Heuristic Subset Selection in Classical Planning

Levi H. S. Lelis
Santiago Franco
Marvin Abisrror
Dept. de Informática
Universidade Federal
de Viçosa, Brazil

Mike Barley
Comp. Science Dept.
Auckland University
Auckland, New Zealand

Sandra Zilles
Dept. of Comp. Science
University of Regina
Regina, Canada

Robert C. Holte
Dept. Comp. Science
University of Alberta
Edmonton, Canada

Abstract

In this paper we present greedy methods for selecting a subset of heuristic functions for guiding A* search. Our methods are able to optimize various objective functions while selecting a subset from a pool of up to thousands of heuristics. Specifically, our methods minimize approximations of A*'s search tree size, and approximations of A*'s running time. We show empirically that our methods can outperform state-of-the-art planners for deterministic optimal planning.

1 Introduction

The A* algorithm [Hart *et al.*, 1968] finds cost-optimal solutions to state-space planning problems when invoked with an admissible heuristic (a function that never overestimates the optimal solution cost for any state). There are various ways of generating admissible heuristics, e.g., pattern databases (PDBs) [Culberson and Schaeffer, 1998; Edelkamp, 2001], but the quality of such heuristics varies greatly.

One way of improving the quality of admissible heuristics is by maximizing over a set of them, an operation that yields an admissible heuristic which can overall be much more accurate than the original underlying heuristics. Ideally, one would generate a large number (say thousands) of admissible heuristics and maximize over them, because a larger set of heuristics increases the chance of having, for any given state, a more accurate heuristic estimate. However, evaluating thousands of heuristics would take too long to compute. We address this problem by proposing a method that carefully selects a subset ζ' of heuristics from a large pool ζ and then maximizes over ζ'. In other words, the final heuristic used to guide A*, h_{max}, is defined by $h_{max}(s, \zeta') = \max_{h \in \zeta'} h(s)$. Our approach is computationally efficient while still benefitting from the most helpful heuristics in the large pool.

We treat the problem of selecting a subset ζ' of ζ as an optimization problem, and present optimization methods for minimizing an approximation of A*'s search tree size (number of nodes generated by A*, denoted as J) and for minimizing an approximation of A*'s running time (denoted as T) while solving a planning task (J and T are defined in Section 4).

We present Greedy Heuristic Selection (GHS), a hill-climbing procedure that adds heuristics from ζ to ζ' one at a time and halts when adding another heuristic does not improve the objective function. GHS needs to efficiently estimate the value of the objective function for a possibly large number of heuristic subsets. We adapt two existing prediction algorithms to quickly provide estimates of J and T for any subset ζ' of ζ, namely, the method presented by Barley et al. [2014] and the one presented by Chen [1992].

Using the problems from the 2011 International Planning Competition (IPC), we empirically evaluate GHS in optimal classical planning problems while minimizing J, T, and maximizing the sum of heuristic values in the state space. Our experiments show that the subsets chosen by our algorithm can be far superior, in terms of coverage, to defining h_{max} over the entire collection ζ and to state-of-the-art methods. In particular, an optimization procedure that minimizes a combination of J and T outperforms all other approaches tested.

2 Related Work

The system most similar to GHS is RIDA* [Barley *et al.*, 2014]. RIDA* also selects a subset from a pool of heuristics to guide the A* search. RIDA* starts with an empty subset and tries all subsets of size one before trying subsets of size two and so on. RIDA* stops after evaluating a fixed number of subsets. While RIDA* is able to select from a pool with tens of heuristics, GHS is able to select from a pool with thousands of heuristics. We show empirically GHS's advantage from being able to evaluate a larger pool of heuristics.

Rayner *et al.* [2013] present an optimization procedure that is similar to ours. In contrast with our work, Rayner *et al.* limited their experiments to a single objective function that sought to maximize the sum of the heuristic values in the state space. Moreover, their method performs a uniform sampling of the reachable state space, and we are unaware of a general way of doing this in domain-independent planning. In this paper we adapt Rayner *et al.*'s approach to planning by using Chen's [1992] Stratified Sampling as its sampling procedure. Our empirical results show that GHS minimizing T substantially outperforms this adaptation of Rayner *et al.*'s approach in domain-independent planning.

