Heterogeneous Heuristic Optimisation and Scheduling for First-Order Theorem Proving

DOI:
10.1007/978-3-030-81097-9_8

Document Version
Final published version

Link to publication record in Manchester Research Explorer

Citation for published version (APA):
Holden, E. K., & Korovin, K. (2021). Heterogeneous Heuristic Optimisation and Scheduling for First-Order Theorem Proving. In *International Conference on Intelligent Computer Mathematics 2021: Intelligent Computer Mathematics* (pp. 107-123). Article Chapter 8 (Intelligent Computer Mathematics; Vol. 12833). Springer Cham. https://doi.org/10.1007/978-3-030-81097-9_8

Published in:
International Conference on Intelligent Computer Mathematics 2021

Citing this paper
Please note that where the full-text provided on Manchester Research Explorer is the Author Accepted Manuscript or Proof version this may differ from the final Published version. If citing, it is advised that you check and use the publisher's definitive version.

General rights
Copyright and moral rights for the publications made accessible in the Research Explorer are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

Takedown policy
If you believe that this document breaches copyright please refer to the University of Manchester's Takedown Procedures [http://man.ac.uk/04Y6Bo] or contact uml.scholarlycommunications@manchester.ac.uk providing relevant details, so we can investigate your claim.

Download date:14. Jul. 2024

Heterogeneous Heuristic Optimisation and Scheduling for First-Order Theorem Proving

Edvard K. Holden[0000-0002-8782-3960] and Konstantin Korovin[0000-0002-0740-621X]

The University of Manchester
{edvard.holden|konstantin.korovin}@manchester.ac.uk

Abstract. Good heuristics are essential for successful proof search in first-order automated theorem proving. As a result, state-of-the-art theorem provers offer a range of options for tuning the proof search process to specific problems. However, the vast configuration space makes it exceedingly challenging to construct effective heuristics. In this paper we present a new approach called HOS-ML, for automatically discovering new heuristics and mapping problems into optimised local schedules comprising of these heuristics. Our approach is based on interleaving Bayesian hyper-parameter optimisation for discovering promising heuristics and dynamic clustering to make optimisation efficient on heterogeneous problems. HOS-ML also use constraint programming to devise locally optimal schedules and machine learning for mapping unseen problems into such schedules. We evaluated HOS-ML on the theorem prover iProver and demonstrated that it can discover new heuristics that considerably improve performance and can solve problems that have not been solved previously by any other system.

Keywords: Theorem Proving · Machine Learning · Heuristic Optimisation · Heuristic Selection · Dynamic Clustering

1 Introduction

Automated Theorem Provers (ATPs) are tools for automatically proving mathematical theorems, and have a wide range of applications from verification of software and hardware to automating interactive theorem proving in systems such as Sledgehammer [14]. ATPs have also contributed to large mathematical formalisation projects such as the MML (Mizar Mathematical Library) through the MPTP (Mizar Problems for Theorem Proving) [20].

State-of-the-art ATPs such as iProver [9], Vampire [10], E [18] and SPASS [22] have large sets of parameters that can be used to guide the proof search. It is well known that slight changes in values of these parameters can render problems from being not solved to being instantly solved and vise versa. Unfortunately, there is no general recipe for good parameters values or *heuristics*. While heuristics are essential for success, finding good heuristics is a major challenge due to the vast number of possible parameter combinations and values. Manually discovering good heuristics is time-consuming and in most cases not feasible even for system experts. For example, iProver has over 100 parameters with parameter types in the domain of reals, integers, Boolean, categorical

and lists. These parameters govern a wide range of simplifications, clause and literal selection strategies in a combination of instantiation, resolution and superposition calculi.

In this paper we develop a new approach, called HOS-ML, for automatically discovering new heuristics and mapping problems into optimised local schedules comprising of these heuristics. One of the key ingredients in our approach is Bayesian hyper-parameter optimisation. Hyper-parameter optimisation works well when applied to a homogeneous set of problems where it tries to find heuristics that optimise some performance metric over the whole set of problems. However, in practice problem sets are largely heterogeneous where vastly different heuristics are required to solve different problems. One way of dealing with this issue is to cluster similar problems and apply hyper-parameter optimisation individually to each cluster. A major challenge is that there is no obvious way of grouping problems into homogeneous clusters of similar problems based solely on syntactic properties. This is because even slight syntactic changes in a problem can result in a completely different problem which requires different heuristics to solve. In this paper, we solve this challenge by interleaving Bayesian hyper-parameter optimisation with dynamic clustering based on evaluation features. In this approach hyper-parameter optimisation and clustering incrementally refine each other: Bayesian hyper-parameter optimisation generates new heuristics that are used for clustering similar problems and in turn clustering similar problems helps Bayesian hyper-parameter optimisation to find diverse heuristics with good performance on each cluster. Other ingredients of HOS-ML include: i) training an embedding model for expanding clusters with similar unsolved problems using machine learning, ii) computing optimal local schedules for clusters using constraint programming, and iii) mapping unseen problems into local schedules using machine learning models.

We implemented HOS-ML and applied it to a theorem prover iProver. Experimental results show that HOS-ML can discover new heuristics that considerably increase the number of solved problems, including problems that have not been solved so far by any other system. Finally, we remark that the HOS-ML approach is rather general and can be applied to other domains for heuristics optimisation over heterogeneous problems.

Related work. Although parameter optimisation for first-order theorem proving received considerable attention [6, 7, 16, 21], it is primarily based on the assumption that problems are homogeneous and optimisation is performed uniformly over the whole problem set. In other domains, heuristic optimisation for heterougenous instances has been approached with one-off static feature clustering [8, 12, 17]. One of the major differences with our approach is that in HOS-ML hyper-parameter optimisation and clustering are dynamically interleaved which strengthen both optimisation and clustering during the run of the algorithm.

