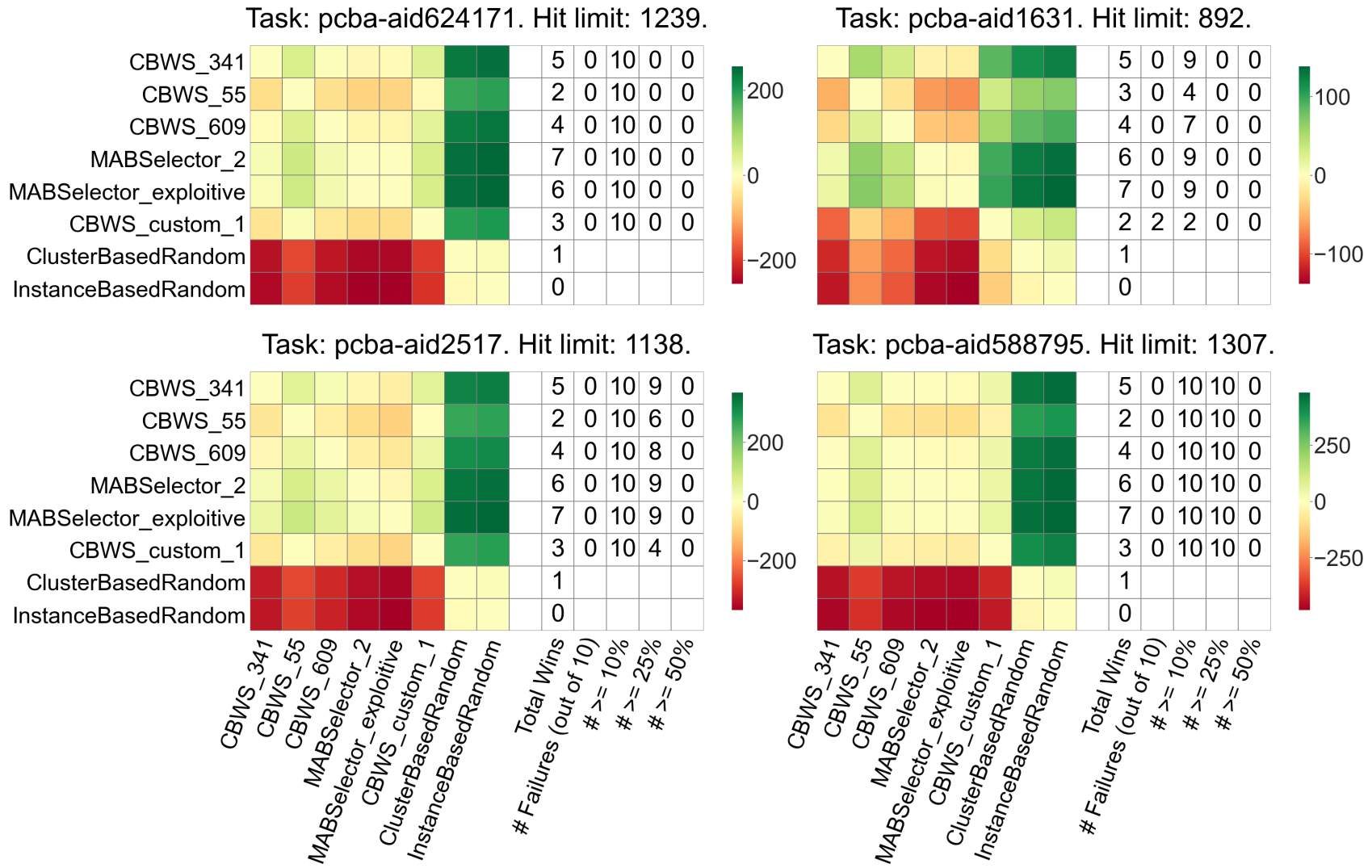


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(12 of 26 cont.)*

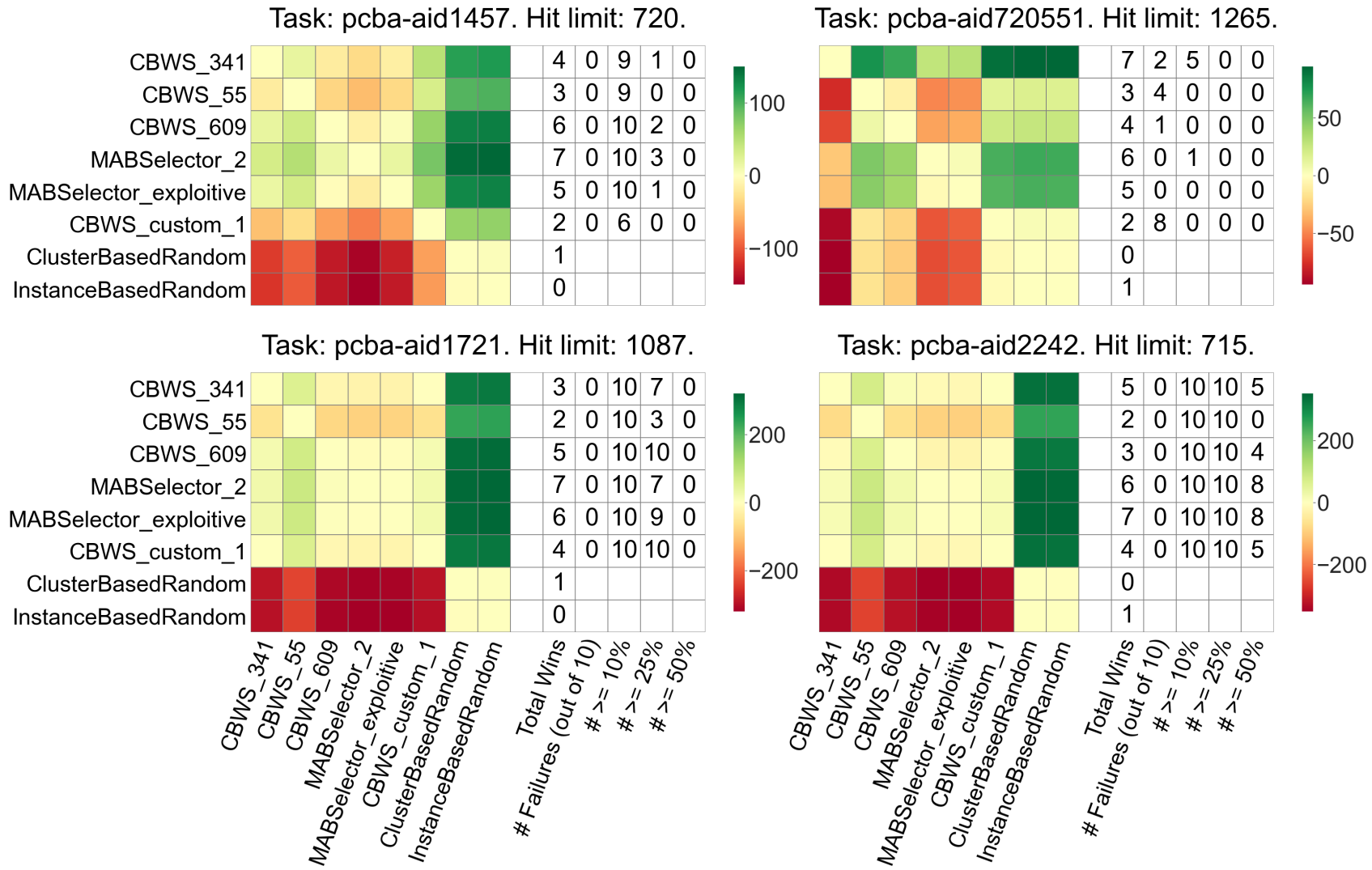


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(13 of 26 cont.)*

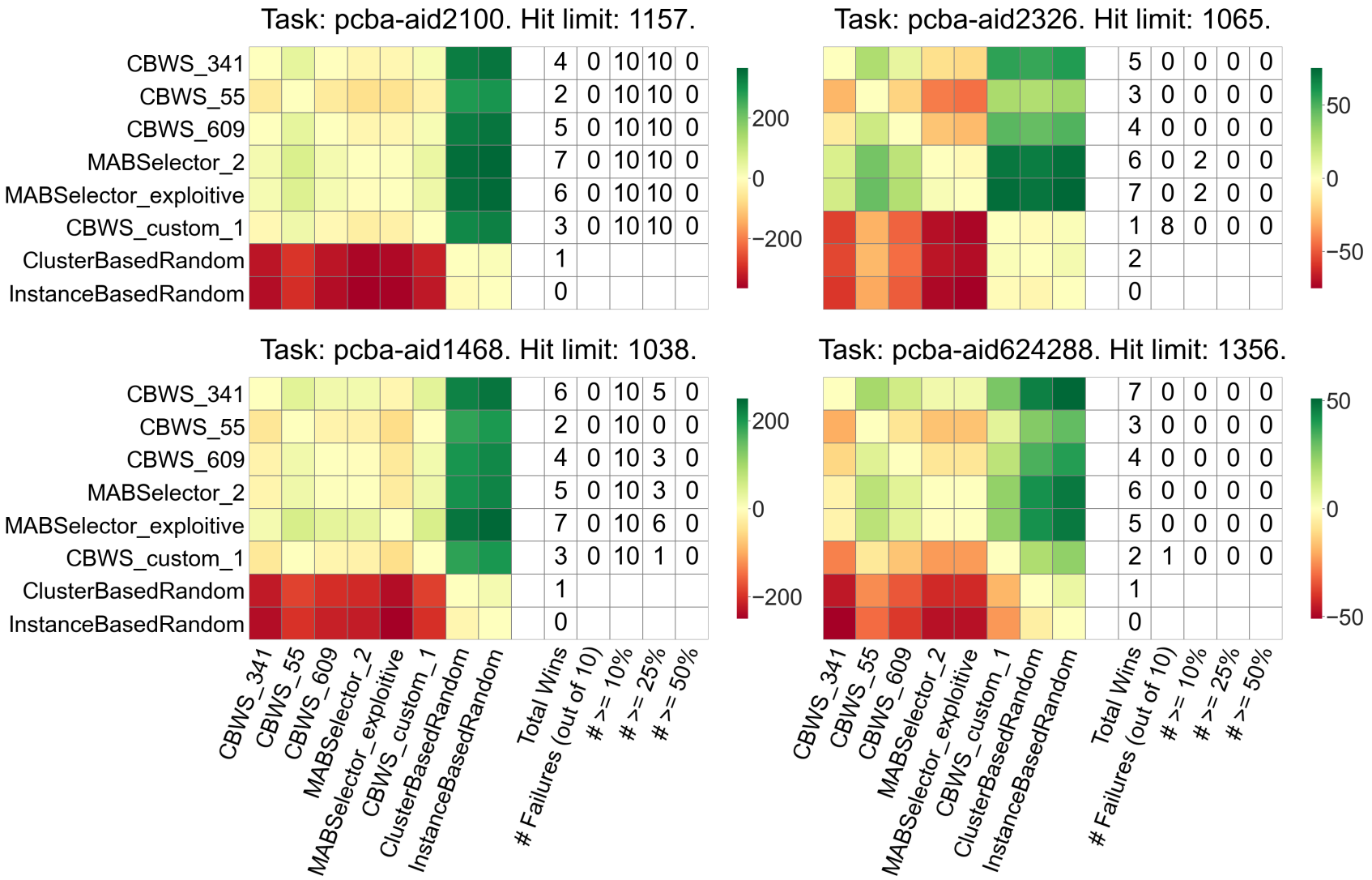


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(14 of 26 cont.)*

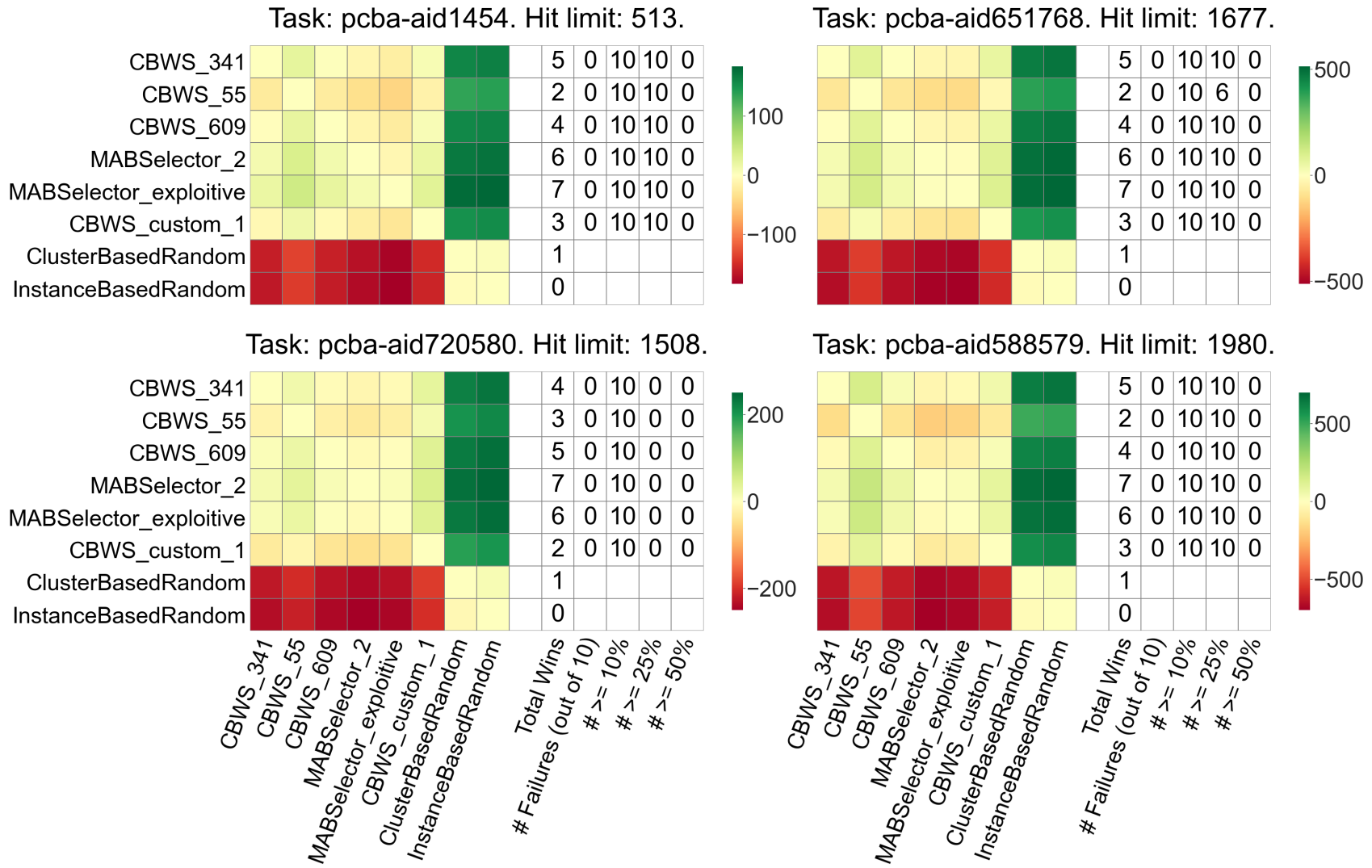


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(15 of 26 cont.)*

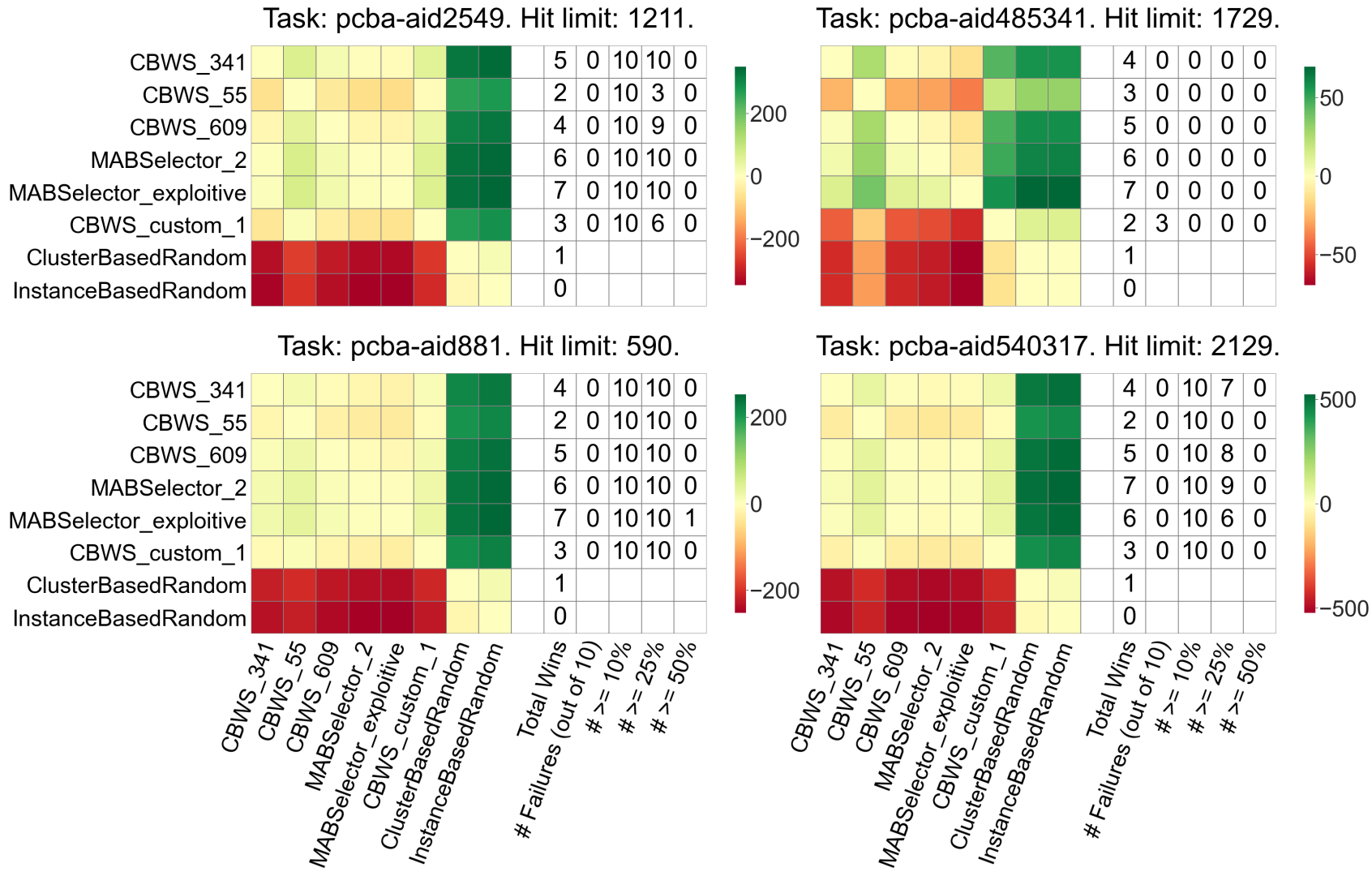


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(16 of 26 cont.)*

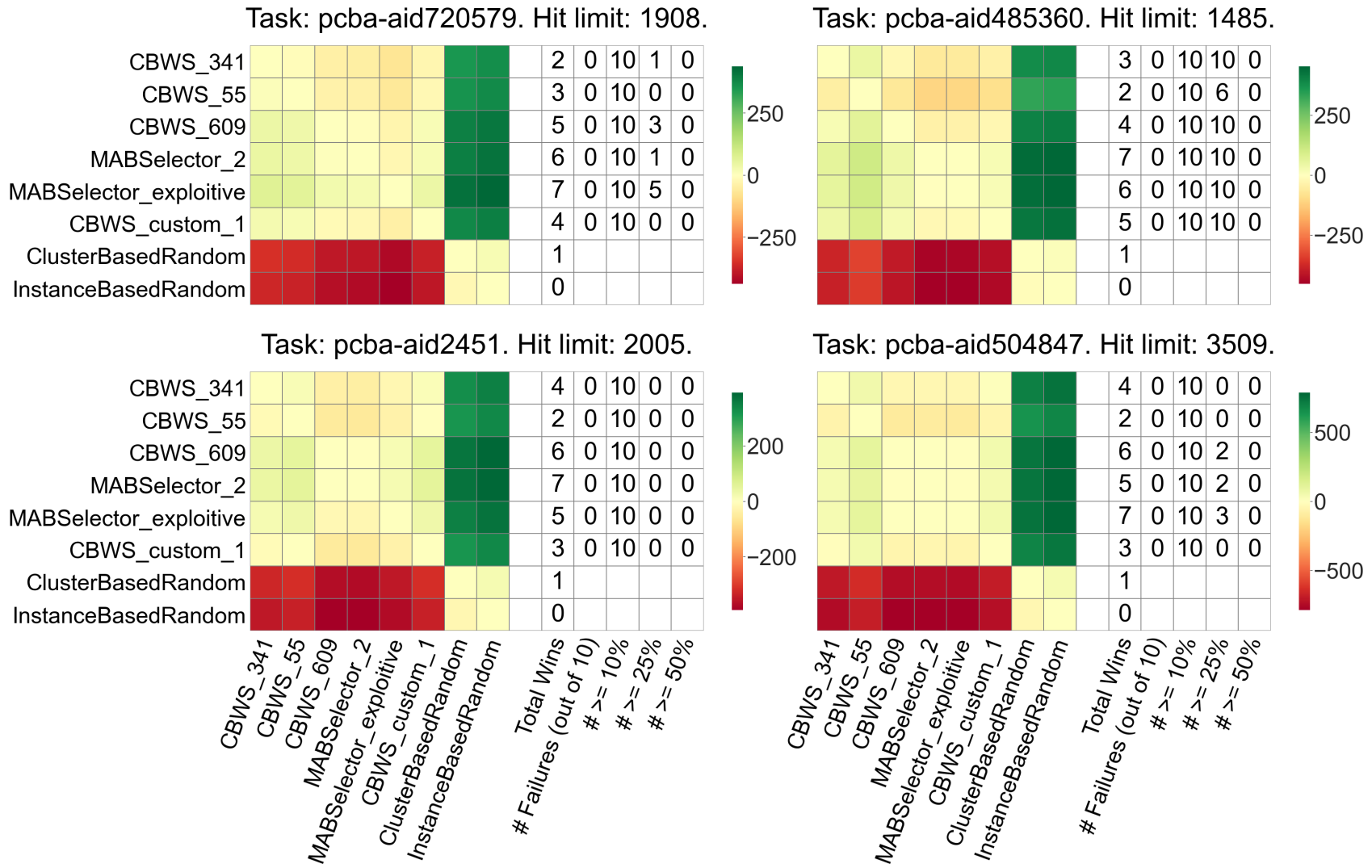