The methods GA-PDB [Edelkamp, 2007] and iPDB [Haslum *et al.*, 2007] also combine a set of PDB heuristics. GA-PDB uses a genetic algorithm to maximize the average value of the resulting heuristic function. iPDB is based on the KRE prediction formula [Korf *et al.*, 2001] and it also uses the average heuristic value as a metric for deciding which PDBs to combine into a heuristic. Seipp *et al.* [2015] presented optimization procedures also aiming at maximizing the average heuristic value in the reachable state space while creating potential heuristics [Pommerening *et al.*, 2015]. In contrast with these works, the methods we introduce use richer prediction models such as Stratified Sampling [Chen, 1992] to select a subset of heuristics while minimizing J and T. Domshlak *et al.* [2012] introduced Selmax, a method for deciding which heuristic to use to evaluate a given node during search. Selmax evaluates only a few heuristics and their experiments consider sets with two heuristics; we consider sets with thousands of heuristics.

3 Background

An *SAS*$^+$ *planning task* [Bäckström and Nebel, 1995] is a 4-tuple $\nabla = \langle \mathcal{V}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$. $\mathcal{V}$ is a set of *state variables*. Each variable $v \in \mathcal{V}$ has a finite domain $\mathcal{D}_v$. A state is an assignment of a value in $\mathcal{D}_v$ to each $v \in \mathcal{V}$, so that the set of all states is $V = \mathcal{D}_{v_1} \times \cdots \times \mathcal{D}_{v_{|\mathcal{V}|}}$. $\mathcal{O}$ is a set of operators, where each operator $o \in \mathcal{O}$ is a triple $\langle pre_o, post_o, cost_o \rangle$ specifying the preconditions, postconditions (effects), and non-negative cost of o. Operator o is applicable to state s if s and pre_o agree on the assignment of values to variables in $\mathcal{V}_{pre_o}$. The effect of o, when applied to s, is to set the variables in $\mathcal{V}_{post_o}$ to the values specified in $post_o$ and to set all other variables to the value they have in s, resulting in a new *generated* state. We call the *children* of state s the states generated by applying the effects of each applicable operator of s (denoted as $children(s)$). Furthermore, s is *expanded* when all its children are generated. $\mathcal{G}$ is the goal condition, an assignment of values to a subset of variables, $\mathcal{V}_G$. A state is a goal state if it agrees on the assignment of values to the variables in $\mathcal{V}_G$. $\mathcal{I}$ is the initial state, and the planning task, ∇, is to find an optimal (least-cost) sequence of operators leading from $\mathcal{I}$ to a goal state. We denote the optimal solution cost of ∇ as C^*. We are interested only in optimal solutions. A heuristic $h(s)$ estimates the cost of a solution path from s to a goal. h is consistent iff $h(s) \leq c(s,t) + h(t)$ for all states s and t, where $c(s,t)$ is the cost of the cheapest path from s to t.

Given a set of consistent heuristics $\zeta = \{h_1, h_2, \cdots, h_M\}$, the heuristic $h_{max}(s, \zeta) = max_{h \in \zeta} h(s)$ is also consistent. When describing our methods we assume all heuristics to be consistent. We define $f_{max}(s, \zeta) = g(s) + h_{max}(s, \zeta)$, where $g(s)$ is the cost of the current path from $\mathcal{I}$ to s. We denote as $g^*(s)$ the cost of the least-cost path from $\mathcal{I}$ to s and note that $g(s) = g^*(s)$ when A* using a consistent heuristic expands s. An *A* search tree* is defined by the states generated by A* using a consistent heuristic while solving a problem ∇. In this paper we use the terms state and node interchangeably.

4 Greedy Heuristic Selection

We introduce Greedy Heuristic Selection (GHS) an algorithm able to find good solutions to the following problem,

$$\underset{\zeta' \subseteq \zeta}{\textbf{minimize}} \quad \Psi(\zeta', \nabla) \,, \tag{1}$$

where Ψ is either J or T (formally defined below).