Heuristic selection is often approached by predicting the optimal heuristic for a given problem [1, 11, 23]. The main drawback of this approach is that it is unclear how to proceed when there are multiple good heuristics. A different approach was carried out in [15], where the internal prover state was used to predict the heuristic to run in the next time-slice. This is a promising approach but does not utilise predictive power of Bayesian hyper-parameter optimisation nor clustering. In our approach, we leverage the power of discovered heuristics by constructing schedules for each homogeneous cluster and build an embedding model for mapping unseen problems into schedules.

2 Hyper-Parameter Optimisation

Let $\mathcal{A}$ be a target algorithm, Θ a parameter space, $\mathcal{C}_\mathcal{A}$ a performance cost function, I a set of problem instances. If p is a problem in I, then $\mathcal{C}_\mathcal{A}(\theta, p) \in \mathbb{R}$ defines the performance cost associated with the run of the algorithm $\mathcal{A}$ with parameters (*heuristic*) θ on a problem p. Performance cost can be running time but can be a more sophisticated function that, e.g., increases cost if the problem was not solved within the given time limit and reduces cost if the problem was solved with this heuristic but was not solved by previously found heuristics. In Section 4.2 we discuss this in more detail.

Hyper-parameter optimisation for $\mathcal{A}$ over a problem set I is the problem of finding parameters $\theta_{\min}$ that minimize the cost function over all problem instances in I:

$$\theta^I_{\min} = \arg\min_{\theta \in \Theta} \Sigma_{p \in I} \mathcal{C}_\mathcal{A}(\theta, p).$$

Hyper-parameter optimisation [4] is well suited for homogeneous problem collections where we can assume that $\theta^I_{\min}$ is optimal or near optimal for all instances $p \in I$.

However, in our setting we are dealing with large collections of heterogeneous problems where there is no uniform best heuristic but rather different heuristics perform better on different classes of problems. Unfortunately, there is no simple criteria for grouping problems into homogeneous clusters that allow for single (or just few) near optimal heuristic(s) per cluster. We can observe that the search for optimal heuristics and clustering are interrelated problems. This observation is at the core of our HOS-ML approach where we interleave search for optimal heuristics with dynamic clustering based on the performance of these heuristics on individual problems.

3 Heterogeneous Heuristic Optimisation and Scheduling

Our approach for heterogeneous heuristic optimisation and scheduling (HOS-ML) is shown in Figure 1. Let us first overview HOS-ML at a high-level and in later sections we elaborate on each component. HOS-ML consists of three phases described below.

Phase 1: Heuristic optimisation for heterogeneous instances. This phase is applied to a set of training problems. The goal of this phase is to discover new heuristics that i) solve problems which could not be solved by heuristics discovered in previous iterations, and ii) improve the performance of problems that were previously solved. These conditions can be represented by a cost function for hyper-parameter optimisation.

One of the main challenges is that problems in our setting are heterogeneous, this prevents hyper-parameter optimisation to discover diverse heuristics which solve different types of problems. To address this challenge we cluster problems based on *dynamic evaluation features*. Dynamic evaluation features are problem features based on the evaluation of all heuristics discovered in previous iterations. These features reflect problem similarity based on the performance of different heuristics, and they are dynamic due to the growing number of discovered heuristics (see, Section 4).

The *heterogeneous heuristic optimisation loop* outlined by thick arrows in Figure 1, interleaves re-clustering based on dynamic evaluation features and hyper-parameter optimisation over each cluster.

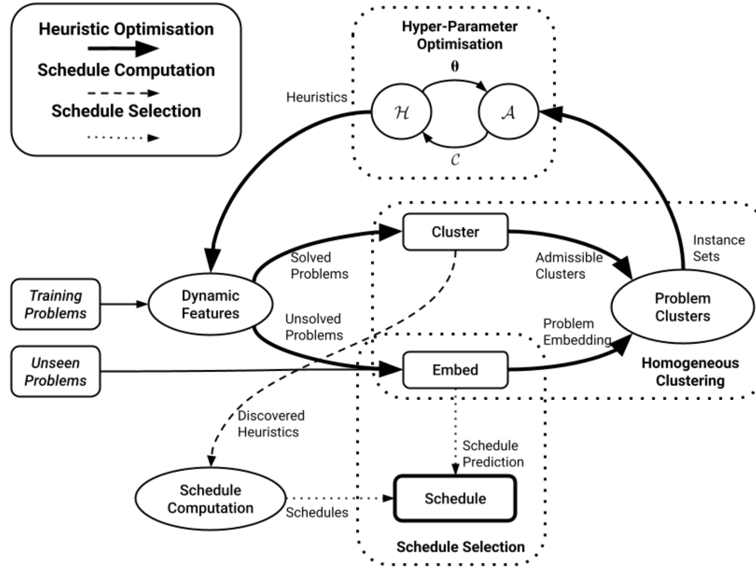

Fig. 1: HOS-ML: heuristic optimisation and selection for heterogeneous problems.

Clustering based on dynamic features is suitable for problems that can be solved by some of the available heuristics but is not applicable to unseen or unsolved problems. We address this problem by training a machine learning model for embedding static problem features into dynamic features. This *admissible embedding model* is re-trained after each loop iteration.