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(17 of 26 cont.)*

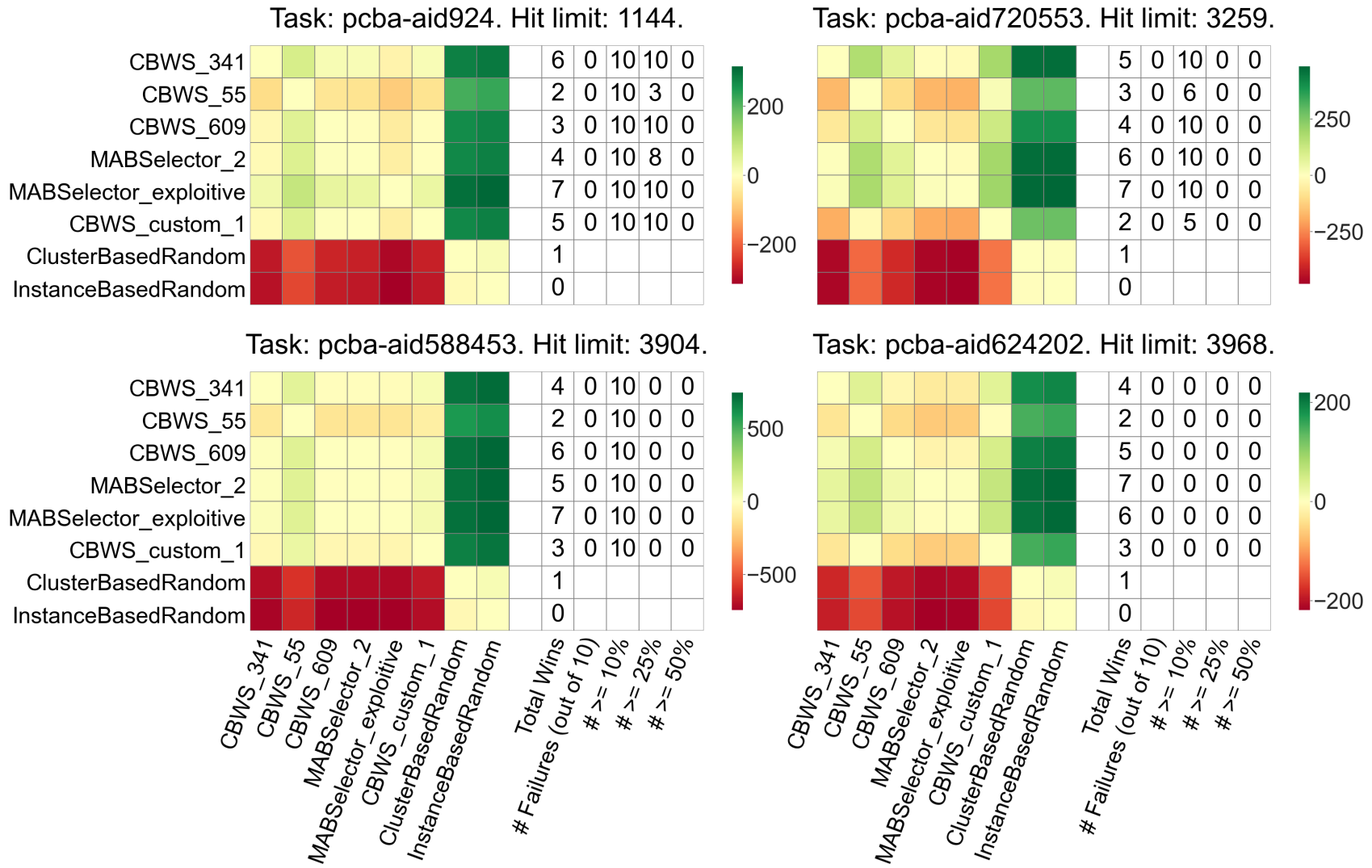


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(18 of 26 cont.)*

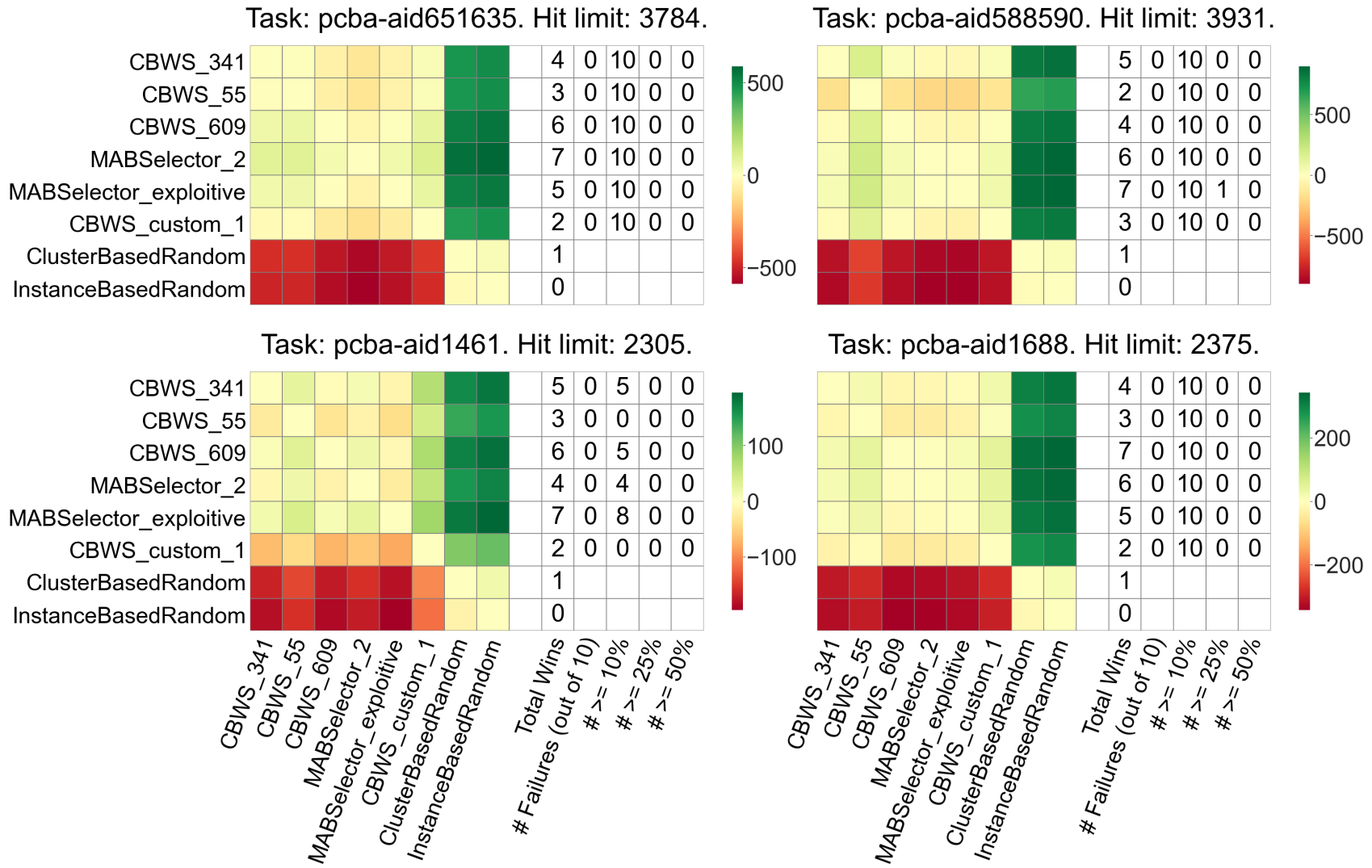


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(19 of 26 cont.)*

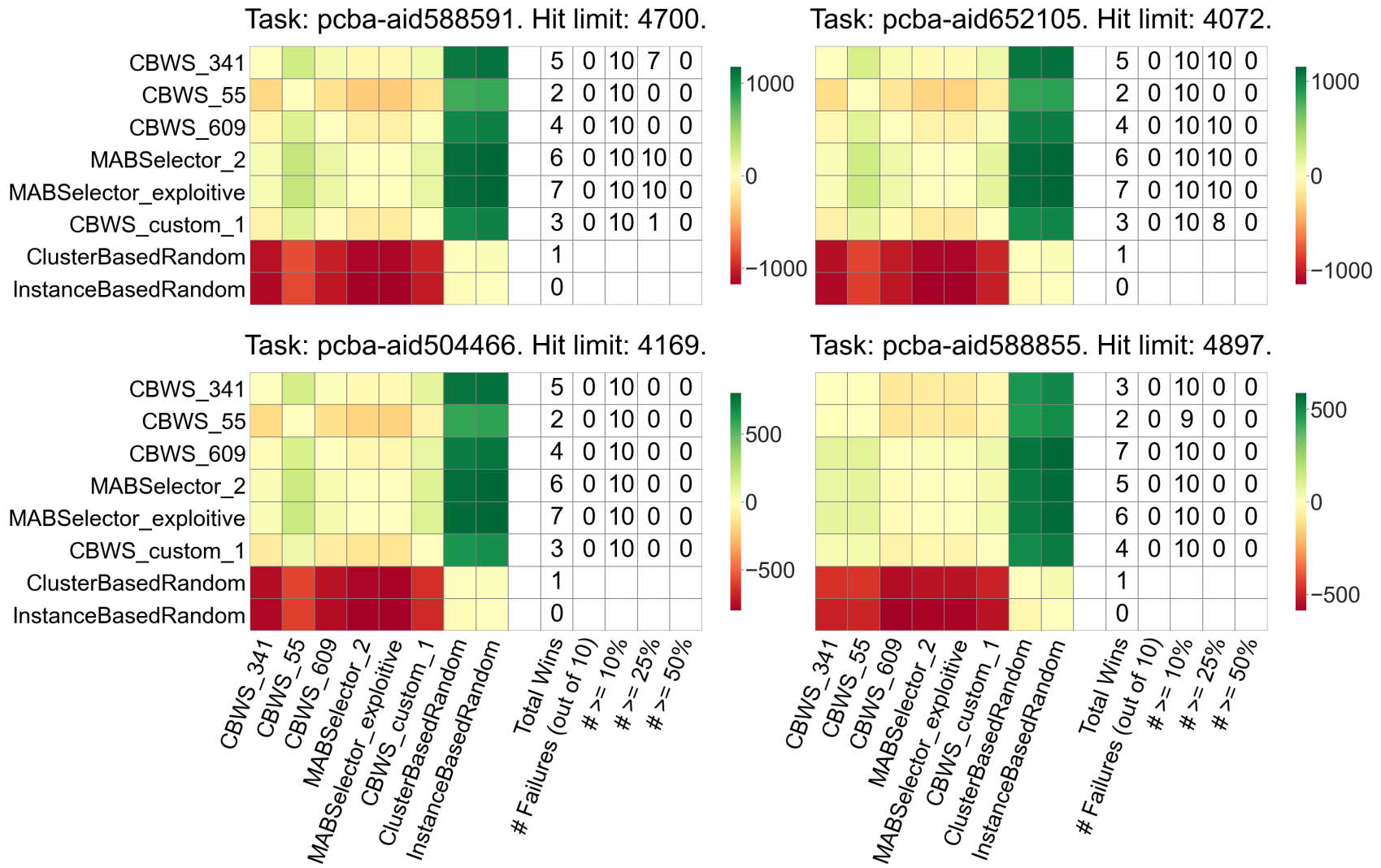


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(20 of 26 cont.)*

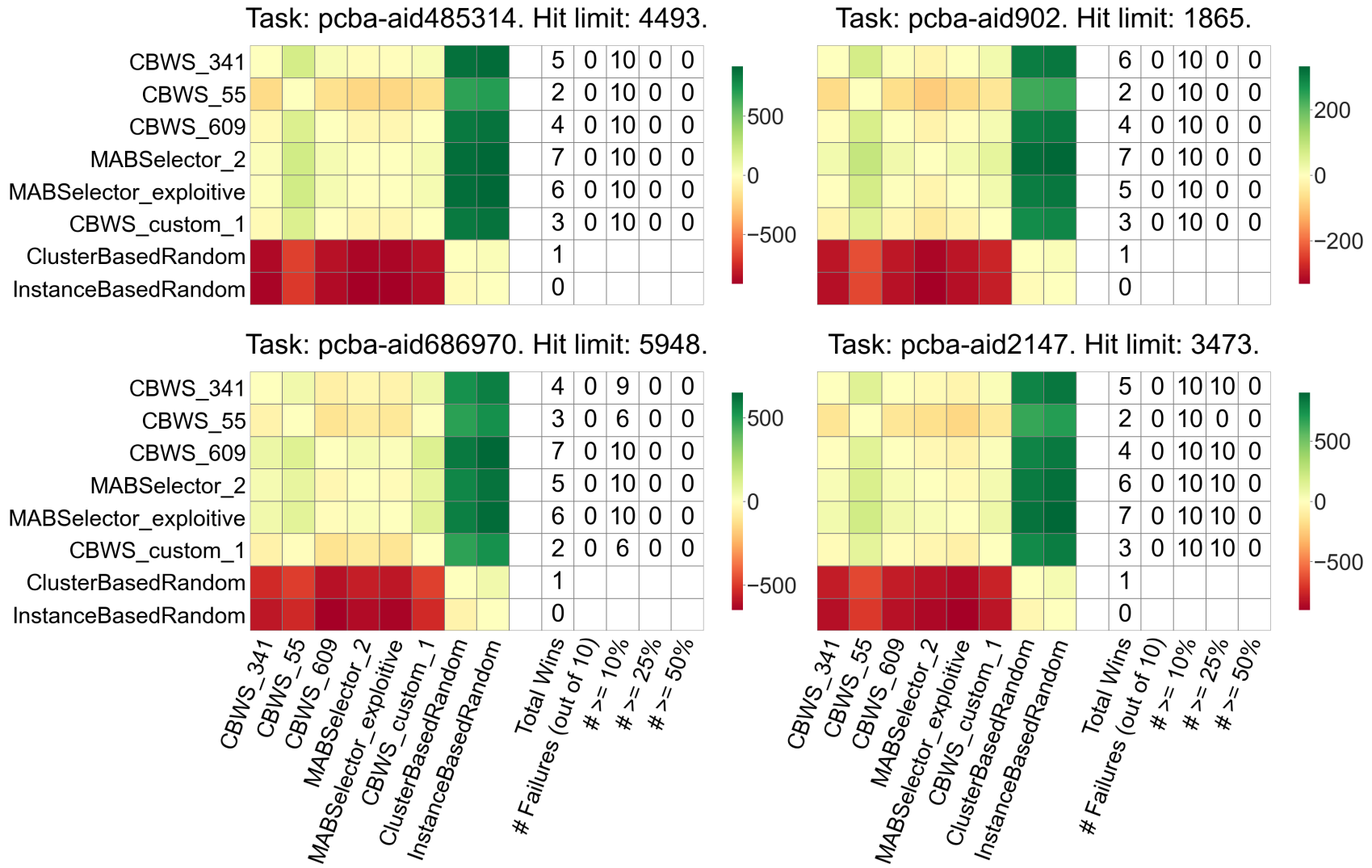


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(21 of 26 cont.)*

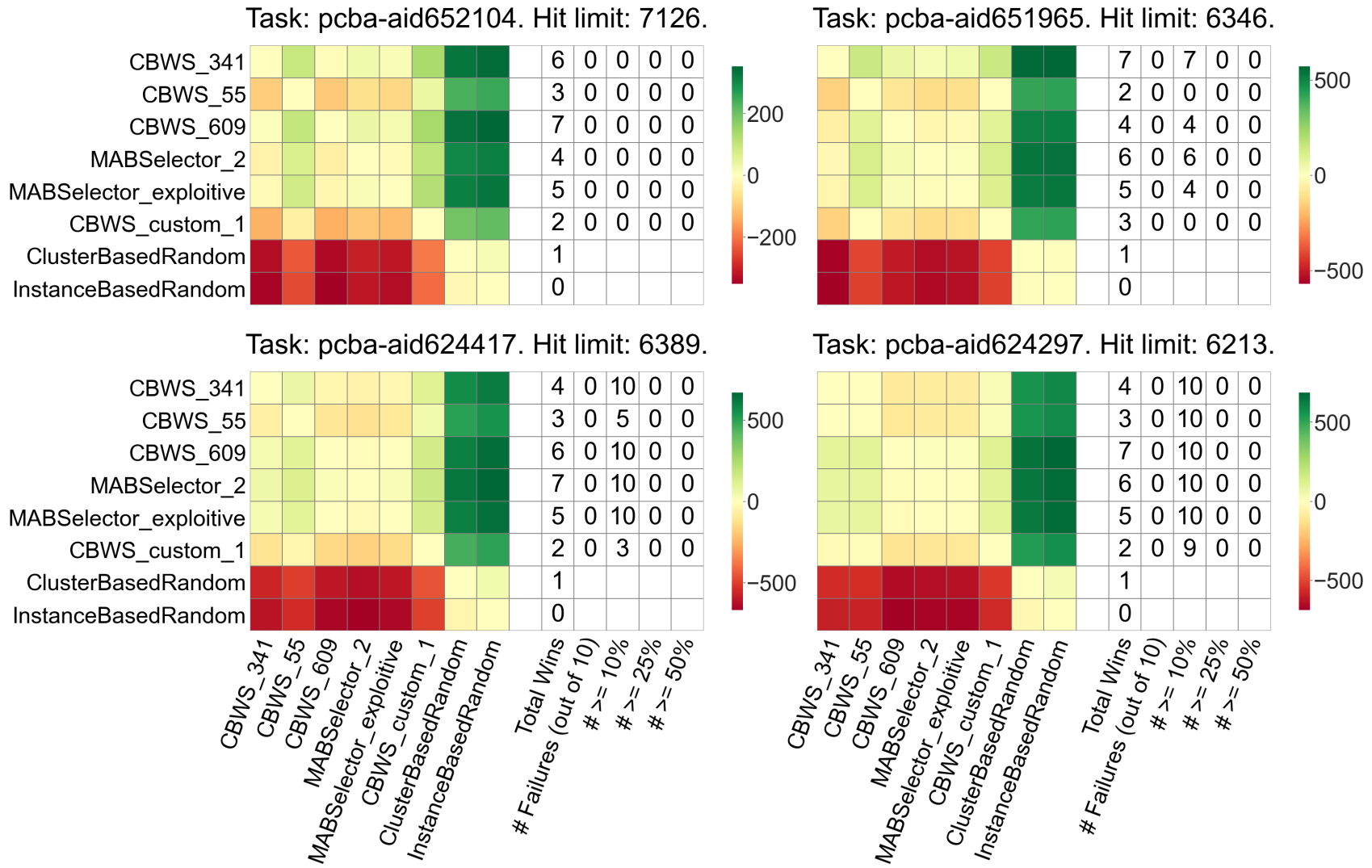