Algorithm 1 Greedy Heuristic Selection

```
Input: problem $\nabla$, set of heuristics $\zeta$
Output: heuristic subset $\zeta' \subseteq \zeta$
1: $\zeta' \leftarrow \emptyset$
2: while $\zeta' \neq \zeta$ do
3:     $h \leftarrow \operatorname{argmin}_{h \in \zeta} \Psi(\zeta' \cup \{h\}, \nabla)$
4:     if $\Psi(\zeta' \cup \{h\}, \nabla) \geq \Psi(\zeta', \nabla)$ then
5:         return $\zeta'$
6:     $\zeta' \leftarrow \zeta' \cup \{h\}$
7: return $\zeta'$
```

Algorithm 1 shows GHS, which receives as input a problem ∇, a set of heuristics ζ, and it returns a subset $\zeta' \subseteq \zeta$. In each iteration i GHS selects a heuristic h from ζ and adds h to ζ', where h is the heuristic that when individually combined with ζ' reduces Ψ the most. GHS returns ζ' once it can no longer reduce the value of Ψ (line 4) or when $\zeta = \zeta'$ (line 2).

Next, we describe two functions Ψ we consider: an approximation of J, and an approximation of T. Although often one is interested in minimizing a planner's running time, we believe that J is also an important metric to be considered. This is because planners often run out memory before running out of time while solving planning problems, and J could serve as a good proxy for memory usage of A*-based planners.

4.1 Approximation of A*'s Search Tree Size

The first objective function Ψ we consider is $J(\zeta', \nabla)$, an upper bound on the size of the search tree A* generates while solving planning problem ∇ using $h_{max}(\zeta')$,

$$J(\zeta', \nabla) = \left| \bigcup\nolimits_{s \text{ with } f_{max}(s, \zeta') \leq C^*} children(s) \right| , \tag{2}$$

where $\bigcup_{s \text{ with } f_{max}(s, \zeta') \leq C^*} children(s)$ is a multiset, as in an A* search the same state could be generated multiple times. $J(\zeta', \nabla)$ assumes that A* expands all states s with $f_{max}(s, \zeta') \leq C^*$ before expanding a goal state. We write $J(\zeta')$ or simply J instead of $J(\zeta', \nabla)$ whenever ζ' and ∇ are clear from the context. We present in this paper two methods for approximating J. We now show that $J(\zeta', \nabla)$ is minimal, for the subset ζ' returned by GHS.

Theorem 1 *Let the heuristics in ζ' as well as $h \notin \zeta'$ be consistent heuristics. We have that $J(\zeta' \cup \{h\}) \leq J(\zeta')$.*

Proof. Fix ζ' and h. Then $J(\zeta' \cup \{h\}) =$

$$\begin{aligned} &= \left| \bigcup\nolimits_{s \text{ with } h_{max}(s, \zeta' \cup \{h\}) \leq C^* - g^*(s)} children(s) \right| \\ &\leq \left| \bigcup\nolimits_{s \text{ with } h_{max}(s, \zeta') \leq C^* - g^*(s)} children(s) \right| \\ &= J(\zeta') \,, \end{aligned}$$

where the inequality follows from the fact that $h_{max}(s, \zeta' \cup \{h\}) \geq h_{max}(s, \zeta')$ for all s. □

Corollary 1 $J(\zeta, \nabla) \leq J(\zeta', \nabla)$ *for any* $\zeta' \subseteq \zeta$.

Lemma 1 *Let ∇ be a planning problem with optimal solution cost C^*, A a set of consistent heuristic functions, and $B \subseteq A$. If $J(A, \nabla) < J(B, \nabla)$, then there exists an element $h \in A$ such that $J(B \cup \{h\}, \nabla) < J(B, \nabla)$.*

Proof. If $J(A, \nabla) < J(B, \nabla)$, then by the definition of J there is a state s such that $f_{max}(s, A) > C^*$ and $f_{max}(s, B) \leq C^*$. Thus, there is a heuristic $h \in A$ such that $h(s) > C^* - g^*(s) \geq h_{max}(s, B)$. Then, $h_{max}(s, B \cup \{h\}) > C^* - g^*(s)$ follows from the definition of h_{max}, and consequently $J(B \cup \{h\}, \nabla) < J(B, \nabla)$. □

Theorem 2 *Given a set ζ of consistent heuristics,* `GHS` *returns a subset ζ' of ζ for which $J(\zeta) = J(\zeta')$.*