By combining problem clustering and embedding, we acquire homogeneous problem clusters. Next, we perform hyper-parameter optimisation over each problem cluster separately. This results in a set of well-performing heuristics for each cluster which we evaluate globally on all training problems. The heuristics evaluation is used to update the dynamic feature representation for each problem. Subsequently, we re-cluster problems based on the new evaluation features, increasing homogeneity of clusters and retrain the admissible embedding model. This loop is repeated until the global time limit or some other termination criterion is reached.

Phase 2: Local schedule computation. After Phase 1 is completed we compute schedules for the final clusters. We use constraint programming to create a schedule for each problem cluster based on discovered heuristics and their evaluation over the training set (Section 5).

Phase 3: Schedule selection. The final phase is the deployment of HOS-ML on unseen problems. This is done by first mapping the problem into a cluster using the admissible embedding model and then extracting the schedule associated with the corresponding

cluster computed in Phase 2. This phase is computationally cheap as the model and schedules have been computed in Phases 1 and 2, respectively.

In the following sections we will detail each part of HOS-ML.

4 Heuristic Optimisation for Heterogeneous Instances

In this section we describe in detail Phase 1, which is heuristic discovery and optimisation on heterogeneous problems. This phase is applied to a set of training problems.

The heuristic discovery and optimisation loop is detailed in Algorithm 1, it takes two inputs: *initial_heuristics* and *problems*. The goal of the algorithm is to discover new *global_heuristics* which improve the performance over *problems* in local clusters, with the *initial_heuristics* serving as starting points. This is achieved through interleaving homogeneous clustering and heuristic optimisation with the use of the inner and outer loops.

The algorithm first clusters solved problems based on the current dynamic heuristic evaluation features (detailed in Section 4.1). Then, the algorithm trains a machine learning model for embedding problems into clusters based on static problem features (detailed in Section 4.3). In this phase, this *admissible embedding model* is used to embed unsolved problems into clusters to achieve a balance between solved and unsolved problems in each cluster.

Next, the algorithm enters the inner loop to perform *local heuristic optimisation* over each problem cluster to discover good heuristics for each cluster which are added to *local_heuristics*. Then, *local_heuristics* are evaluated globally on the whole problem set to obtain new dynamic evaluation features for each problem. In the next iteration of the outer loop the problems are re-clustered based on these new dynamic evaluation features. With each iteration, problem clusters become increasingly more homogeneous with respect to accumulated heuristics performance. In the initial iterations, when there are only few heuristics we randomly sub-sample large clusters into smaller clusters.

In the following, we describe the key parts of Algorithm 1 which are: dynamic evaluation clustering, local heuristic optimisation, and the admissible embedding model.

4.1 Dynamic Evaluation Clustering

One way of clustering similar problems is using syntactic features such as the number of formulas, number of equalities, number of Horn or EPR formulas, etc. Such clustering is suitable when problems fall into well-behaved fragments, e.g., Horn or EPR. However, most problems are mixtures of formulas with different properties and do not fall into such classes. For such problems syntactic features poorly reflect similarity as e.g., adding a single non-Horn formula to a Horn problem can drastically change the behaviour of the problem and similar for other types of formulas.

In this work, we propose to use dynamic evaluation features which are based on solver performance under different heuristics. Such features directly link problem similarity with the solver performance, moreover these features are dynamically extended during the run of Algorithm 1 due to newly discovered heuristics by local heuristic optimisation.

Algorithm 1 Heterogeneous Heuristic Optimisation

Input: $initial_heuristics, problems$
Output: Learnt heuristics ($global_heuristics$)

1: $global_heuristics \leftarrow evaluate_heuristics(initial_heuristics, problems)$
2: **repeat**
3: $evaluation \leftarrow get_evaluation(global_heuristics)$
4: $solved, unsolved \leftarrow split(problems, evaluation)$
5: $problem_clusters \leftarrow compute_clusters(solved, evaluation)$
6: $cluster_model \leftarrow train_model(problem_clusters, solved)$
7: $problem_clusters \leftarrow problem_clusters \cup embed(cluster_model, unsolved)$
8: $local_heuristics \leftarrow \emptyset$
9: **for** $cluster \in problem_clusters$ **do**
10: $incumbent \leftarrow select_best_heuristic(cluster, global_heuristics)$
11: $local_heuristics \leftarrow local_heuristics \cup optimise(incumbent, cluster)$
12: **end for**
13: $global_heuristics \leftarrow global_heuristics \cup evaluate(local_heuristics, problems)$
14: **until** Timeout
15: **return** $global_heuristics$

Given a problem p, a heuristic θ, and a time limit β, the function $time_\beta$ gives the solving time t of θ executed on p with the time limit β, if a solution is found and ∞ otherwise. Given a set of heuristics $H \subset \Theta$, we can obtain the problem's heuristic evaluation vector $\mathbf{e_p}$ by computing $time_\beta$ for each problem-heuristic pair. The *evaluation vector* $\mathbf{e_p}$ represents the relationship between p and H. By computing the evaluation vector for the set of problems I, we obtain the heuristic evaluation matrix $\mathbf{E_{IxH}}$.

Admissible Features. The evaluation vector represents solving times of the successful solving attempts. However, similar problems can have different solving times, e.g., problems may differ by size but not structure. We want to cluster problems so local heuristic optimisation can transfer learning from simpler problems to more complex problems of the same type. This is achieved using clustering based on admissible features. First, we define when a heuristic is admissible for a problem. Assume we have problem p and a set of heuristics $H = \{\theta_1, \cdots, \theta_n\}$, where at least one of the heuristics solves p within the time limit β. Further, $t^*_H(p)$ is the fastest solution of p in H. Then, θ is *admissible in* H for p if $time_\beta(\theta, p)$ is approximate to $t^*_H(p)$, where the tolerance is defined by additive and multiplicative constants ϵ_k and ϵ_p, respectively.

$$Admissible_{(H,\beta)}(\theta, p) = \begin{cases} 1 & \text{if } time_\beta(\theta, p) \leqslant t^*_H(p) \cdot (1 + \epsilon_p) + \epsilon_k, \\ 0 & \text{otherwise.} \end{cases}$$

In particular, a heuristic is admissible for a problem, if it yields either the fastest or close to the fastest known solving time, and is not admissible if its performance is considerably worse or does not solve the problem at all. We can compute admissible heuristics based on the problem evaluation vector $\mathbf{e_p}$ and obtain its *admissible heuristic vector* $\mathbf{a_p}$.