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(22 of 26 cont.)*

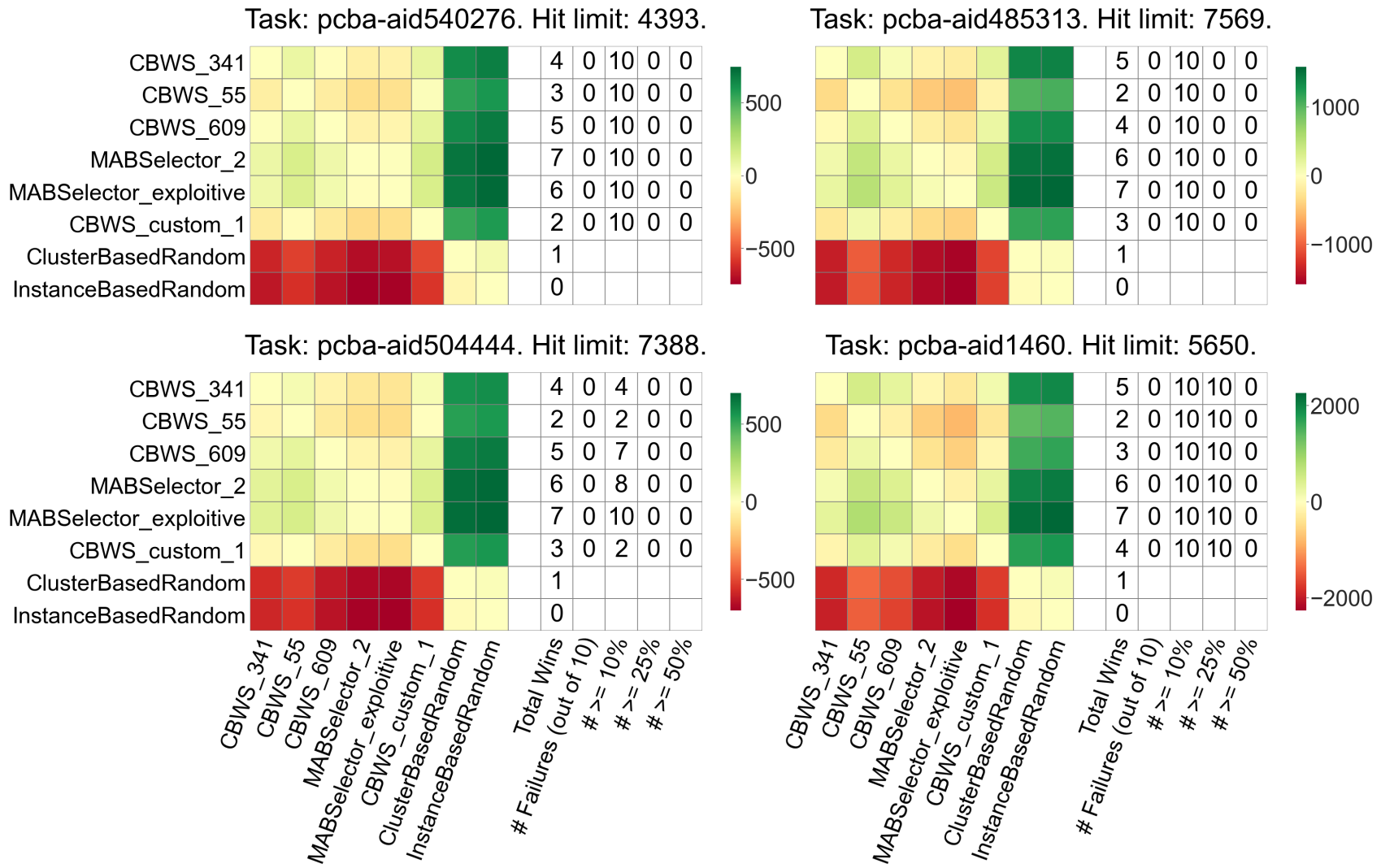


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(23 of 26 cont.)*

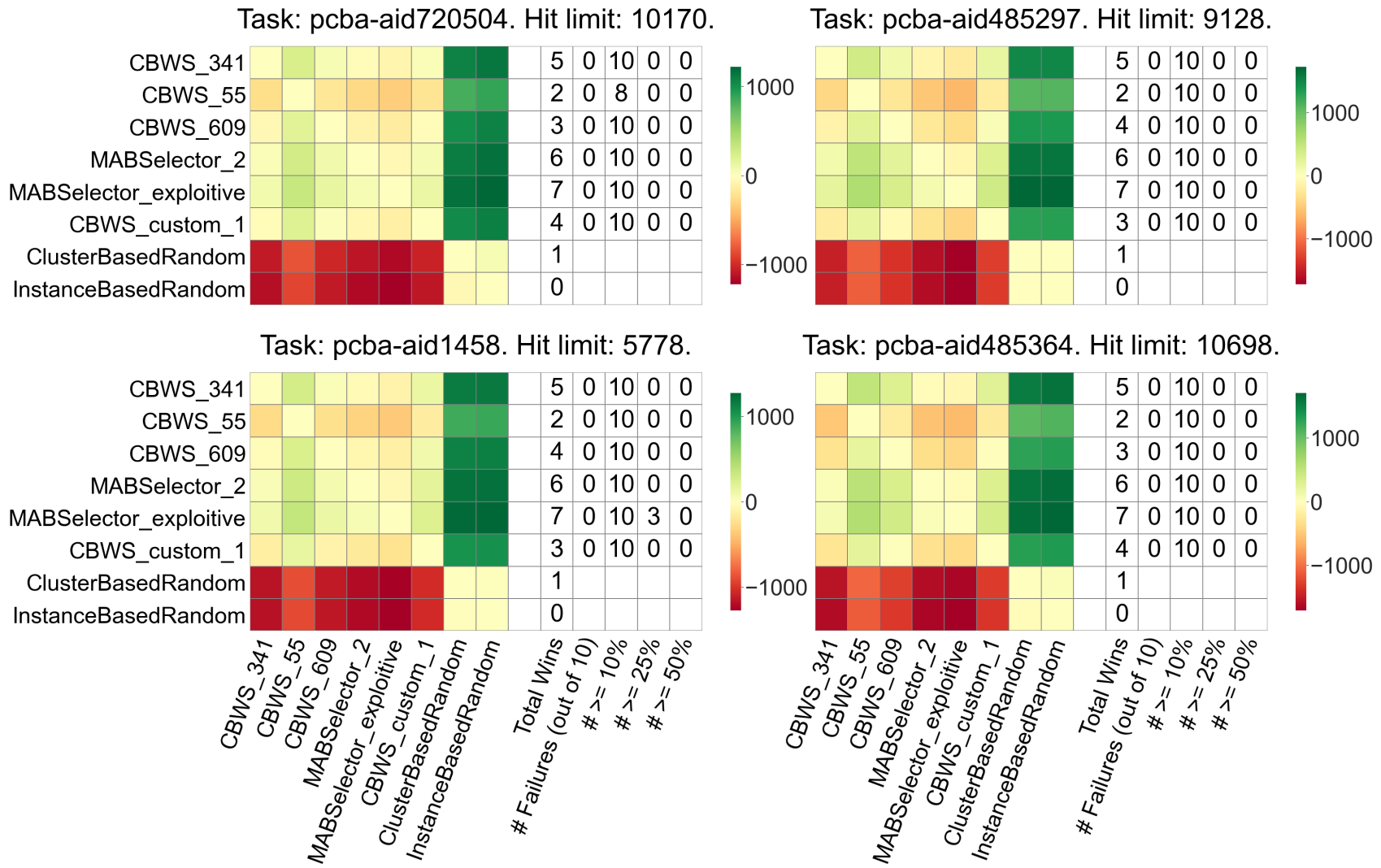


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(24 of 26 cont.)*

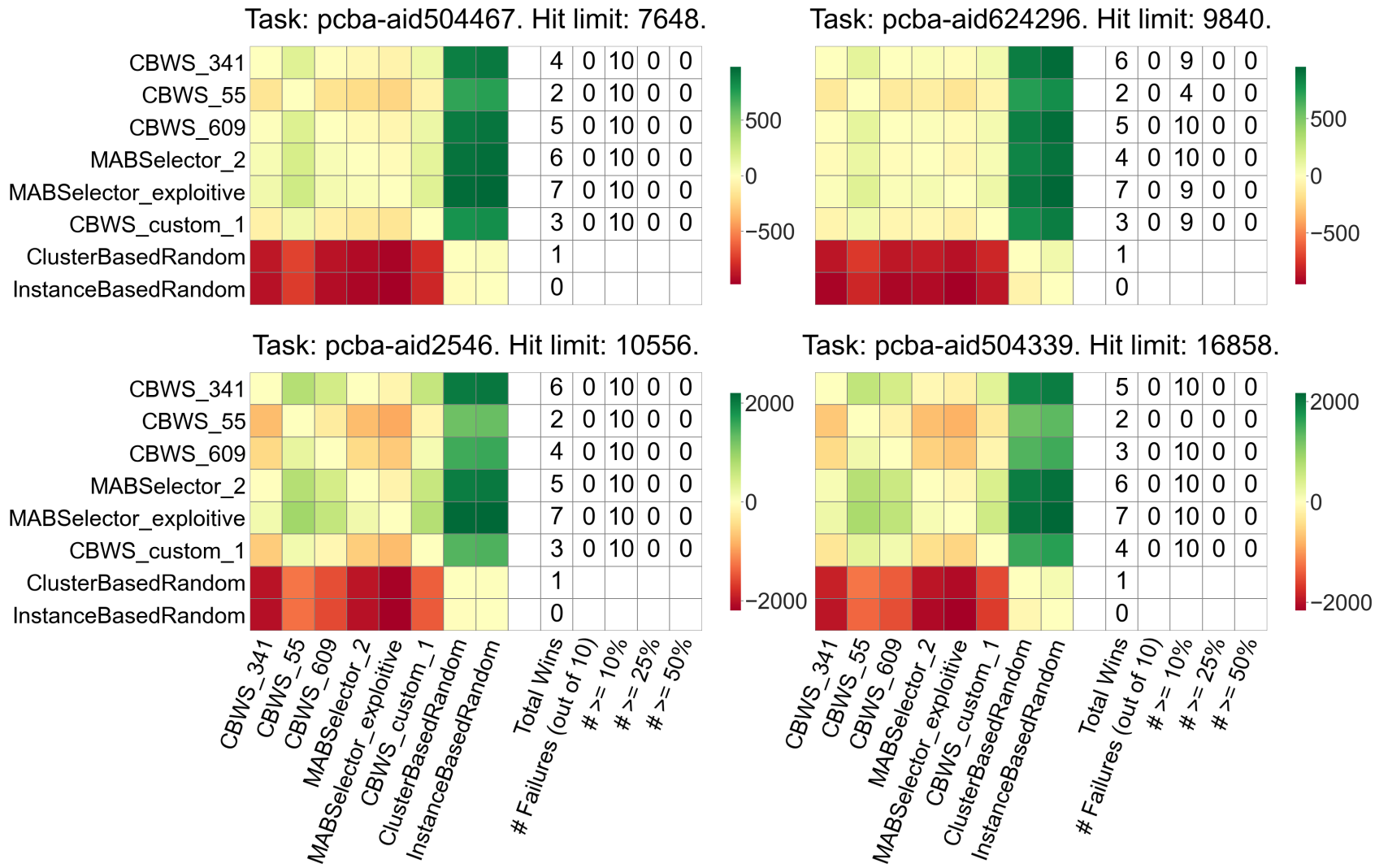


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(25 of 26 cont.)*

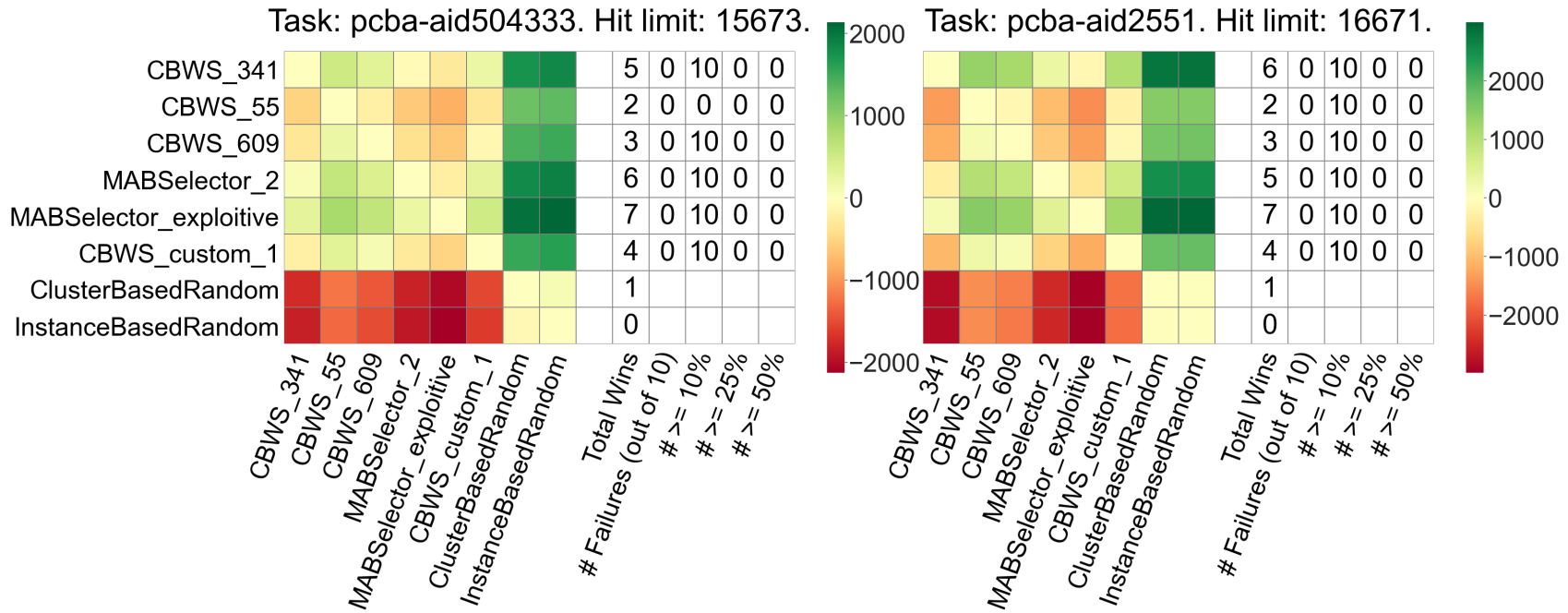


Figure B.1: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(26 of 26)*

B.2 Experiment 3.1: Per Task Total Unique Hits CEM

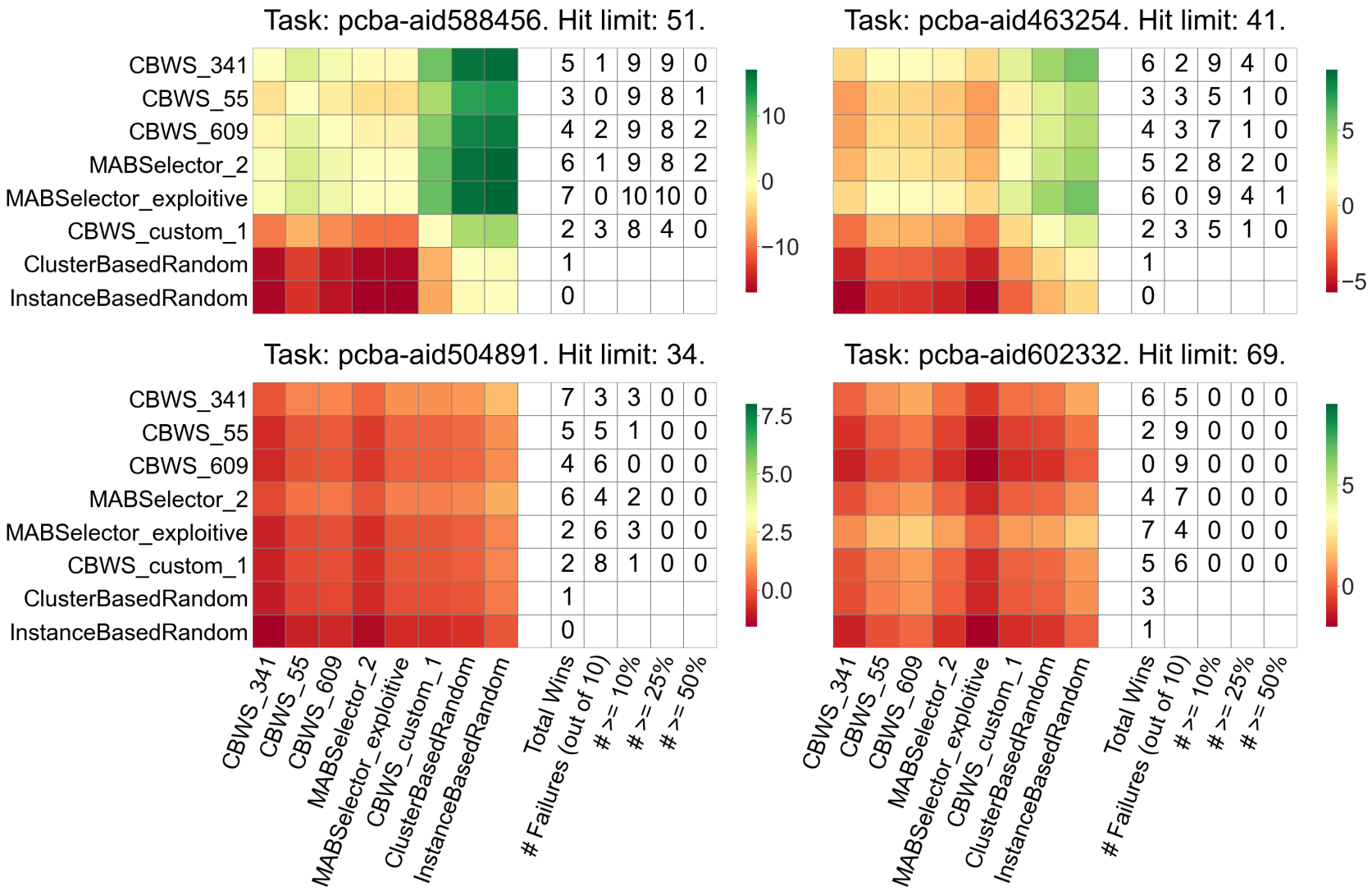