Proof. `GHS` stops when no $h \in \zeta$ satisfies $J(\zeta' \cup \{h\}) < J(\zeta')$. Thus, $J(\zeta) < J(\zeta')$ cannot be true (Lemma 1). Since $J(\zeta)$ is minimal (Corollary 1), $J(\zeta') = J(\zeta)$. □

4.2 Approximation of A*'s Running Time

Another Ψ we consider is $T(\zeta', \nabla)$, an approximation to the running time of A* when using $h_{max}(\zeta')$ for solving ∇.

$$T(\zeta', \nabla) = J(\zeta', \nabla) \times (t_{h_{max}(\zeta')} + t_{gen}) , \qquad (3)$$

where $t_{h_{max}(\zeta')}$ is the heuristic evaluation time of $h_{max}(\zeta')$, and t_{gen} is the node generation time. We assume $t_{h_{max}(\zeta')}$ and t_{gen} to be constant throughout the state space for a given domain and heuristic. This is reasonable to assume for several heuristics, *e.g.,* PDBs, and several types of problem domains. We write $T(\zeta')$ or simply T instead of $T(\zeta', \nabla)$ whenever ζ' and ∇ are clear from the context.

$T(\zeta', \nabla)$ represents only an approximate model of the A* running time. For an exact computation we would have to also account for other factors such as the time spent in operations on A*'s open list. However, it is usually hard to approximate the running time of such operations because they could vary during the search. Nevertheless, T captures the computational cost of the heuristic function being employed, which we expect to dominate the running time of planning systems.

Next we show that if all $h \in \zeta$ have the same evaluation time t_h, then the subset ζ' `GHS` returns is guaranteed to have the lowest T-value amongst all subsets of size $|\zeta'|$ or larger.

Theorem 3 *Let ζ be a set of consistent heuristics. If $t_{h_i} = t_{h_j}$ for any $h_i, h_j \in \zeta$, then* `GHS` *yields a subset ζ' of ζ with $T(\zeta', \nabla) \leq T(\zeta'', \nabla)$ for any $\zeta'' \subseteq \zeta$ with $|\zeta'| \leq |\zeta''|$.*

Proof.

$$\begin{aligned} T(\zeta', \nabla) &= J(\zeta', \nabla) \times (t_{h_{max}(\zeta')} + t_{gen}) \\ &\leq J(\zeta', \nabla) \times (t_{h_{max}(\zeta'')} + t_{gen}) \\ &\leq J(\zeta'', \nabla) \times (t_{h_{max}(\zeta'')} + t_{gen}) = T(\zeta'', \nabla) . \end{aligned}$$

The first inequality follows from $t_{h_{max}(\zeta')} \leq t_{h_{max}(\zeta'')}$ (because $|\zeta'| \leq |\zeta''|$ and t_h is constant for any $h \in \zeta$), the second from $J(\zeta', \nabla)$ being minimal (Theorem 2). □

5 Estimating Tree Size and Running Time

In practice `GHS` uses approximations $\hat{J}$ of J and $\hat{T}$ of T instead of their exact values. This is because computing J and T exactly could be as expensive as solving ∇.

We investigate two methods for estimating $\hat{J}$ and $\hat{T}$, Culprit Sampling (`CS`) [Barley *et al.*, 2014] and Stratified Sampling (`SS`) [Chen, 1992]. Both `CS` and `SS` must be able to quickly estimate the values of $\hat{J}(\zeta')$ and $\hat{T}(\zeta')$ for any subset ζ' of ζ so that they can be used in `GHS`'s optimization process. This is achieved by estimating *f-culprits* and *b-culprits*.