Admissible Distance. Admissible heuristic vectors create a performance-based feature representation based in the known heuristic evaluations. Next, we need a distance function that can be used to group problems with similar behavioural properties. Let us note that Euclidean distance is poorly suited for this purpose, instead we considered Jaccard similarity distance and Sørensen-Dice distance, the latter lead to better clustering in our experiments. Let $|\cdot|$ be the L_1 norm over binary vectors, which is equal to the sum of all 1s in the vector. The Sørensen-Dice semi-metric distance can be defined as follows.

$$d(\mathbf{a_p}, \mathbf{a_{p'}}) = 1 - \frac{2 * (\mathbf{a_p} \cdot \mathbf{a_{p'}})}{|\mathbf{a_p}| + |\mathbf{a_{p'}}|}$$

Fig. 2: Sørensen-Dice distance between two admissible vectors $\mathbf{a_p}$ and $\mathbf{a_{p'}}$.

The Sørensen-Dice distance ranges between 0 and 1, 0 if the admissible vectors are equal and 1 if they have no admissible heuristics in common.

Admissible Clustering. We use the K-medoids algorithm (see, e.g., [13]) to cluster problems based on their admissible heuristic vectors and Sørensen-Dice similarity distance. K-medoids clustering partitions the problems into K clusters by minimising the sum of distances between each data point and the medoid of their cluster. The medoids act as a cluster centre and must be a data point, making it more robust towards outliers. K-medoid clustering tries to minimise the K-medoids cost function.

$$cost = \sum_{i=1}^{K} \sum_{p \in k_i} d(\mathbf{a_p}, \mathbf{m_i})$$

Fig. 3: The K-medoids cost function is the sum over all clusters of dissimilarities between the medoid of the cluster and members of the cluster.

The optimal number of clusters K can be computed using the elbow method. This is done by analysing the function representing dependency between cluster distortions and the number of clusters. The optimal number of clusters is reached at the elbow point, which is roughly defined as the point of maximum curvature of this function. This means that at this point, increasing the number of clusters does not result in any significant increase in cluster quality.

4.2 Local Heuristic Optimisation

After acquiring $problem_clusters$, we iterate local heuristic optimisation over each homogeneous $cluster$ to discover new heuristics. The heuristic optimisation searches for heuristics which minimise the following cost function.

Heuristic cost function. A new heuristic improves the performance of $global_heuristics$, if either it solves problems that were previously not solved or improves solving times of previously solved problems.

To accommodate these requirements we define the cost function as follows. Let H be the set of $global_heuristics$, β the time limit, $t^*_H(p)$ the fastest solution of p by heuristics in H. Consider a problem cluster I. Then we define the *heuristic cost function* for cluster I wrt H as:

$$\mathcal{C}^I_H(\theta,p) = \begin{cases} time_\beta(\theta,p) & \text{if } time_\beta(\theta,p) < \infty \text{ and } t^*_H(p) < \infty \\ -\beta|I| & \text{if } time_\beta(\theta,p) < \infty \text{ and } t^*_H(p) = \infty \\ \beta & \text{if } time_\beta(\theta,p) = \infty \end{cases}$$

Heuristic optimisation. For each cluster, we first compute the $incumbent$, which acts as a baseline and a starting point in the heuristic space. The $incumbent$ is computed as the heuristic in *global_heuristics* with the smallest cost in the $cluster$. Next, we use Bayesian hyper-parameter optimisation to discover efficient heuristics for each $cluster$, as shown in Figure 4. The Bayesian optimiser builds a statistical model for predicting the cost of running a heuristic on the problem cluster and uses this model to find promising heuristics. The most promising heuristics are evaluated on the problem cluster. Associated costs are used to update the model belief, which improves the prediction of heuristics costs. Continuously updating the model and evaluating the most promising heuristics is crucial as each evaluation demands considerable computation resources. In this work, we use the hyper-parameter optimiser SMAC (Sequential Model-based Algorithm Configuration) [5].

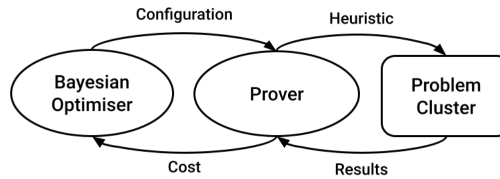

Fig. 4: The Bayesian optimiser creates heuristics which are evaluated over the cluster. The performance is scored and returned to the optimiser as the cost.

Global evaluation. After performing hyper-parameter optimisation, we select a subset of locally evaluated heuristics to evaluate globally. For this we greedily compute a set cover of the solved problems by the most effective heuristics and add them to the set of $local_heuristics$. Next, the *local_heuristics* are evaluated on the global problem set, and added to the *global_heuristics*. Evaluations of new heuristics are used to extend dynamic evaluation features and re-cluster problems as described in Section 4.1. Discovering new heuristics by local heuristics optimisation and re-clustering continues for each iteration of Algorithm 1 until it reaches the termination condition.