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(1 of 26 cont.)*

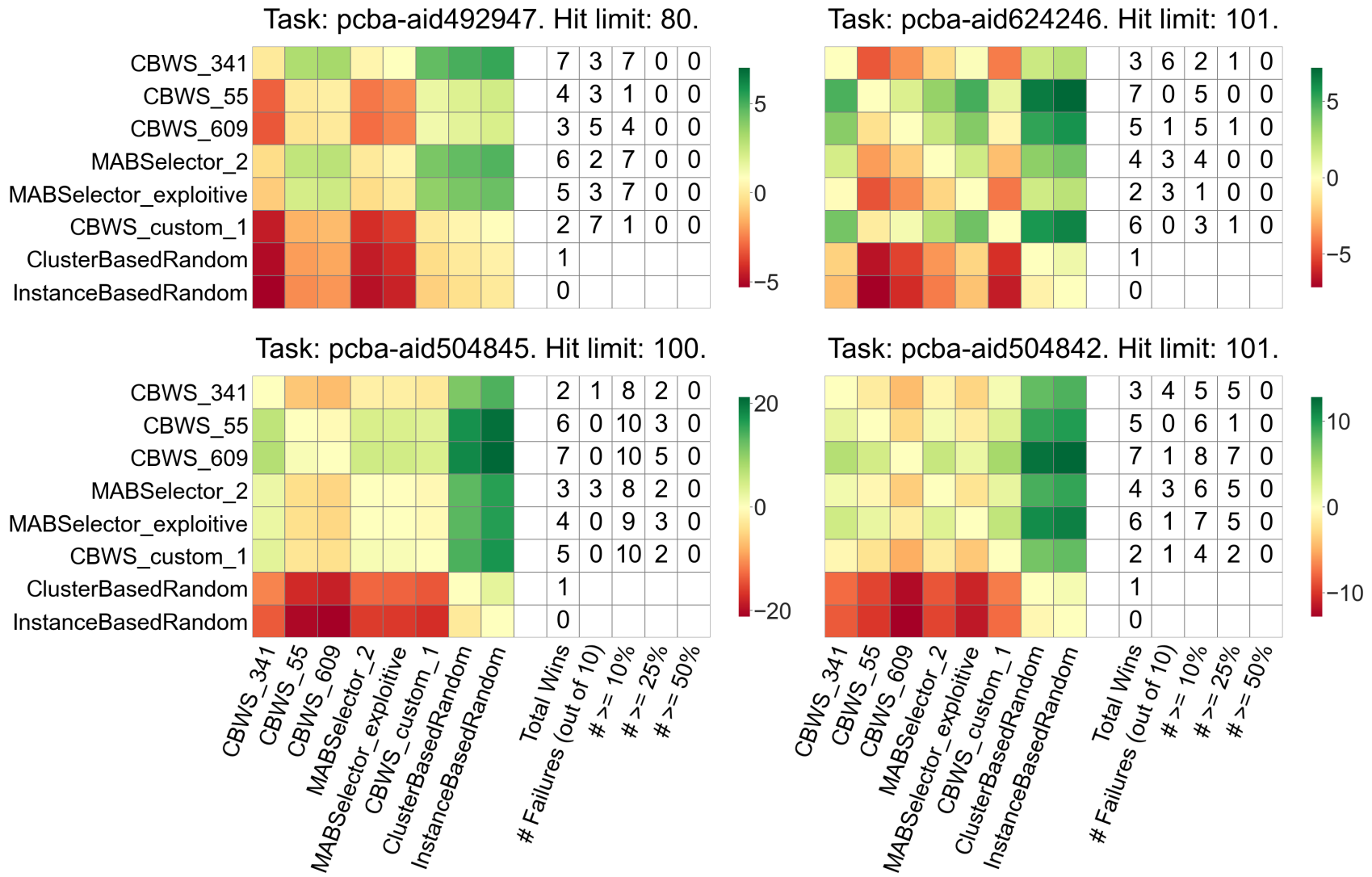


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(2 of 26 cont.)*

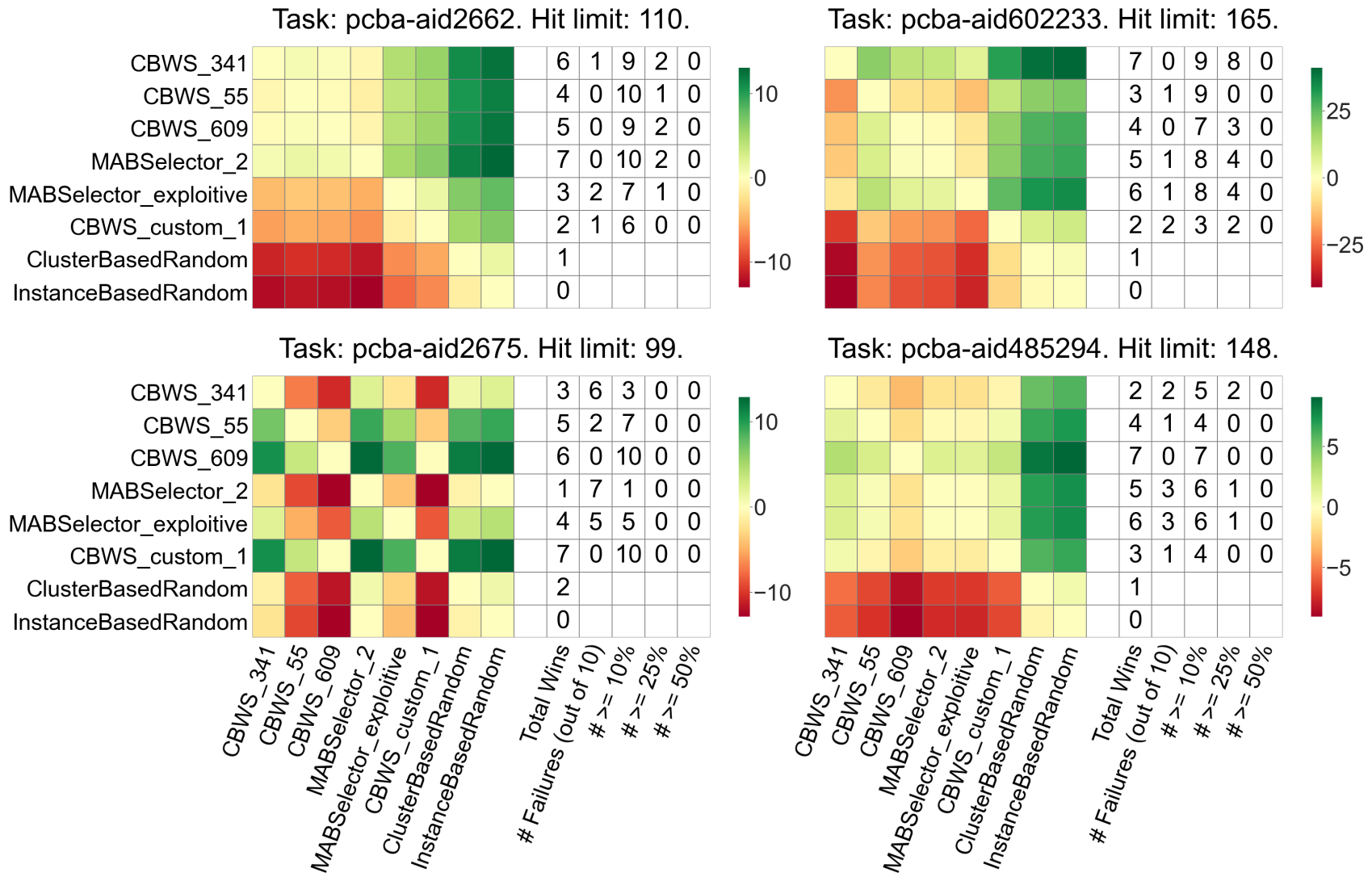


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(3 of 26 cont.)*

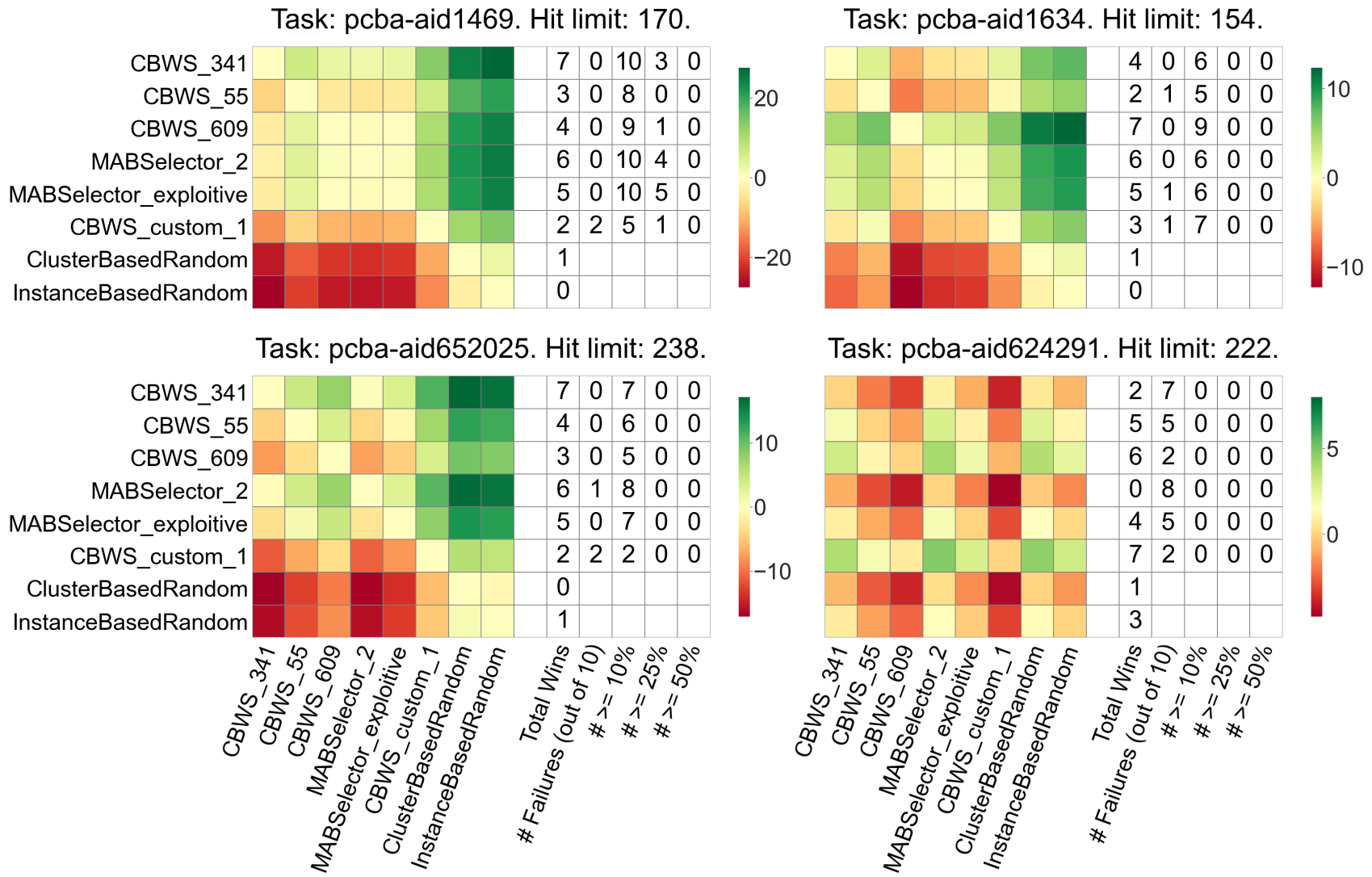


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(4 of 26 cont.)*

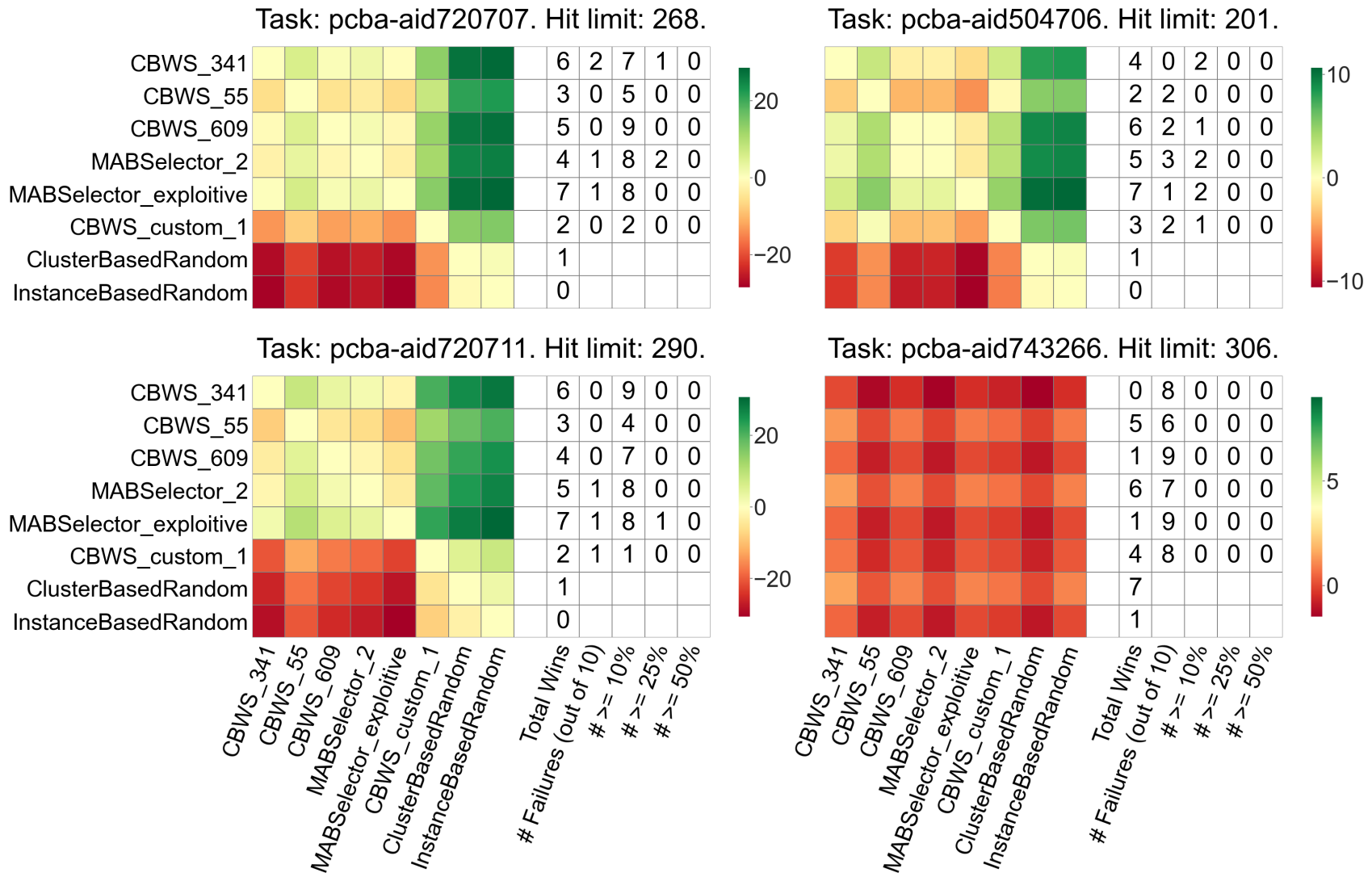


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(5 of 26 cont.)*

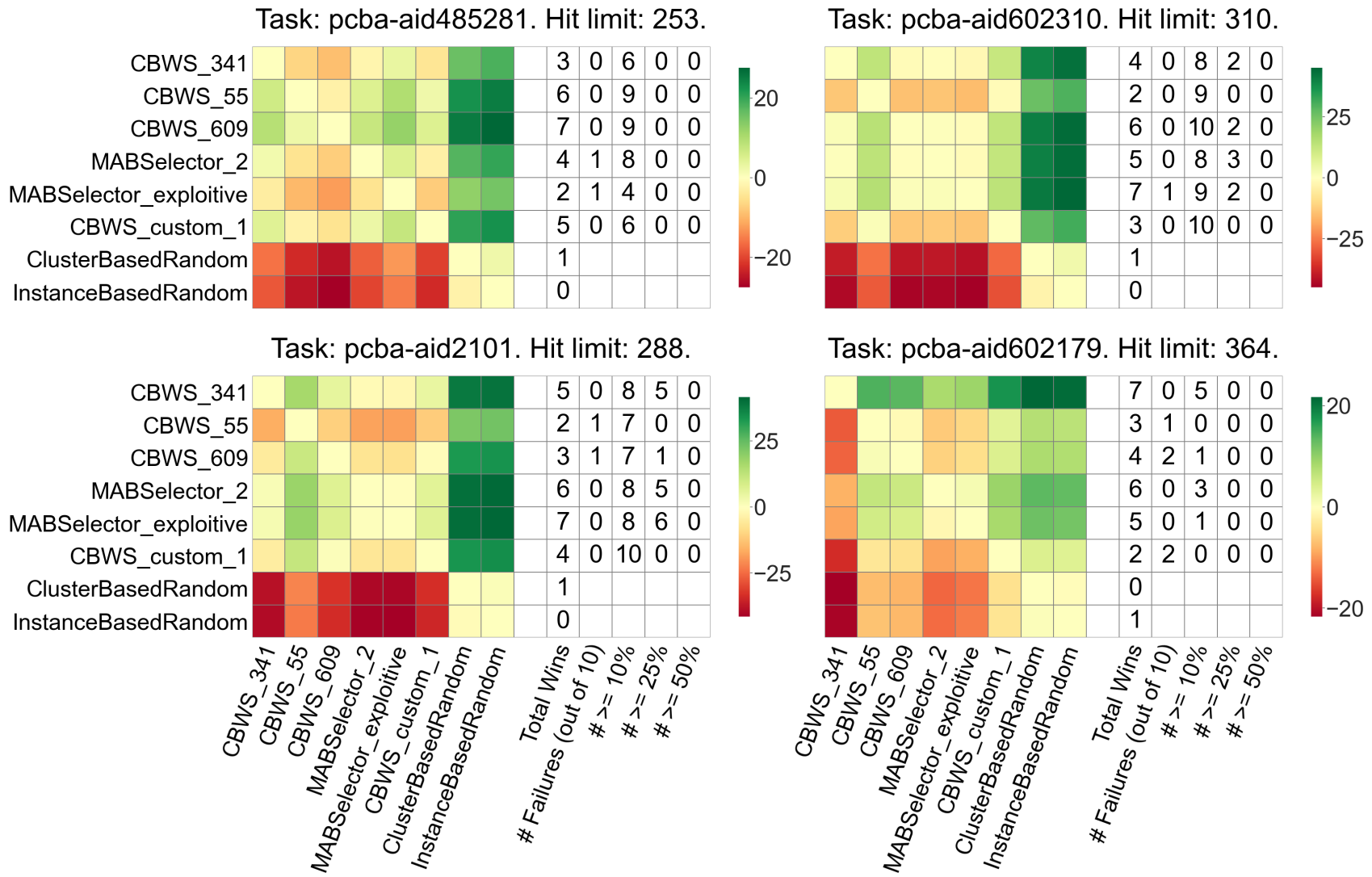


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(6 of 26 cont.)*