Definition 1 (f-culprit) *Let $\zeta = \{h_1, h_2, \cdots, h_M\}$ be a set of heuristics. The f-culprit of a node n in an A* search tree is defined as the tuple $F(n) = \langle f_1(n), f_2(n), \cdots, f_M(n)\rangle$, where $f_i(n) = g(n) + h_i(n)$. For any tuple F, the counter C_F denotes the total number of children of nodes n in the search tree with $F(n) = F$.*

Definition 2 (b-culprit) *Let $\zeta = \{h_1, h_2, \cdots, h_M\}$ be a set of heuristics and b a lower bound on the solution cost of ∇. The b-culprit of a node n in an A* search tree is defined as the tuple $B(n) = \langle y_1(n), y_2(n), \cdots, y_M(n)\rangle$, where $y_i(n) = 1$ if $g(n) + h_i(n) \leq b$ and $y_i(n) = 0$, otherwise. For any binary tuple B, the counter C_B denotes the total number of children of nodes n in the search tree with $B(n) = B$.*[1]

We now describe how `CS` and `SS` estimate f- and b-culprits and how the culprits can be used to quickly estimate $\hat{J}(\zeta')$ and $\hat{T}(\zeta')$ for any subset ζ' of ζ.

5.1 Culprit Sampler (CS)

`CS` runs an A* search bounded by a user-specified time limit and then compresses the information obtained in the A* search (i.e., the f-values of all nodes expanded according to all heuristics h in ζ) into b-culprits, which are later used for computing $\hat{J}$. The f-culprits are needed for computing the b-culprits, as we explain below. The maximum number of f-culprits and b-culprits in an A* search tree equals the number of nodes in the tree expanded by the time-bounded A* search. However, in practice the total number of f- and b-culprits is usually much lower than the number of nodes in the tree.

Given ∇, ζ, and a user-specified time limit, `CS` samples the A* search tree as follows.

1. `CS` runs A* using $h_{max}(s, \zeta)$ for one fourth of the user-specified time limit. We denote as OPEN the set of states in the A*'s open list when A* halts.
2. In the remainder of its running time, `CS` randomly selects a state s from OPEN and computes $h_{min}(s, \zeta) = \min_{h \in \zeta} h(s)$ for s, as well as for every node s' in s's subtree for which $f_{min}(s, \zeta) = f_{min}(s', \zeta)$, where $f_{min}(s, \zeta) = h_{min}(s, \zeta) + g(s)$. For each node sampled in this procedure, we store its f-culprit and its counter. We use h_{min} because A* using $h_{min}(s, \zeta)$ expands node s if it were to expand s with any $h \in \zeta$ individually.
3. Let f_{maxmin} be the largest f-value according to h_{min} encountered in `CS`'s sampling procedure. We compute the set $\mathbf{B}$ of b-culprits and their counters based on the f-culprits and on the value of f_{maxmin}. This is done by iterating over all f-culprits once.

`CS` is run only once in `GHS`'s execution. The value of $\hat{J}(\zeta', \nabla)$ for any subset ζ' of ζ is then computed by iterating over all b-culprits $\mathbf{B}$ and summing up the relevant values of C_B. The relevant values of C_B represent the number of

[1]The vector $B(n)$, and thus the values of C_B for any B, depend on the bound b, which we assume to be fixed.

nodes A* would generate in a search bounded by b if using $h_{max}(\zeta')$. This computation is written as, $\hat{J}(\zeta', \nabla) = \sum_{B \in \mathbf{B}} W(B)$, where $W(B)$ is 0 if there is a heuristic in ζ' whose y-value in B is zero (i.e., there is a heuristic in ζ' that prunes all nodes compressed into B), and C_B otherwise.

In addition to computing $\hat{J}$, the value of $\hat{T}$ requires approximations of t_h for all $h \in \zeta$ and of t_{gen}. These values are measured in a separate process, before executing CS, by sampling a small number of nodes from ∇'s start state.

5.2 Stratified Sampling (SS)

Chen [1992] presented Stratified Sampling (SS), a method for estimating the search tree size of backtracking search algorithms. In this section we briefly describe Chen's SS. For a detailed description of SS, see Lelis *et al.* [2013].

SS approximates any function $\varphi = \sum_{n \in S} z(n)$, where S is the set of nodes in a search tree and z is a function assigning a numerical value to a node. φ represents a numerical property of the tree. For GHS, $z(n) = |children(n)|$, so that φ is the number of nodes generated by the search procedure.