4.3 Embedding Unsolved Problems

Admissible heuristic vectors are suitable for clustering problems that are solved by at least one heuristic. However, if a problem has no solutions by $global_heuristics$ within the time limit β, admissible heuristics can not be determined. Nevertheless, unsolved problems should be clustered with similar solved problems, which would help heuristic optimisation to discover heuristics that can solve these problems. We observe that, in most cases, for some sufficiently large time limit β^* each unsolved problem will have at least one admissible heuristic in $global_heuristics$. This is the case in our application due to completeness of the underling calculi for first-order logic. Unfortunately, such time limits could be arbitrary large and infeasible to compute in practice. Instead, we propose to build a machine learning model to predict admissible heuristic vectors using static problem features. This *admissible embedding model* is trained on solved problems and is applied to unsolved problems to predict admissible heuristic vectors. Using this model we can assign each unsolved problems to a nearest cluster of solved problems. As a result, we acquire homogeneous problem clusters consisting of both solved and unsolved problems, as shown in Figure 5.

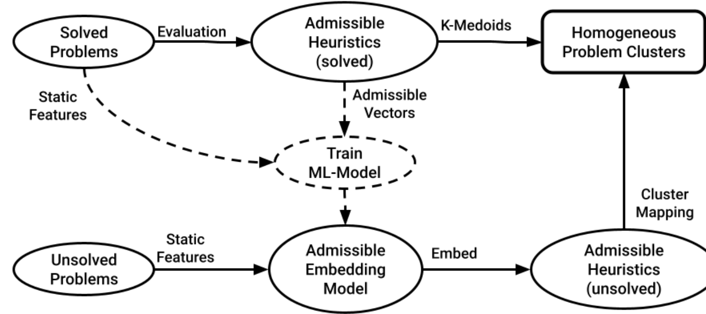

Fig. 5: Computing homogeneous clusters consisting of both solved and unsolved problems.

Static problem features. We consider two types of static features: syntactic features and solver state features. Syntactic features include properties such as the number of equational, Horn, EPR formulas, etc. As we noted before, such features do not always reflect algorithmic properties of formulas. To mitigate this we consider *solver state features*. During a run, the solver executes numerous function calls and applies various simplification techniques. Solver state features include solver statistics on key function calls, successful simplifications and corresponding timing statistics. We compute the solver state features by attempting a problem with single heuristic for a low-timelimit and extracting the solver statistics after termination.

Admissible embedding model. For a problem p, we denote its static feature vector as $\mathbf{s_p}$. Let H be the set of $global_heuristics$. The *admissible embedding model* $\mathcal{E}$ learns

the mapping between problems static features and their admissible heuristics vectors. As admissible heuristics vectors are binary vectors, we use a multi-label machine learning model as the embedding function. We separate the multi-label classification task into $|H|$ binary classification tasks, where a separate binary classification model $\mathcal{M}_\theta$ is trained for each heuristic θ in H. We considered different machine learning methods for building binary classification models: random forests, tree models and neural networks. In our experiments the decision tree algorithm XGBoost [2] yielded the best performance. Once binary models are trained for each heuristic, the admissible embedding model is then defined as $\mathcal{E}(\mathbf{s_p}) = (\mathcal{M}_{\theta_1}(\mathbf{s_p}), \cdots, \mathcal{M}_{\theta_{|H|}}(\mathbf{s_p}))$. The admissible embedding model can then be used to predict the admissible heuristics vector of any given problem.

Mapping unsolved problems to clusters. The embedding model is trained on the solved problems and used to predict the unsolved problems' admissible heuristics vector. Next, we need to map problems into discovered homogeneous clusters. This is achieved by first computing the admissible distance between each predicted admissible vector and each cluster medoid. Second we add the n closest unsolved problems to each cluster. This results in homogeneous clusters consisting of both solved and unsolved problems.

5 Local Schedules for Heterogeneous Instances

In this section we describe Phase 2 of HOS-ML, which is the computation of heuristic schedules for each homogeneous problem clusters.

The scheduled running time t of heuristic θ is described by the pair (θ, t). A *heuristic schedule* is an ordered set of heuristic-time pairs $[(\theta_1, t_1), \cdots, (\theta_n, t_n)]$ where the total running time does not exceed the global time limit $\sum_{i=1}^{n} t_i \leq \beta$. We describe the task of creating a schedule as the *heuristic scheduling problem*. Given the problem set I, the heuristic set $H \subset \Theta$ and the heuristic evaluations $\mathbf{E_{I \times H}}$, find the heuristic run-times $[t_1, \ldots, t_n]$, which maximise the performance on I subject to the global time-limit β.

We solve the heuristic scheduling problem using constraint programming with the following encoding. Let $|H| = n$ and $|I| = m$. First we create the run-time variables $t_1, \ldots, t_n$ which represent the running time of each heuristic in H. Next, we ensure that the total running time of the schedule does not exceed β with the constraint $\sum_{i=1}^{n} t_i \leq \beta$. Using known evaluations $\mathbf{E_{I \times H}}$ we represent that heuristic h_i solves problem p_j in allocated time t_i as $E_{ji} \leq t_i$. A problem p_j is solved by the schedule if $E_{j1} \leq t_1 \vee \ldots \vee E_{jn} \leq t_n$ holds. We denote this condition as s_j.

The objective is to maximise the number of problems solved by the schedule. Hence, the task of the constraint solver is to find the heuristic runtimes $t_1, \ldots, t_n$ which maximise $\sum_{i=1}^{m} s_i$.

After acquiring the solution, we discard all heuristics θ_i for which $t_i = 0$ and order the remaining heuristics in ascending order according to their run-times. The result is a heuristic schedule that maximises the number of solved problems over the problem set based on the known heuristic performance.