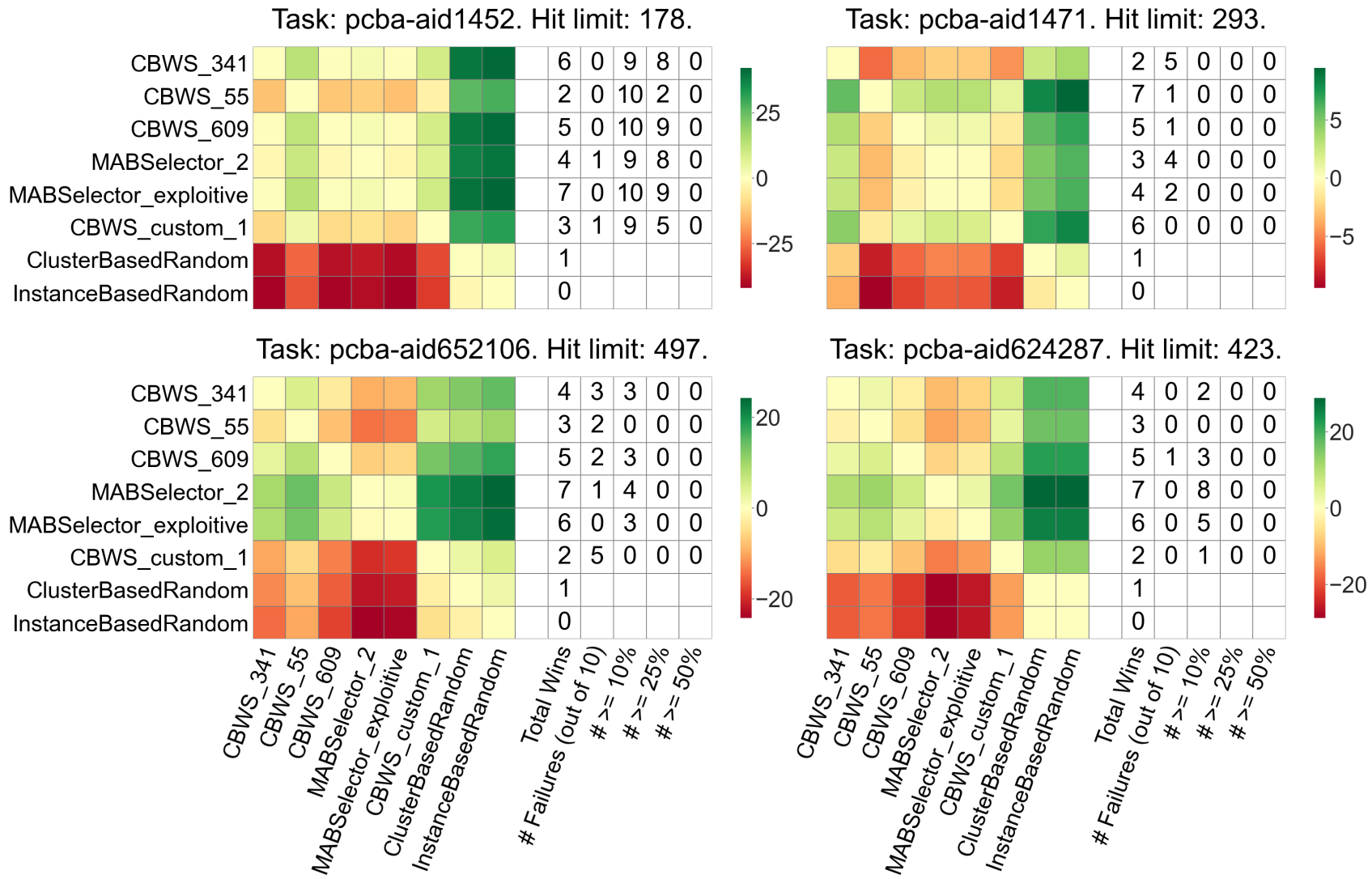


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(7 of 26 cont.)*

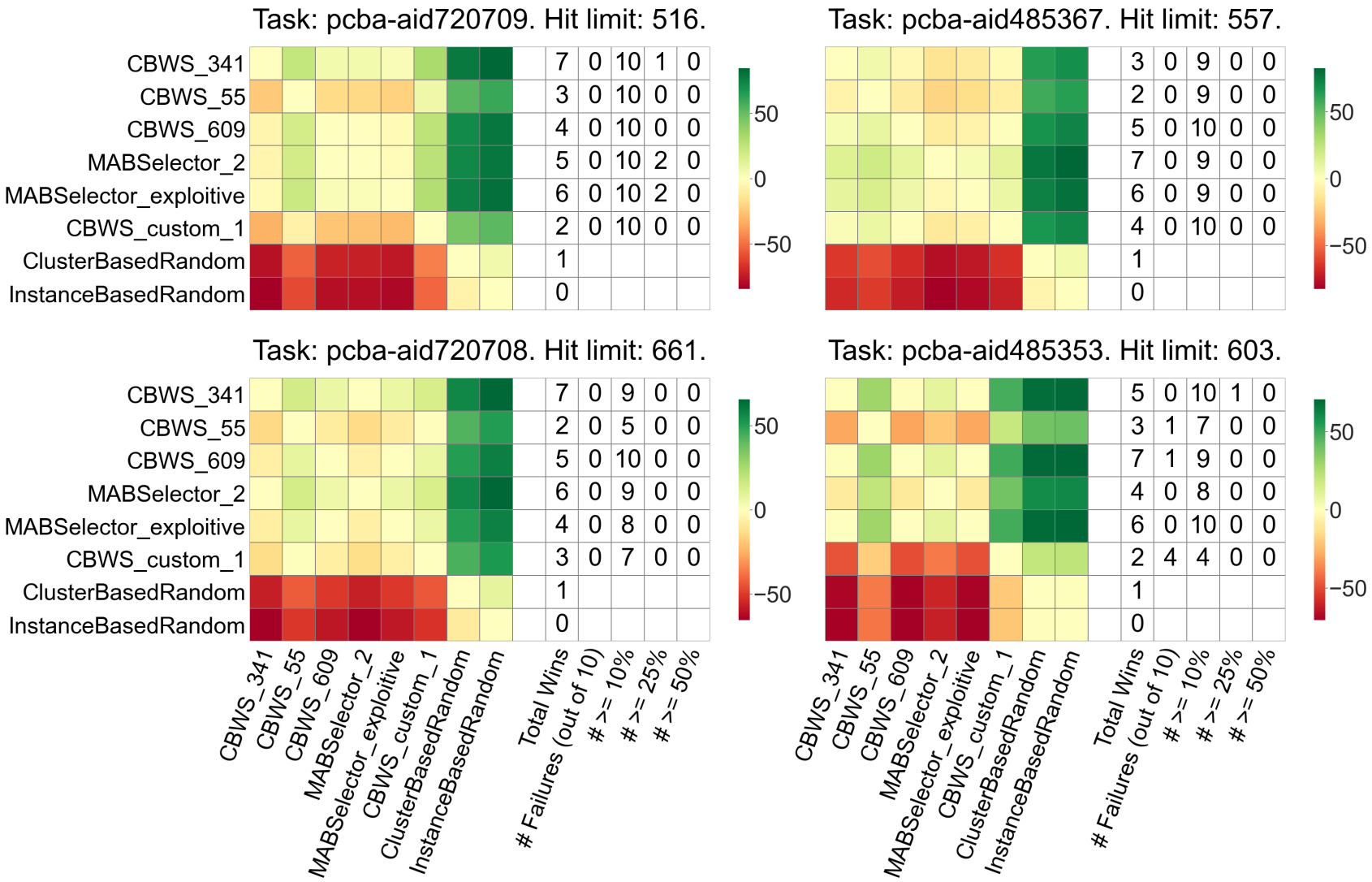


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(8 of 26 cont.)*

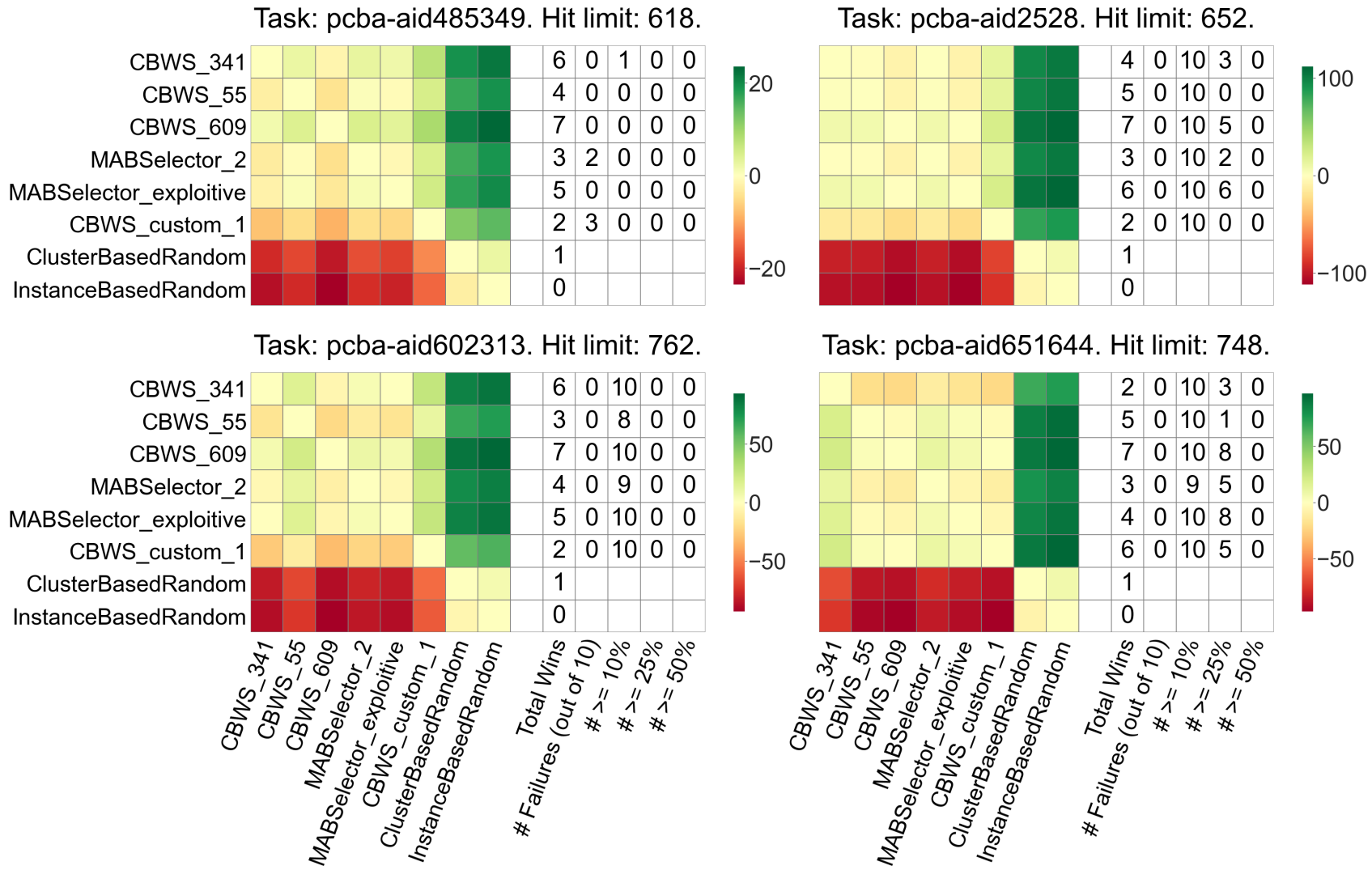


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(9 of 26 cont.)*

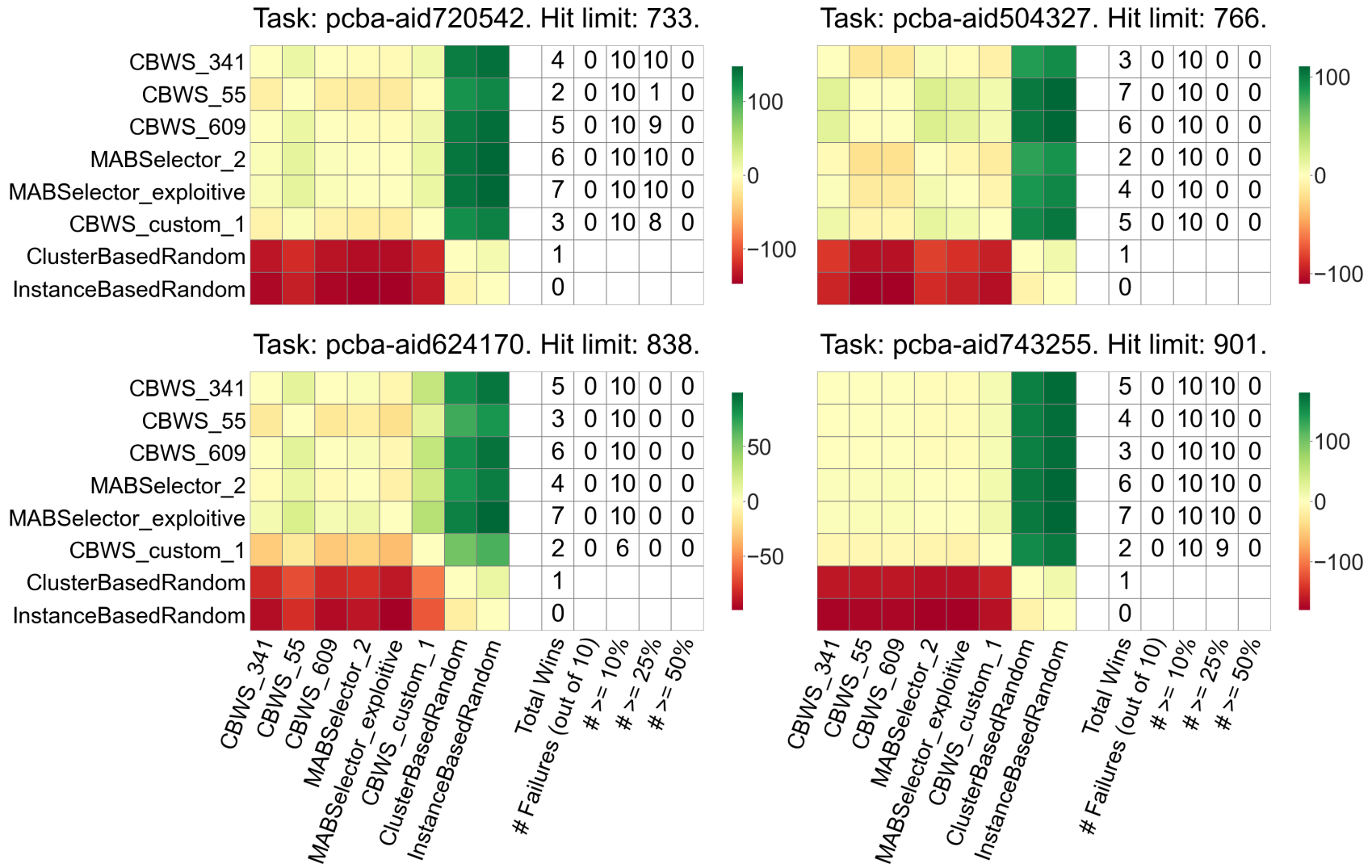


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(10 of 26 cont.)*

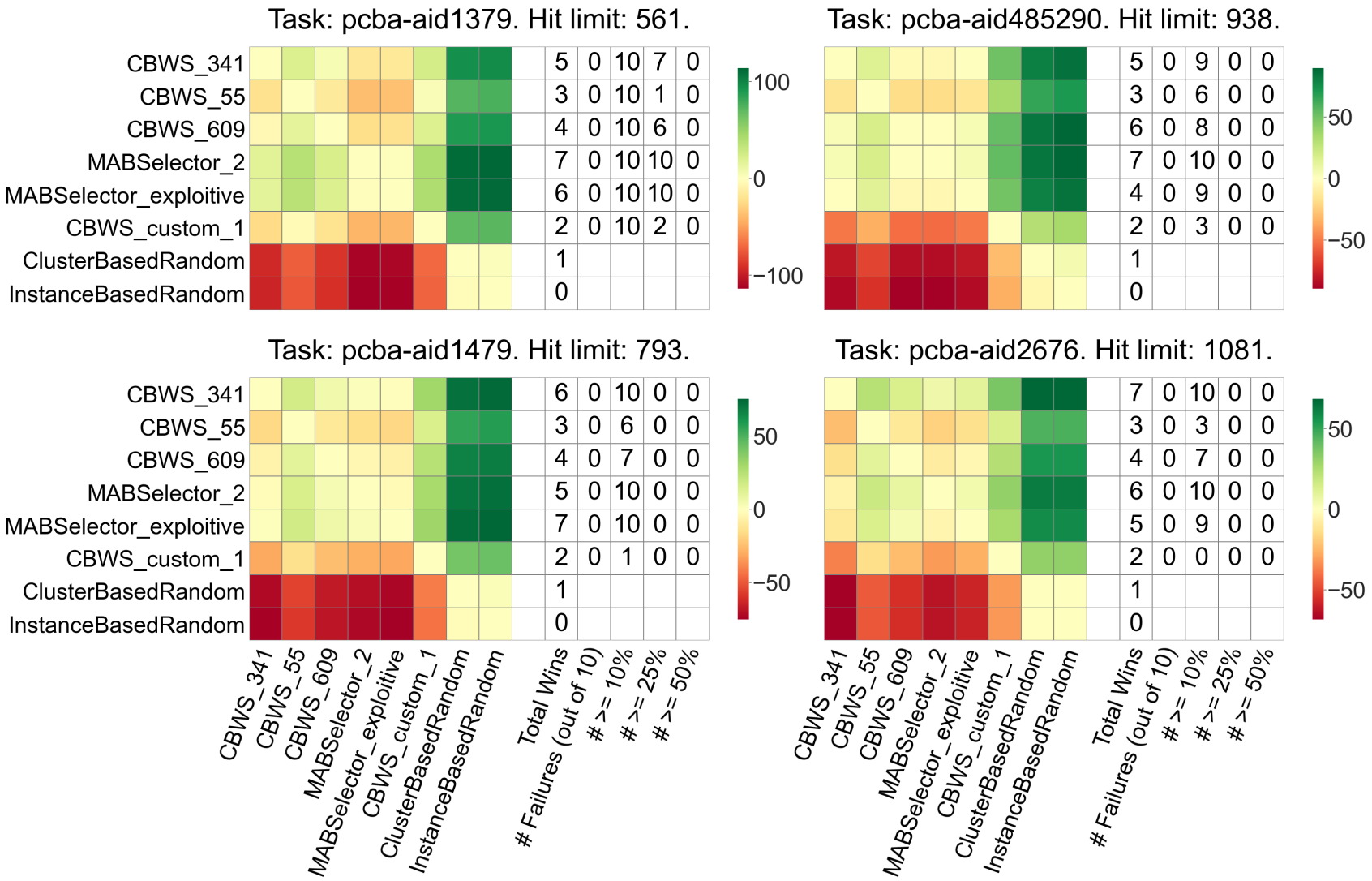


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(11 of 26 cont.)*

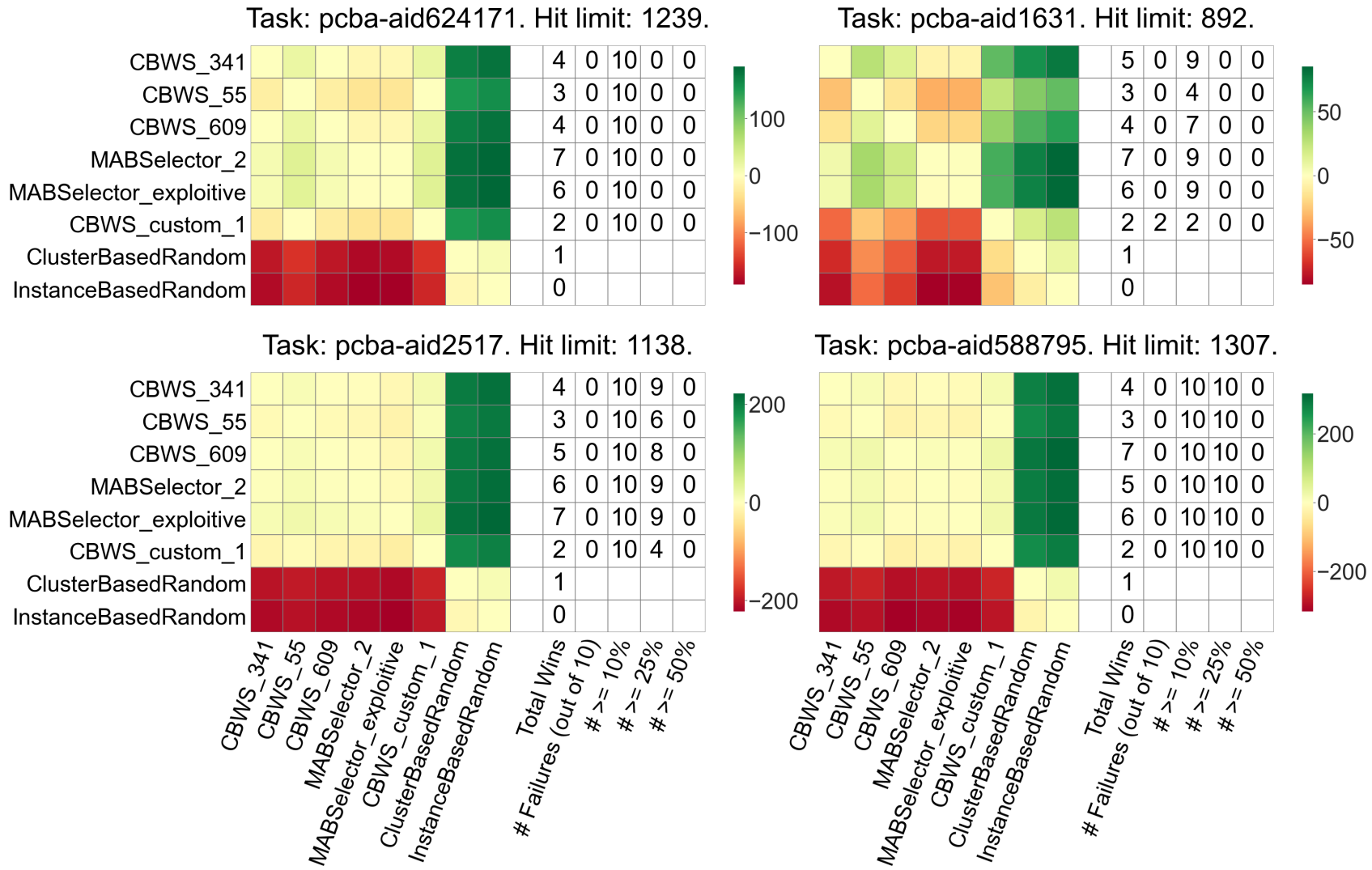