SS works by sampling the search tree with a procedure similar to random walks. SS's sampling procedure is limited by a value d. That is, SS does not expand node n if $f(n) > d$. Each sample of SS is called a probe. Chen proved that as the number of probes grows large, SS's estimate of φ converges to its exact value. As Lelis *et al.* [2014b] showed, SS does not detect duplicate nodes in its sampling procedure. Hence, as A* does not expand duplicates, SS's predictions usually overestimates A*'s search tree size. We show empirically that SS is still able to allow GHS to select good subsets.

Similar to CS, we define a time limit for SS and use an iterative-deepening approach to ensure an estimate of $\hat{J}$ and $\hat{T}$ before reaching the time limit. SS's sampling is initially limited by the h-value of the start state ($d = h_{max}(\mathcal{I}, \zeta)$). After p probes, if there is still time, we double the value of d. The values of $\hat{J}$ and $\hat{T}$ are given by the prediction produced for the last d-value in which SS finishes all p probes. If SS cannot finish all p probes during its first iteration, $\hat{J}$ and $\hat{T}$ are given by the prediction from the incomplete first iteration.

SS must also estimate the values of $\hat{J}(\zeta')$ and $\hat{T}(\zeta')$ for any subset ζ' of ζ. This is achieved by using SS to estimate b-culprits (see Definition 2) instead of the search tree size directly. Similar to CS, SS uses h_{min} of the heuristics in ζ to decide when to prune a node while sampling. This ensures that SS expands a node n if A* employing at least one of the heuristics in ζ would expand n according to bound d.

6 The Hybrid Approach

In this section we introduce an approach called Hybrid, that minimizes both $\hat{J}$ and $\hat{T}$ while using SS and CS. That is, Hybrid uses GHS+SS to select a subset of heuristics from a pool ζ_1 composed of heuristics with the same evaluation time. We call the selection of a subset of ζ_1 the *first selection*. Once the first selection $\zeta' \subseteq \zeta_1$ is made, Hybrid tests all possible combinations of the resulting $h_{max}(\zeta')$ with heuristics from another pool ζ_2, composed of heuristics with possibly different evaluation times, while minimizing $\hat{T}$ as estimated by CS. We call this step the *second selection*.

The intuition behind Hybrid is to apply GHS with its strongest settings, i.e., according to Theorems 2 and 3, the subset ζ' GHS chooses in the first selection is minimal with respect to $\hat{J}$ and minimal amongst all subsets whose size is equal or larger than $|\zeta'|$ with respect to $\hat{T}$ (all $h \in \zeta_1$ have the same evaluation time). After such a selection is made, we reduce all heuristics in ζ_1 to a single heuristic $h_{max}(\zeta')$. Then, if ζ_2 has only a few heuristics, we are able to select from $\{h_{max}(\zeta')\} \cup \zeta_2$ the exact subset that minimizes $\hat{T}$. In our experiments ζ_1 may have thousands of heuristics, depending on the problem instance, while ζ_2 has always only two.

We use SS instead of CS in the first selection because the former is able to make better subset selections while minimizing $\hat{J}$, as shown in Section 7.1. Finally, as indicated in our experiments shown in Section 7.2, CS is more effective if one is interested in minimizing $\hat{T}$ while selecting from a pool of heuristics with different evaluation times. That is why we use CS as a predictor for Hybrid's second selection.

7 Empirical Evaluation

The practical effectiveness of GHS depends on its ability to find good approximations $\hat{J}$ and $\hat{T}$ and on the quality of ζ. In order to verify its practical effectiveness, we have implemented GHS in Fast Downward [Helmert, 2006] and tested the A* performance using subsets of heuristics selected by GHS while minimizing different objective functions.

We run two sets of experiments. In the first set we verify whether the approximations $\hat{J}$ and $\hat{T}$ provided by CS and SS allow GHS to make good subset selections. In the second set of experiments we test the effectiveness of GHS by measuring the total number of problem instances solved by A* using the heuristic subset selected by GHS.

In our experiments we first generate and store in memory all heuristics in ζ, and only then we use GHS to select a subset ζ' of ζ. The process of generating ζ is limited by 1GB of memory and 600 seconds. The sampling procedure is bounded by a 300-second time limit for both CS and SS.

SS requires one to define an equivalence relation of the nodes in the search tree. In our experiments we consider nodes equivalent if they have the same f-value. This equivalence relation has shown to be effective in guiding SS in other application domains [Lelis *et al.*, 2013; 2014a]. We use the number of SS probes $p = 500$ in all our experiments.