6 Schedule Selection

Phase 2 of HOS-ML computes schedules for each homogeneous problem cluster. This results in a set of optimal local schedules.

In Phase 3, we create a mapping between unseen problems and local schedules based on the embedding function, as shown in Figure 6. First, we extract the static features of the problems and predict their admissible heuristic vectors. Next, we use the predicted admissible heuristics to map the problems to their appropriate clusters. Finally, we attempt the problems with the schedule of their assigned cluster.

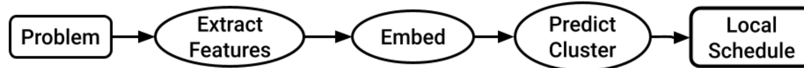

Fig. 6: Mapping a problem to a local schedule.

This concludes the description of all phases of HOS-ML. In the next section we discuss implementation and evaluation results.

7 Experimental Evaluation

HOS-ML is implemented in Python 3. Pandas and scikit-learn were used for data handling and processing. The heuristic discovery phase[1] uses SMAC [5] as the hyper-parameter optimiser, while CP-SAT[2] is used as the constraint solver for computing the schedules[3]. To perform clustering, we use the sklearn_extra implementation of K-medoids. Kneed[4] is used to compute the optimal value of K. The binary base predictor in the embedding function is implemented using XGBoost [2].

The base prover for HOS-ML is iProver [3, 9] a high performance theorem prover for first-order logic which is based on a combination of instantiation, resolution and superposition calculi. iProver heuristics are made up of 120 different parameters with diverse parameter values consisting of boolean, real, ordinal, priority lists and categorical values. The extensive range of values and parameters yield a vast and complex heuristic space. Our system supports optimisation over all of iProver parameters. However, in this experimental evaluation, we restrict optimisation to the parameters related to the newly developed superposition functionality [3]. We also use iProver to compute the prover state features, which comprise of 170 individual statistics covering both the problem properties and the prover behaviour. The experiments were run on a cluster of 33 machines, each with 4 Intel(R) Xeon(R) CPU L5410 @ 2.33GHz.

[1] Heuristic discovery is available at: https://gitlab.com/korovin/iprover-smac
[2] CP-SAT is available at: https://github.com/google/or-tools
[3] Schedule computation is available at: https://gitlab.com/edvardholden/scpeduler
[4] kneed is available at: https://github.com/arvkevi/kneed

7.1 Discovering New Heuristics

Our first experiment evaluated the heterogeneous heuristic optimisation phase on the TPTP library (v7.4.0) [19]. The library contains problems with varying difficulty from different domains ranging from verifying authentication protocols to MPTP problems from the Mizar mathematical library. The training set was created by randomly sampling 4000 of the 17053 FOF and CNF problems.

We ran five iterations of the outer loop of HOS-ML, with iProver's default heuristic as the initial heuristic. For each iteration, we optimised iProver's superposition options on three sampled problem clusters. The hyper-parameter optimiser evaluated 1000 candidate heuristics, each with a time limit of 20 seconds. The algorithm ran for approximately 61 hours and discovered a total of 53 new heuristics. The discovered heuristics where evaluated on a testing set consisting of the remaining TPTP problems (13053) with a time-limit of 20 seconds.

Next, we compared the performance difference between the default iProver heuristic and the set of heuristics discovered by heterogeneous heuristic optimisation. The results are shown in Table 1. We can observe that the new heuristics considerably increase the number of solved problems in both problem sets. The discovered heuristics also decrease the average solving time of the problems solved in the intersection of both approaches.

	Training		Testing		Total	
	Solved	Avg Time	Solved	Avg Time	Solved	Avg Time
Default	1975	1.81	6774	1.53	8749	1.60
Discovered	2272	0.98	7771	0.66	10043	0.73

Table 1: The performance of the default and the global heuristics (20s).

7.2 Revealing Homogeneity with Admissible Evaluation Clustering

One of the key ideas of HOS-ML is to discover homogeneous problem clusters by clustering on admissible heuristic features. To verify this claim, we compute admissible clusters and apply the best local heuristic of each cluster to its members. Further, we compute the intersection between problems solved by the best global heuristic and the set of problems solved by the best local heuristics. Next, we compute the average solving time of the problems in the intersection. If the performance of the global and local approaches is the same, the clusters are equivalent to random sampling. However, if the local heuristics perform better, the clusters are more homogeneous.

The global heuristics sampled at each iteration of the outer loop forms the five heuristic sets A, B, C, D and E applied to the 4000 training problems. Next, we compute the performance of both approaches as shown in Table 2. We can observe that the local heuristics offer a considerable performance increase. Hence, we acquire homogeneous problem clusters through admissible clustering.

Heuristic set	A	B	C	D	E
Number of heuristics	18	32	41	51	54
Number of clusters	12	18	98	119	124
Solved global	1975	1975	1975	1975	1975
Solved local	2106	2159	2254	2269	2271
Solved intersection	1958	1960	1975	1975	1975
Avg Time Global	1.76	1.76	1.81	1.81	1.81
Avg Time Local	1.45	1.35	1.29	1.22	1.22
Performance Increase	17.32%	22.89%	28.60%	32.47%	32.51%

Table 2: Performance of the best global heuristic versus the best local heuristics.