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(12 of 26 cont.)*

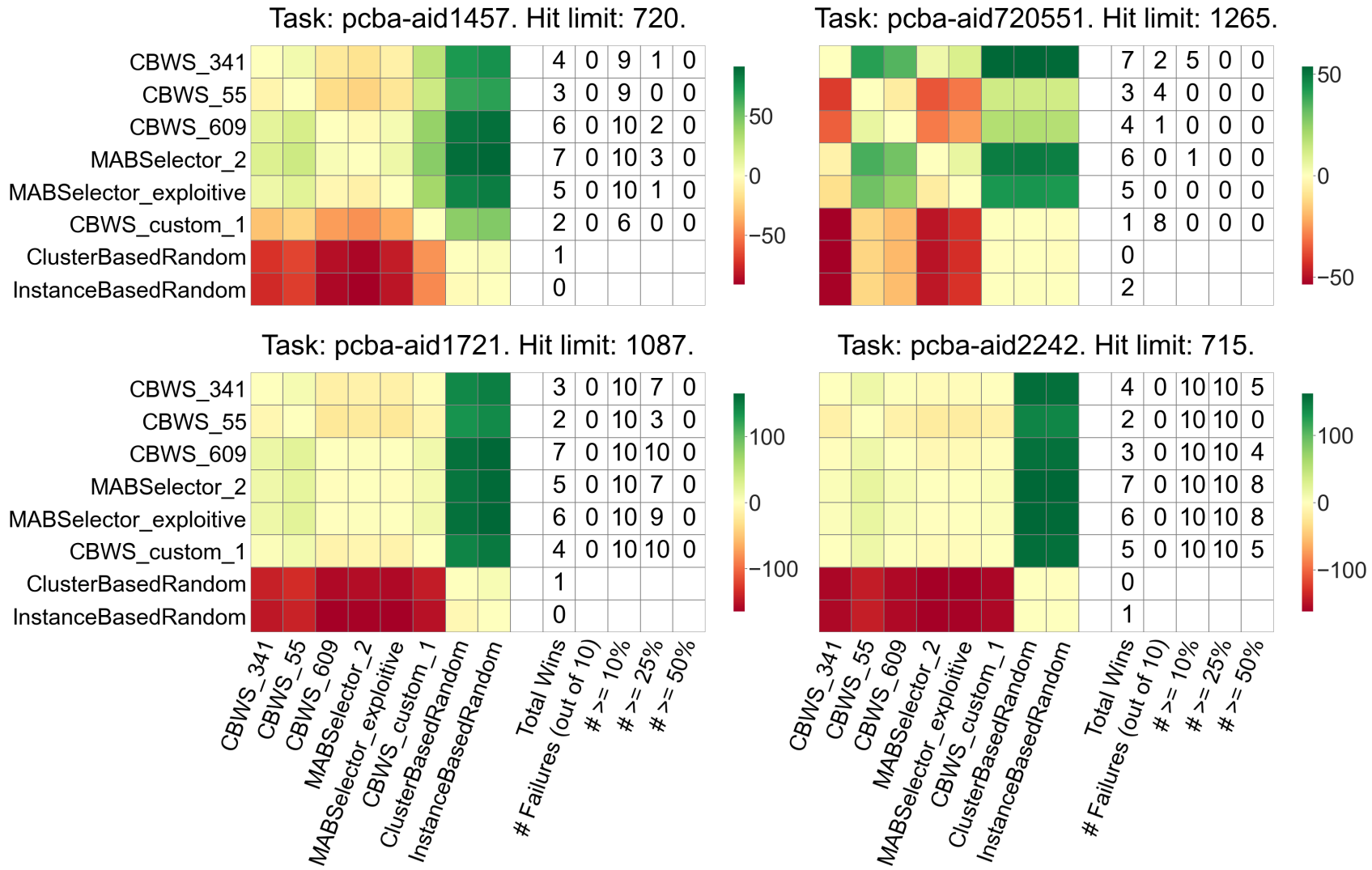


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(13 of 26 cont.)*

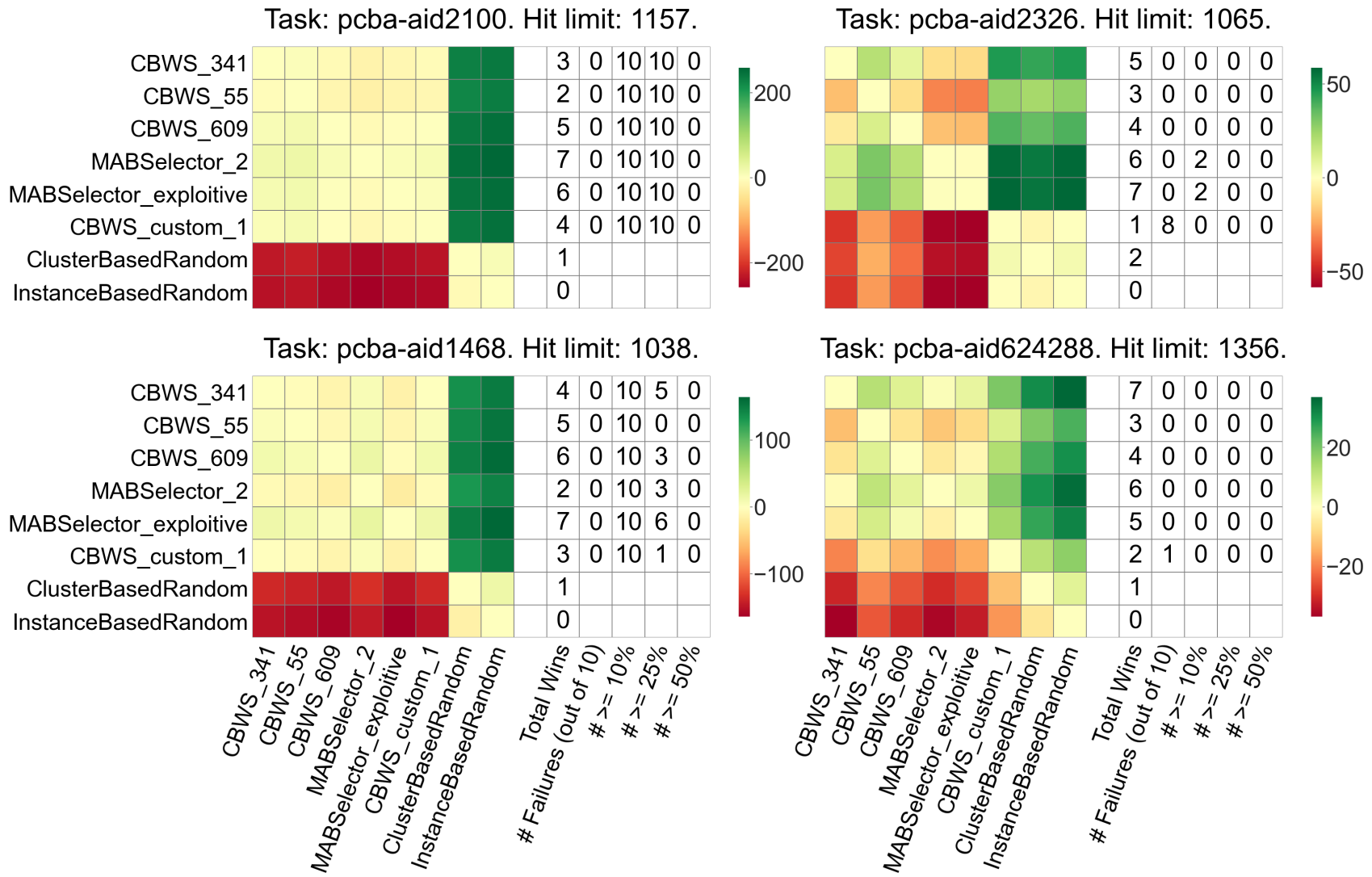


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(14 of 26 cont.)*

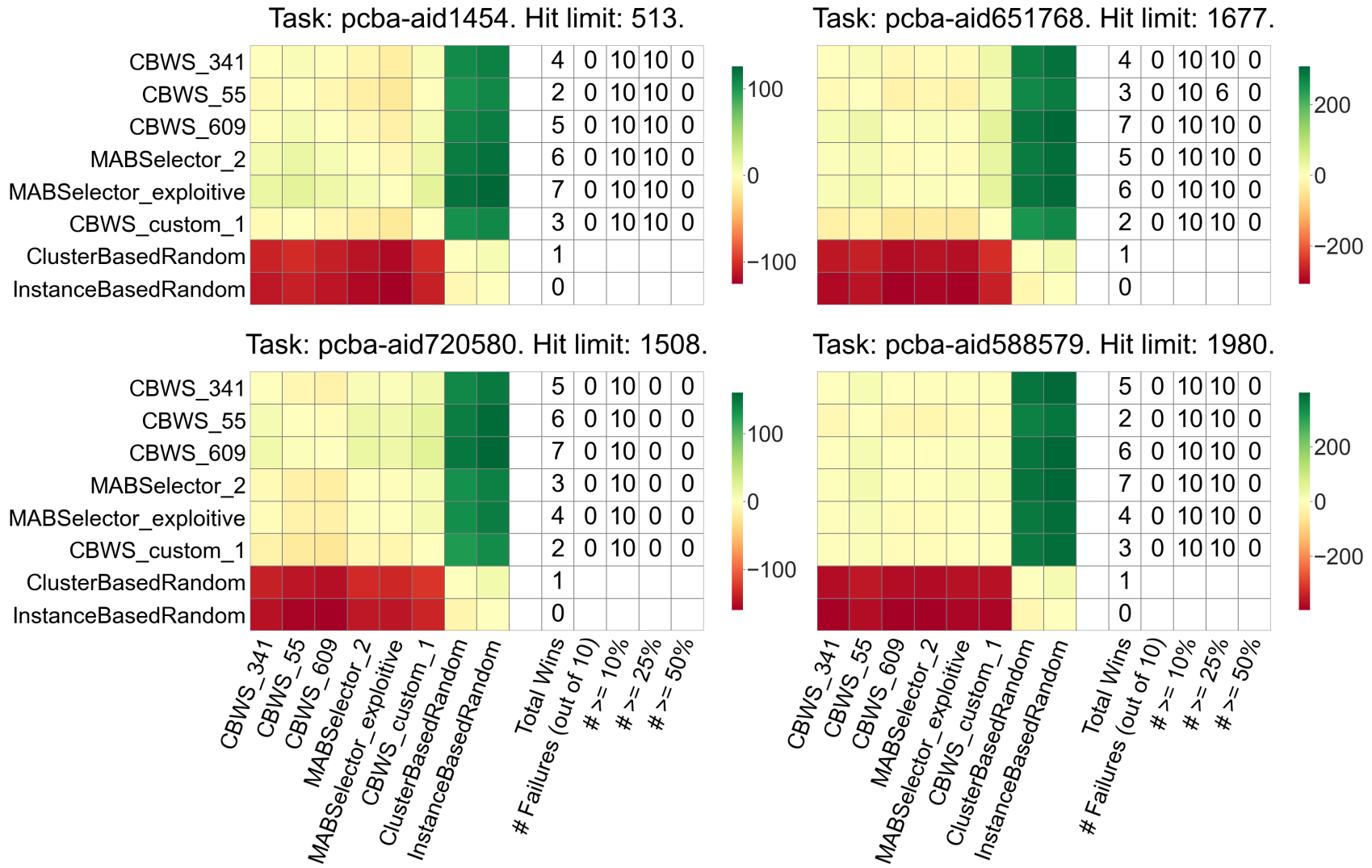


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(15 of 26 cont.)*

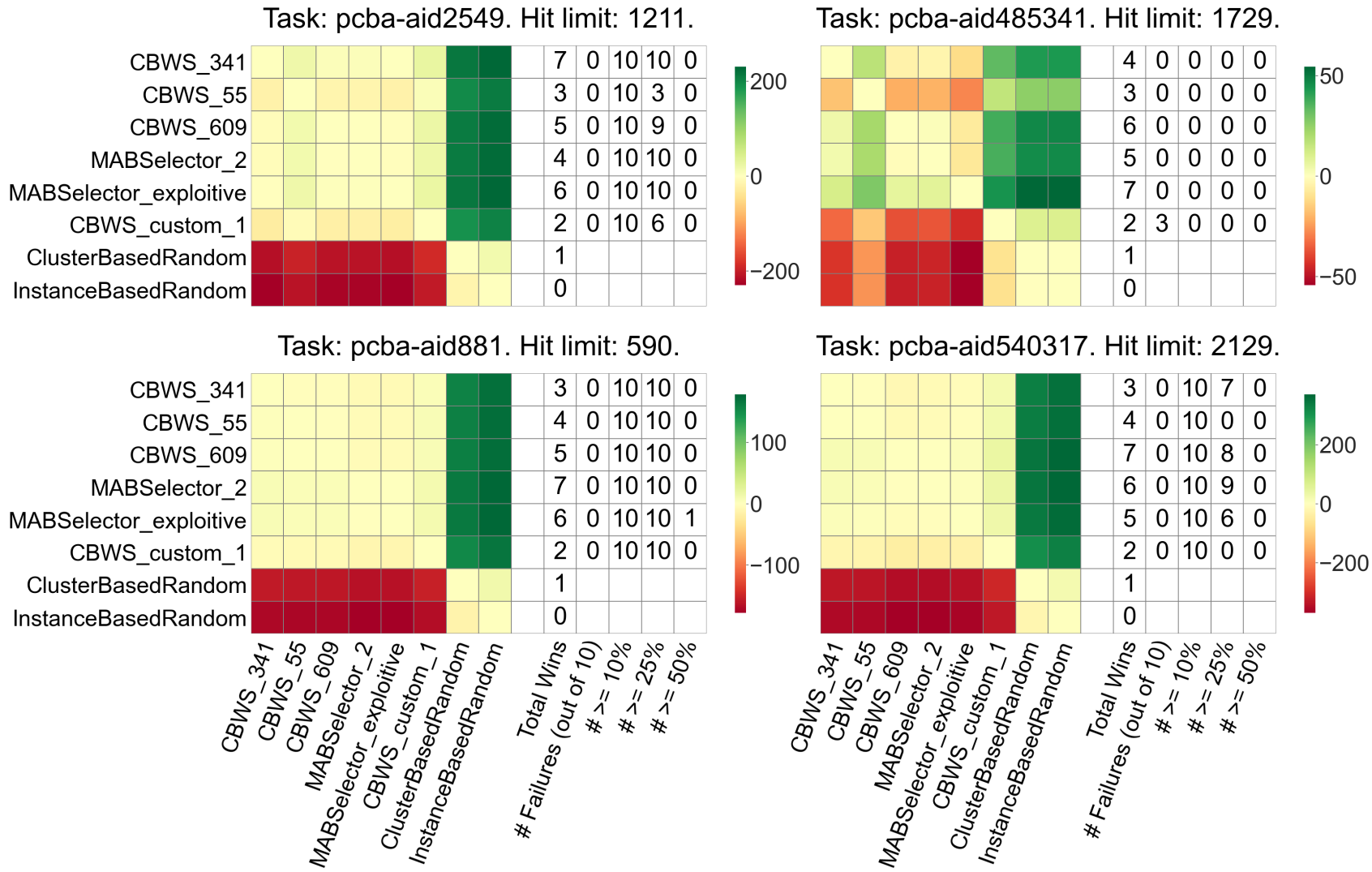


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(16 of 26 cont.)*

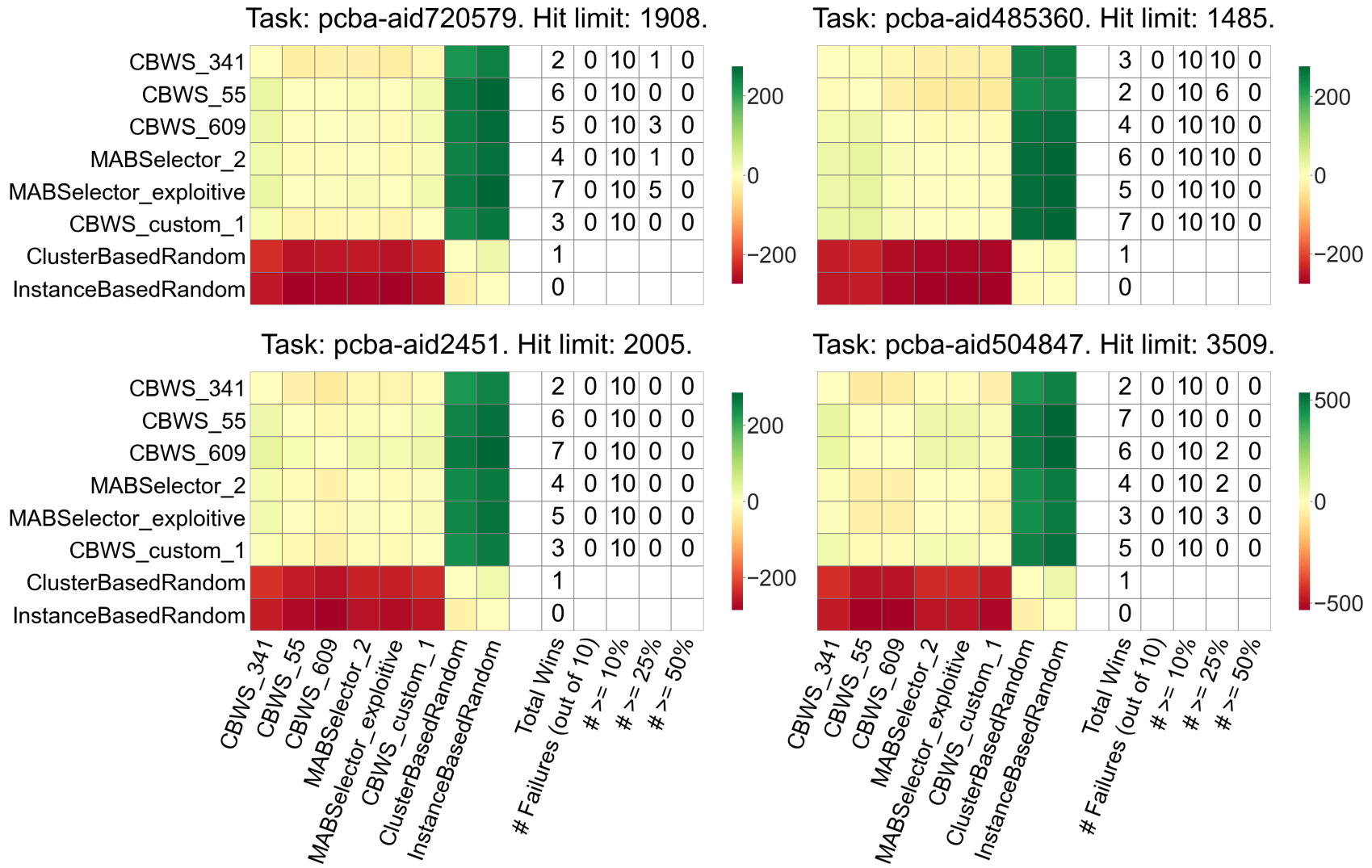