We ran our experiments on the 2011 IPC instances instead of the 2014 instances because the former do not have domains with conditional effects, which are currently not handled by the regression search that builds PDB heuristics. The 2011 IPC benchmark has 280 instances of 14 different domains (20 instances per domain). All experiments are run on 2.67 GHz machines with 4 GB, and are limited to 1,800 seconds of running time. We use the preprocessor described by Alcázar and Torralba [2015] for mutex detection in all planners tested.

7.1 Empirical Evaluation of $\hat{J}$ and $\hat{T}$

GHS is guaranteed to return a subset ζ' that minimizes J. We now test whether the $\hat{J}$-values provided by CS and SS allow

Domain	SS		CS		$\|\zeta\|$	n
	Ratio	$\|\zeta'\|$	Ratio	$\|\zeta'\|$		
Barman	1.11	17.70	1.50	30.25	5,168.50	20
Elevators	11.50	2.00	1.04	21.00	168.00	1
Floortile	1.02	43.07	1.02	42.36	151.29	14
Openstacks	1.00	1.00	1.00	1.00	390.69	13
Parking	1.00	5.53	1.01	7.26	21.74	19
Parcprinter	3.62	1.00	2.21	13.00	1,189.00	1
Pegsol	1.00	31.00	1.00	57.00	90.00	2
Scanalyzer	1.23	30.57	1.57	19.43	72.86	7
Sokoban	1.32	7.00	1.01	24.00	341.00	1
Tidybot	1.00	2.35	1.00	8.59	3,400.18	17
Transport	1.00	14.70	1.03	14.30	171.70	10
Visitall	1.03	99.33	1.19	48.67	256.33	3
Woodworking	32.43	3.00	199.66	5.00	1,289.00	5

Table 1: Ratios of the number of nodes generated using $h_{max}(\zeta')$ to the number of nodes generated using $h_{max}(\zeta)$.

GHS to make good subset selections. This test is made by comparing $J(\zeta')$ with the number of nodes generated by A* while using $h_{max}(\zeta)$, which is minimal.

In contrast with objective function J, there is no easy way to find the minimum of T in general. We experiment then with the special case in which all heuristics in ζ have the same evaluation time. This way, if GHS selects a subset ζ' which is optimal with respect to J, then we know that ζ' has the lowest T-value amongst all subsets of size $|\zeta'|$ or larger (Theorem 3).

We collect values of $J(\zeta)$ and $J(\zeta')$ as follows. For each instance ∇ in our test set we generate a set of PDB heuristics using the GA-PDB algorithm [Edelkamp, 2007] as described by Barley *et al.* [2014] (we call each PDB generated by this method a GA-PDB). We chose to use only GA-PDBs in this experiment as they all have nearly the same evaluation time. The number of GA-PDBs generated is limited in this experiment by 600 seconds and 1GB of memory. The GA-PDBs generated form our ζ set. GHS then selects a subset ζ' of ζ. Finally, we use $h_{max}(\zeta')$ and $h_{max}(\zeta)$ to independently try to solve ∇. We call the system which uses A* with $h_{max}(\zeta)$ the Max approach. For GHS we allow 1,200 seconds in total for both selecting ζ' and for running A* with $h_{max}(\zeta')$, and for Max we allow 1,200 seconds for running A* with $h_{max}(\zeta)$. Since we used 600 seconds to generate the heuristics, both Max and GHS were allowed 1,800 seconds in total for solving each problem. We test both CS and SS.

We refer to the approach that runs A* guided by a heuristic subset selected by GHS using CS as GHS+CS. Similarly, we write GHS+SS when SS is used as predictor.

Table 1 shows the average ratios of $J(\zeta')$ to $J(\zeta)$ for both SS and CS in different problem domains. The value of J, for a given problem instance, is computed as the number of nodes generated up to the largest f-layer which is fully expanded by all approaches tested (Max, GHS+SS, and GHS+CS). We only present results for instances that are not solved during GHS's CS sampling process. The column "n" shows the number of instances used to compute the averages of each row. We also show the average number of GA-PDBs generated ($|\zeta|$) and the average number of GA-PDBs selected by GHS ($|\zeta'|$). We see that for most of the problems GHS, using either CS or SS, is selecting a near-optimal subset of ζ. For example, in Tidybot GHS selects only a few heuristics out of thousands when using either SS or CS, and the resulting A* search tree size is optimal for both approaches.