7.3 Embedding Evaluation Features

HOS-ML embeds problems into admissible heuristic features during heuristic optimisation and selection. To evaluate the embedding performance we create a model for embedding the selected 4000 TPTP problems into the evaluation data from experiment 7.1, as follows. First, we compute static problem features by collecting prover statistics of the problems with a 1-second time-limit. The prover statistics are transformed into features through log-scaling and standardisation. Next, we remove all problems that were either solved during processing or failed to parse within the time-limit. Further, we remove unsolved problems and problems with solutions below five seconds. This results in a challenging yet solvable problem set which is further divided into training and validation sets with a 70–30% split. The multi-label model comprises of binary classification models for each heuristic, trained using XGBoost [2].

In Table 3 we see that a single binary model is able to capture whether a heuristic is admissible for a problem. In Table 4 we can observe that the embedding model is able to predict admissible heuristic vectors of problems with good accuracy.

- **Admissible Similarity:** The Sørensen-Dice similarity between two admissible vectors $\mathbf{a}$ and $\mathbf{a}'$, which is equal to $1 - d(\mathbf{a}, \mathbf{a}')$.
- **Geometric Accuracy:** The sensitivity is the true positive rate, and the specificity is the true negative rate of the model predictions. The geometric accuracy is defined as $\sqrt{sensitivity * specificity}$, and by computing the average of each binary model geometric accuracy, we obtain the average geometric accuracy.

7.4 Optimal Scheduling of Heuristics

After discovering a set of strong heuristics, we devise a strategy for applying the heuristics to new problems. In this section, we evaluate three different heuristic strategies, each with a time limit of 20 seconds per problem:

- **Best Heuristic:** The heuristic which solves the most training problems.
- **Global Schedule:** The global heuristic schedule computed over the problem set.
- **Admissible Schedule:** A set of local schedules computed for each problem cluster.

Actual \ Predicted	1	0
1	157	21
0	41	112

Metric	Score
Accuracy	0.81
F1-Score	0.78
Geometric	0.80

Table 3: One of the binary XGBoost models.

Heuristic Set	Geometric	Similarity
A	0.72	0.71
B	0.71	0.68
C	0.72	0.68
D	0.72	0.68
E	0.72	0.67

Table 4: The multi-label model.

We constructed schedules on the training problems and evaluated their performance on the testing problems. The results of each approach are shown in Table 5, and Figure 7. The best heuristic and the global schedule perform similarly on the training set in terms of the number of solved problems. However, on the testing set it becomes apparent that attempting a problem with multiple heuristics is advantageous. Still, to utilise the full potential of a heuristic set, it is essential to create schedules for problems with similar performance. This is illustrated by the admissible schedule solving nearly one thousand problems more than the global schedule on the test set.

Approach	Training	Testing	Total
Best Heuristic	1975	6774	8749
Global Schedule	1976	6794	8770
Admissible Schedule	2258	7637	9895

Table 5: The number of solved problems of each scheduling approach (20s).

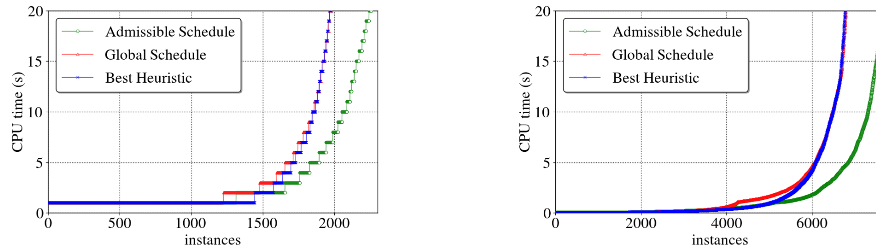

Fig. 7: The performance on the training set (left) and testing set (right).

7.5 Overall Performance Contribution

TPTP problems are rated based on their difficulty on a scale from zero to one. A problem solved by almost all state-of-the-art ATP systems has a rating of zero, while a problem

with no recorded solutions has a rating of one. There are 3984 TPTP problems with a rating of 0.9 or higher, and we characterise these problems as "highly challenging".

When run with a 20 seconds time limit, the 54 global heuristics discovered by HOS-ML solved 47 highly challenging problems. Nevertheless, challenging problems are likely to require more time. While the conventional time limit in ATP is 300 seconds, this would carry a substantial computational cost for 54 heuristics. Instead, we reduce the heuristic set by computing the set cover of solved problems and select the ten most contributing heuristics. Next, we evaluated the selected ten heuristics over all TPTP problems with a time limit of 300 seconds.

The ten heuristics solved a total of 10696 problems, of which 130 problems have a rating of 0.9 or above. These include 54 MPTP problems from the Mizar mathematical library. When combining these results with the 20-second evaluations, the number of highly challenging solved problems increases to 136 problems. Thirteen of these problems have the rating one, including four MPTP problems. As a result, the new heuristics solve MML problems with no previously recorded ATP solutions.

8 Conclusion

In this paper, we presented HOS-ML, a new method for heuristic optimisation and scheduling over heterogeneous problem sets. HOS-ML interleaves dynamic clustering with hyper-parameter optimisation and uses machine learning for embedding problems into clusters and local schedules. We applied HOS-ML to iProver and demonstrated that HOS-ML can discover new heuristics that can considerably improve prover performance over heterogeneous instances. Our evaluation showed that HOS-ML discovered heuristics that increase the number of solved TPTP problems by 14.8%, including problems with the rating 1, that have not been previously solved by any other system. These heuristics also decrease the solving time of previously solved problems by 54.4%. As a future work we will investigate applications of HOS-ML to different domains.