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(17 of 26 cont.)*

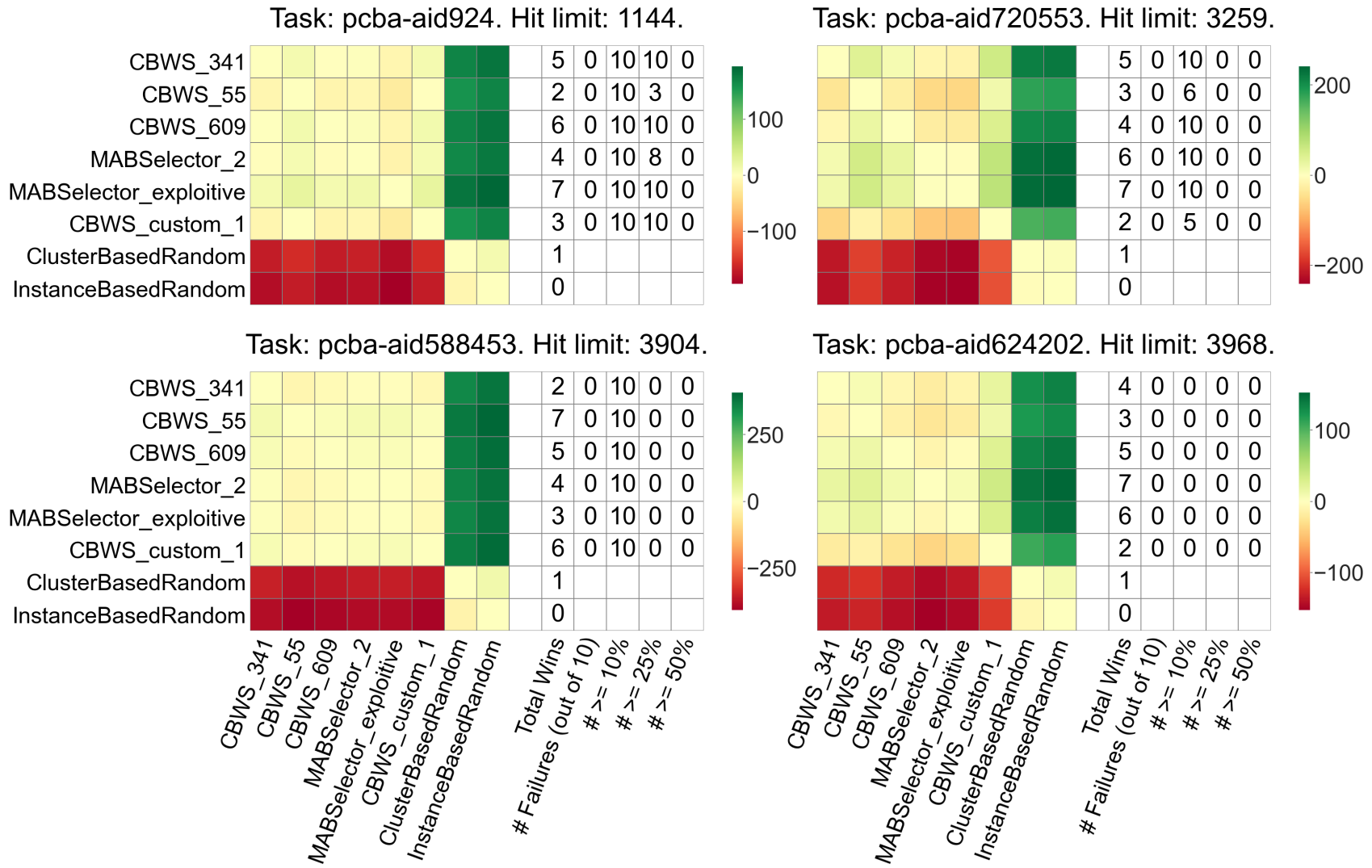


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(18 of 26 cont.)*

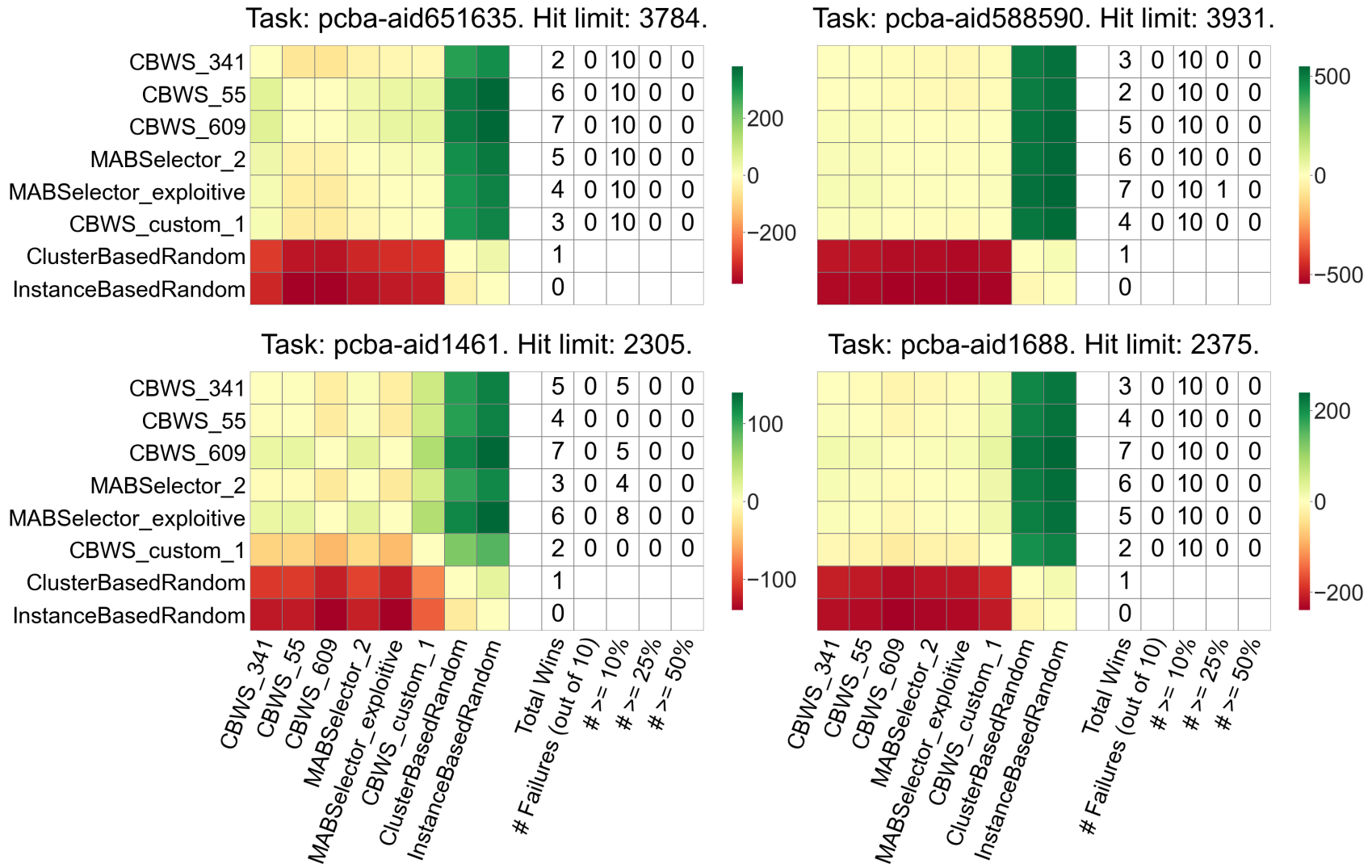


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(19 of 26 cont.)*

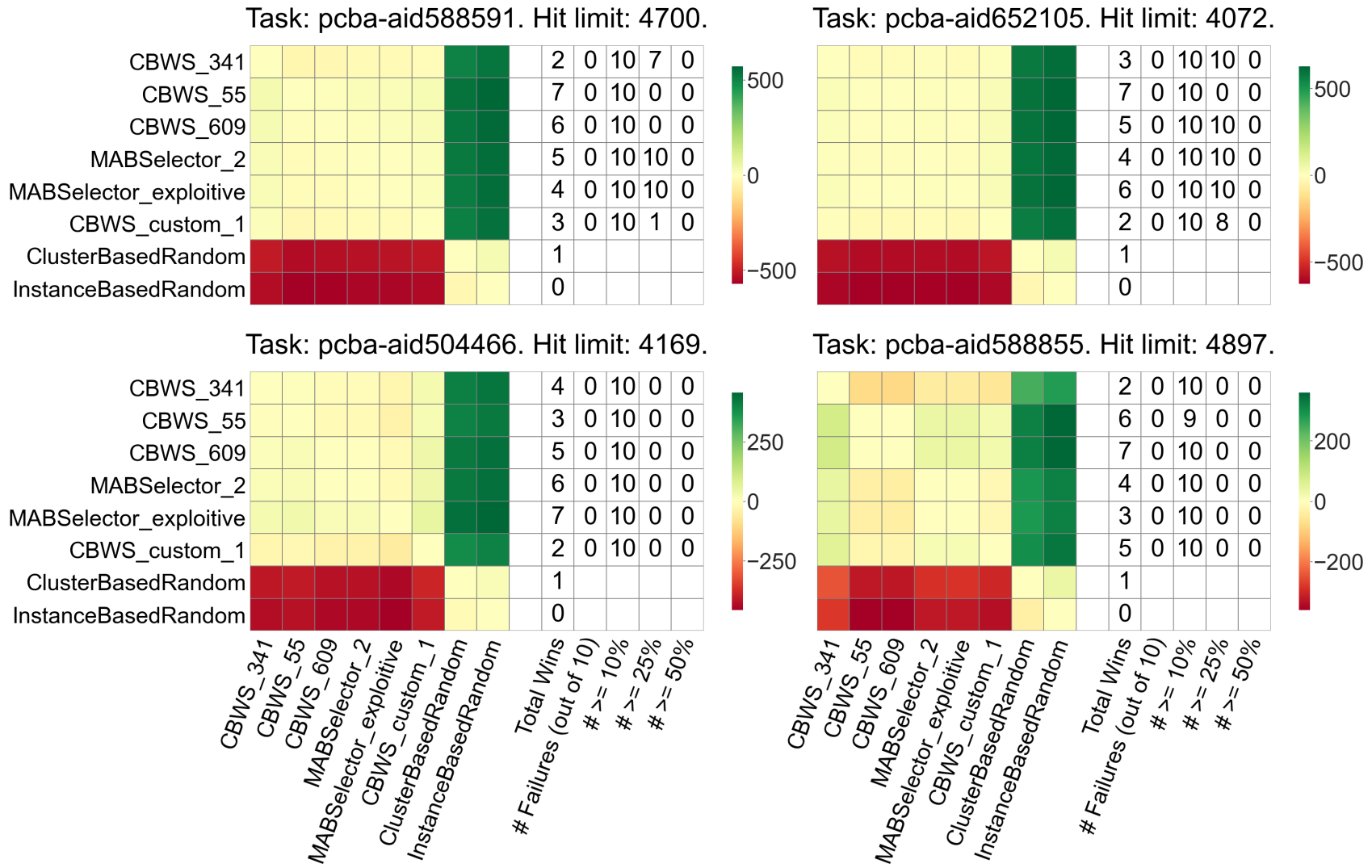


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(20 of 26 cont.)*

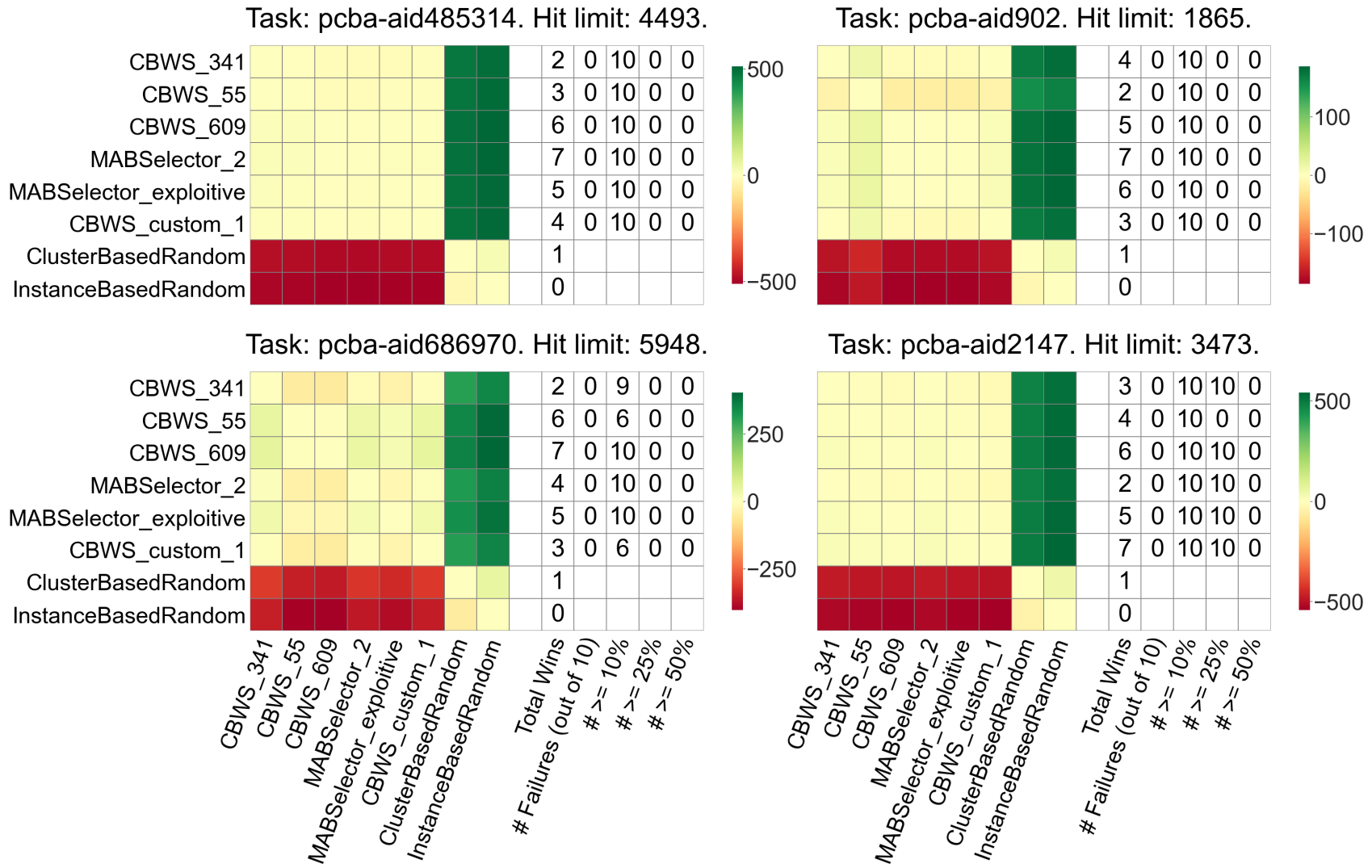


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(21 of 26 cont.)*

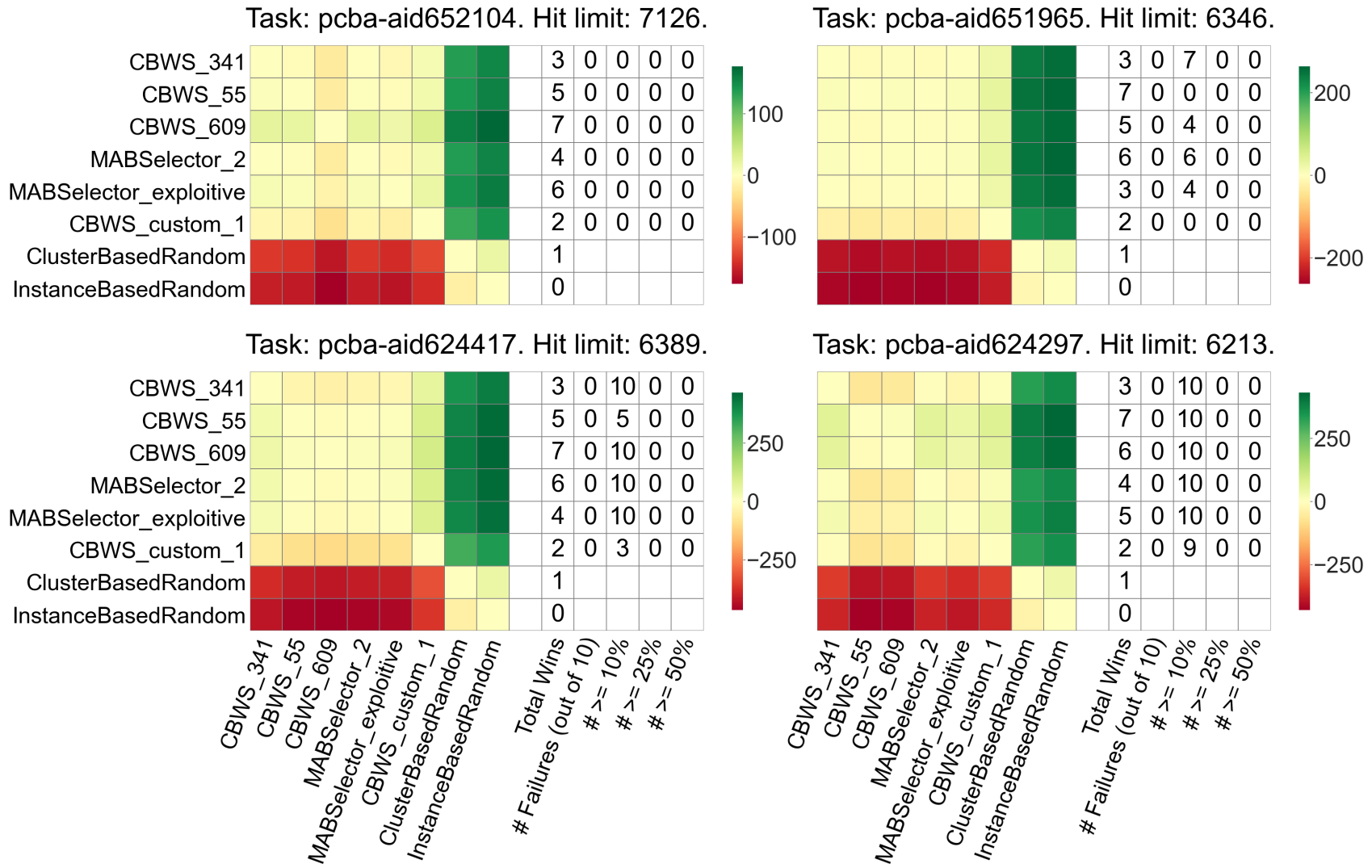


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(22 of 26 cont.)*

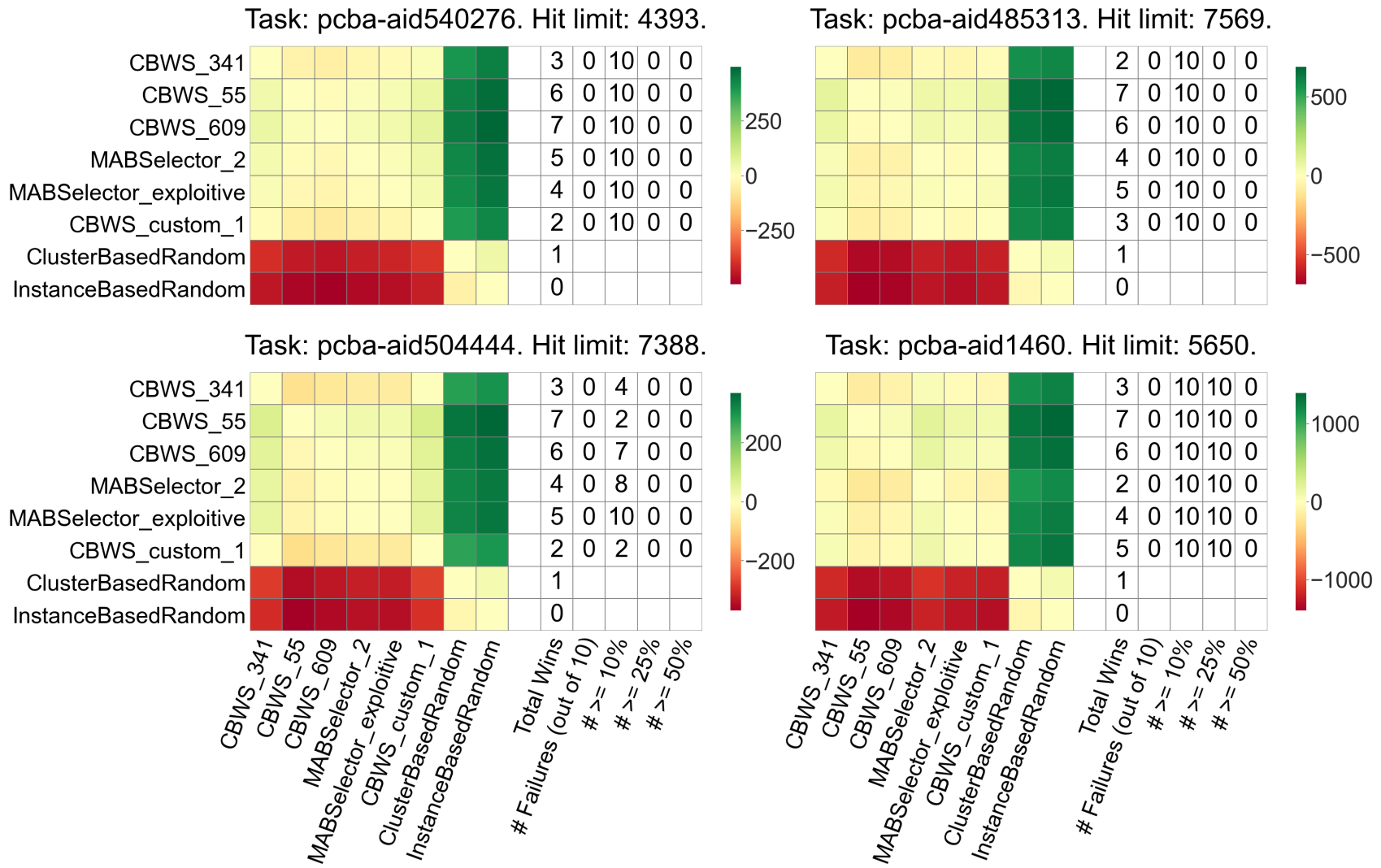


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(23 of 26 cont.)*

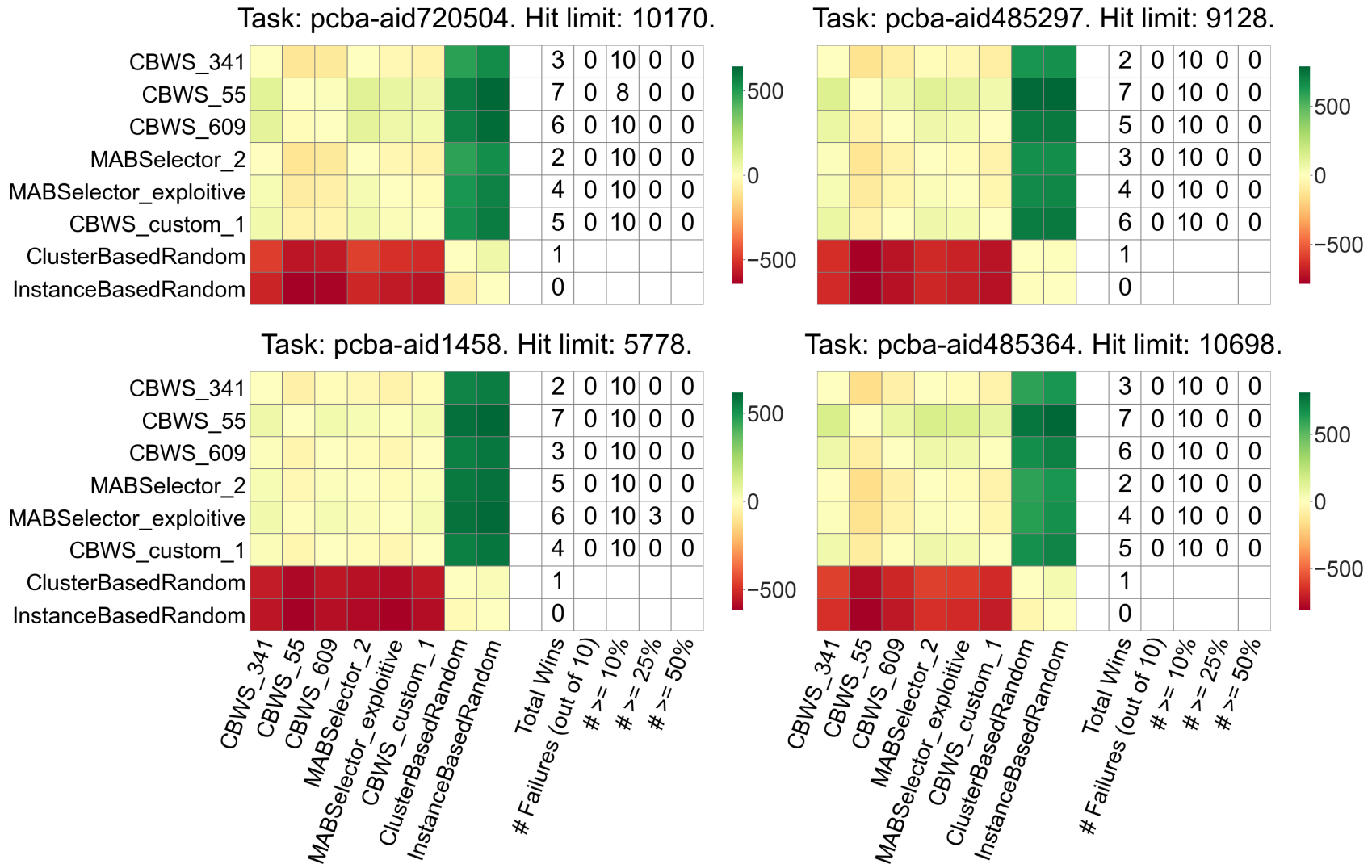


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(24 of 26 cont.)*

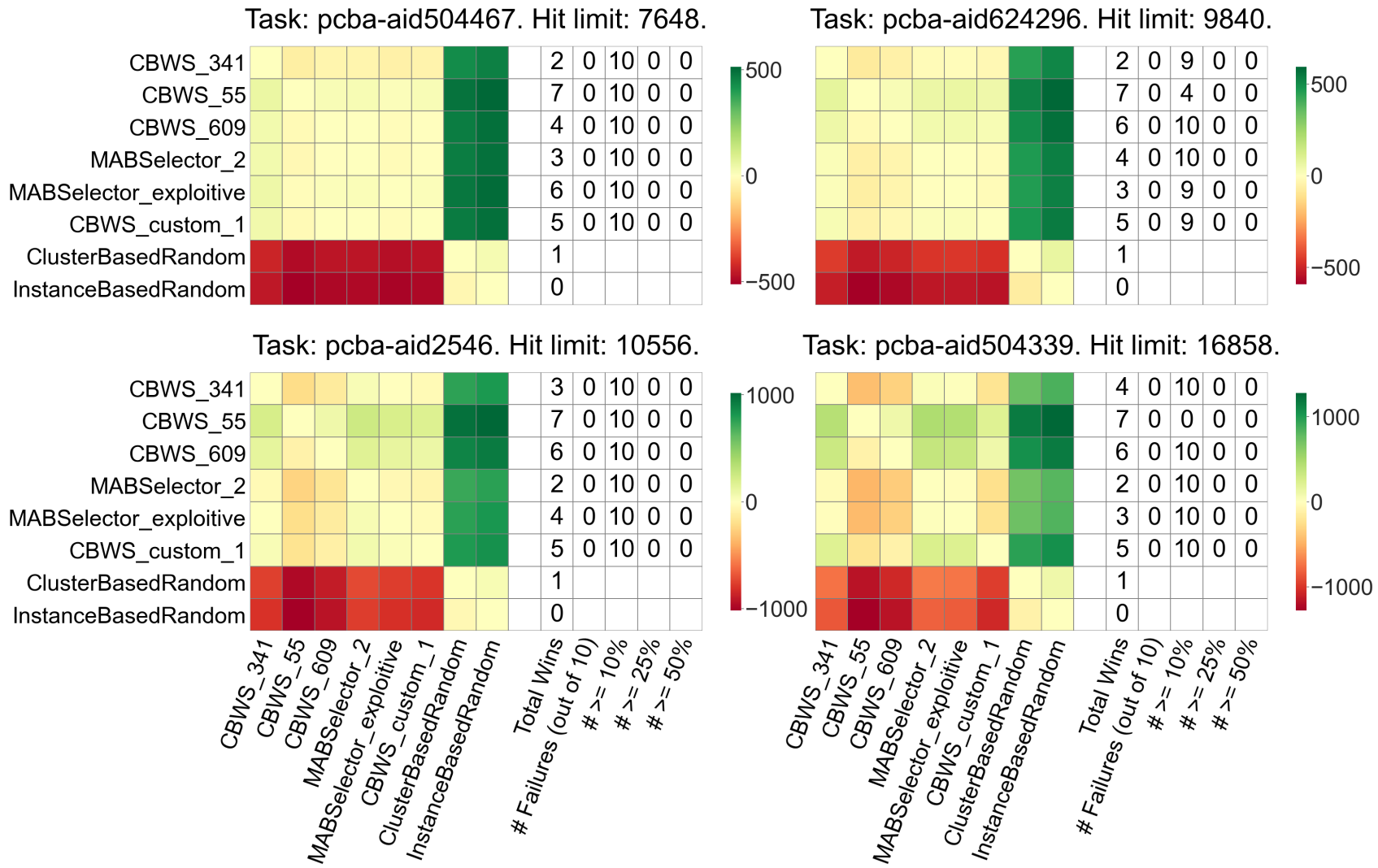


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(25 of 26 cont.)*

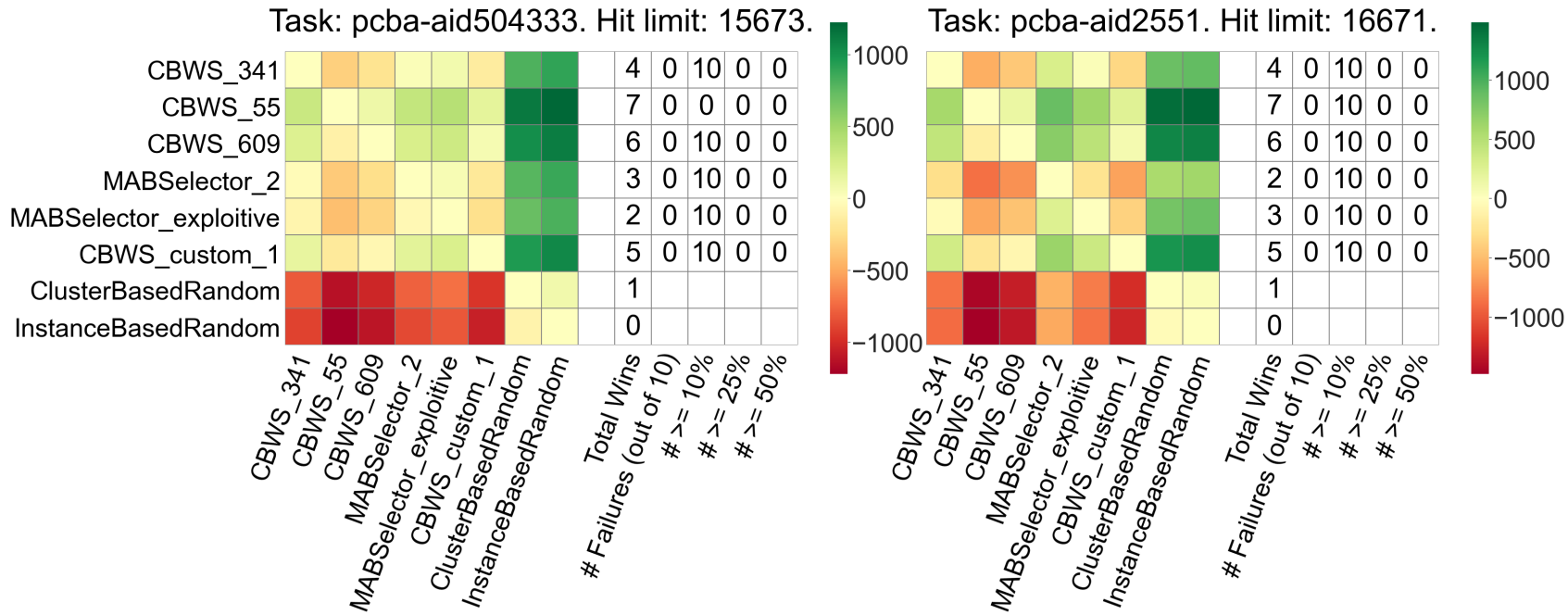


Figure B.2: Experiment 3.1 per task contrast estimation based on medians (CEM) heatmaps for **Total Unique Hits** after 50 iterations along with extra columns denoting counts for various conditions. *(26 of 26)*