References

1. Bridge, J.P., Holden, S.B., Paulson, L.C.: Machine learning for first-order theorem proving - learning to select a good heuristic. J. Autom. Reason. **53**(2), 141–172 (2014)
2. Chen, T., Guestrin, C.: XGBoost. Proc. of the 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (2016)
3. Duarte, A., Korovin, K.: Implementing superposition in iProver (system description). In: Peltier, N., Sofronie-Stokkermans, V. (eds.) Automated Reasoning - 10th Int. Joint Conf., IJCAR , Proc., Part II. LNCS, vol. 12167, pp. 388–397. Springer (2020)
4. Feurer, M., Hutter, F.: Hyperparameter optimization. In: Hutter, F., Kotthoff, L., Vanschoren, J. (eds.) Automated Machine Learning - Methods, Systems, Challenges, pp. 3–33. The Springer Series on Challenges in Machine Learning, Springer (2019)
5. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Parallel algorithm configuration. In: Hamadi, Y., Schoenauer, M. (eds.) Learning and Intelligent Optimization - 6th Int. Conf., LION 6. LNCS, vol. 7219, pp. 55–70. Springer (2012)
6. Jakubuv, J., Suda, M., Urban, J.: Automated invention of strategies and term orderings for vampire. In: Benzmüller, C., Lisetti, C.L., Theobald, M. (eds.) GCAI 2017, 3rd Global Conf. on Artificial Intelligence. EPiC Series in Comp., vol. 50, pp. 121–133. EasyChair (2017)

7. Jakubuv, J., Urban, J.: Blistrtune: hierarchical invention of theorem proving strategies. In: Bertot, Y., Vafeiadis, V. (eds.) Proc. of the 6th ACM SIGPLAN Conf. on Certified Programs and Proofs, CPP. pp. 43–52. ACM (2017)
8. Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K.: ISAC - instance-specific algorithm configuration. In: Coelho, H., Studer, R., Wooldridge, M.J. (eds.) ECAI 2010 - 19th Eur. Conf. on Artificial Intelligence. Frontiers in Artificial Intelligence and Applications, vol. 215, pp. 751–756. IOS Press (2010)
9. Korovin, K.: iProver - an instantiation-based theorem prover for first-order logic (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) Automated Reasoning, 4th Int. Joint Conf., IJCAR 2008, Proc. LNCS, vol. 5195, pp. 292–298. Springer (2008)
10. Kovács, L., Voronkov, A.: First-order theorem proving and vampire. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification - 25th Int. Conf., CAV . Proc. LNCS, vol. 8044, pp. 1–35. Springer (2013)
11. Kühlwein, D., Schulz, S., Urban, J.: E-males 1.1. In: Bonacina, M.P. (ed.) Automated Deduction - CADE-24 - 24th Int. Conf. on Automated Deduction. Proc. LNCS, vol. 7898, pp. 407–413. Springer (2013)
12. Lindawati, Lau, H.C., Lo, D.: Instance-based parameter tuning via search trajectory similarity clustering. In: Coello, C.A.C. (ed.) Learning and Intelligent Optimization - 5th Int. Conf., LION 5. Selected Papers. LNCS, vol. 6683, pp. 131–145. Springer (2011)
13. Park, H., Jun, C.: A simple and fast algorithm for k-medoids clustering. Expert Syst. Appl. **36**(2), 3336–3341 (2009)
14. Paulson, L.C.: Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. In: Schmidt, R.A., Schulz, S., Konev, B. (eds.) Proc. of the 2nd Works. on Practical Aspects of Automated Reasoning, PAAR-2010. EPiC Series in Comp., vol. 9, pp. 1–10. EasyChair (2010)
15. Rawson, M., Reger, G.: Dynamic strategy priority: Empower the strong and abandon the weak. In: Konev, B., Urban, J., Rümmer, P. (eds.) Proc. of the 6th Works. on Practical Aspects of Automated Reasoning co-located with Federated Logic Conf. 2018 (FLoC 2018). CEUR Works. Proc., vol. 2162, pp. 58–71. CEUR-WS.org (2018)
16. Schäfer, S., Schulz, S.: Breeding theorem proving heuristics with genetic algorithms. In: Gottlob, G., Sutcliffe, G., Voronkov, A. (eds.) Global Conf. on Artificial Intelligence, GCAI. EPiC Series in Comp., vol. 36, pp. 263–274. EasyChair (2015)
17. Schneider, M., Hoos, H.H.: Quantifying homogeneity of instance sets for algorithm configuration. In: Hamadi, Y., Schoenauer, M. (eds.) Learning and Intelligent Optimization - 6th Int. Conf., LION 6, Revised Selected Papers. LNCS, vol. 7219, pp. 190–204. Springer (2012)
18. Schulz, S.: System description: E 1.8. In: McMillan, K.L., Middeldorp, A., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning - 19th Int. Conf., LPAR-19. Proc. LNCS, vol. 8312, pp. 735–743. Springer (2013)
19. Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. J. of Automated Reasoning **59**(4), 483–502 (2017)
20. Urban, J.: MPTP 0.2: Design, implementation, and initial experiments. J. Autom. Reason. **37**(1-2), 21–43 (2006)
21. Urban, J.: Blistr: The blind strategymaker. In: Gottlob, G., Sutcliffe, G., Voronkov, A. (eds.) Global Conf. on Artificial Intelligence, GCAI 2015. EPiC Series in Comp., vol. 36, pp. 312–319. EasyChair (2015)
22. Weidenbach, C., Dimova, D., Fietzke, A., Kumar, R., Suda, M., Wischnewski, P.: SPASS version 3.5. In: Schmidt, R.A. (ed.) Automated Deduction - CADE-22, 22nd Int. Conf. on Automated Deduction. Proc. LNCS, vol. 5663, pp. 140–145. Springer (2009)
23. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Satzilla: Portfolio-based algorithm selection for SAT. J. Artif. Intell. Res. **32**, 565–606 (2008)