The exceptions in Table 1 are the ratios for Elevators, Parcprinter, and Woodworking. In all three domains GHS+SS failed to make good heuristic selections because SS was not able to sample "deep enough" into the problem's search tree. For instance, for the Elevators instance SS was able to perform only 245 probes with the initial f-boundary within the time limit of 300 seconds. GHS+CS is able to make better selections for this instance because it is able to sample deeper into the search tree. For the Parcprinter instance as well as the Woodworking instances neither SS nor CS are able to sample deep enough to make good heuristic selections. These results suggest that SS could benefit from an adaptive approach for choosing SS's number of probes. For example, for the Elevators instance SS could have performed better by sampling deeper into the tree with fewer probes. We intend to investigate this direction in future works.

In general SS is able to sample deeper than CS into the search tree, which allowed GHS+SS to solve more instances in this experiment. In total, out of the 280 instances, GHS+SS solved 199 instances, while GHS+CS solved 194, and Max 182 (these numbers are not shown in Table 1).

7.2 Comparison with Other Planning Systems

The objective of this second set of experiments is to compare GHS with state-of-the-art planners. Our evaluation metric is coverage, i.e., number of problems solved within a 1,800-second time limit, which includes the time to generate ζ, select ζ', and run A* using $h_{max}(\zeta')$. The ζ set of heuristics is composed of a number of different GA-PDBs, a PDB heuristic produced by the iPDB method [Haslum *et al.*, 2007] and the LMCut heuristic [Pommerening and Helmert, 2013]. The generation of GA-PDBs is limited by 600 seconds and 1 GB of memory. We use one third of 600 seconds to generate GA-PDBs with each of the following maximum numbers of entries: $\{2 \cdot 10^4, 2 \cdot 10^5, 2 \cdot 10^6\}$. Our approach generates up to thousands of GA-PDBs. We use exactly the same ζ set for Max and all GHS approaches.

7.3 Systems Tested

GHS is tested for minimizing $\hat{J}$ (Size) and $\hat{T}$ (Time). We also use GHS to maximize the sum of heuristic values in the state space (Sum), as suggested by Rayner *et al.* [2013]. Rayner *et al.* assumed that one could uniformly sample states in the state space in order to estimate the sum of the heuristic values for a given heuristic subset, which in general cannot be done in domain-independent planning. Thus, we adapted Rayner *et al.*'s method by using SS to estimate the sum of heuristic values in the search tree rooted at ∇'s start state. Below, Size+SS refers to the approach that uses A* guided by a heuristic selected by GHS while minimizing a $\hat{J}$ provided by SS. We name other combinations of objective functions and prediction algorithms in a similar way (e.g., Time+CS).

In addition to testing all combinations of prediction algorithms (CS and SS) and objective functions (Time, Size) we also experiment with Hybrid using a pool ζ_1 composed solely of GA-PDB heuristics, and ζ_2 of iPDB and LMCut.

Domains	Hybrid	CS		SS		Sum	Max	RIDA*	SY1	SY2	StSp1	StSp2	iPDB	LMCut	M&S
		Time	Size	Time	Size										
Barman	7	5	4	4	4	4	4	4	10	11	4	4	4	4	4
Elevators	19	19	19	19	19	19	19	19	20	20	18	18	18	17	12
Floortile	15	14	14	14	14	14	14	14	14	14	14	14	14	8	10
Nomystery	20	20	20	19	19	20	20	20	16	16	20	20	14	19	18
Openstacks	17	17	15	17	15	15	11	15	20	20	17	17	15	17	17
Parcprinter	18	18	18	16	15	19	18	18	17	17	18	18	17	16	16
Parking	7	7	2	7	2	2	2	7	2	1	5	5	2	7	7
Pegsol	18	18	19	19	19	19	19	19	19	20	19	19	17	20	19
Scanalyzer	13	14	12	11	14	14	14	14	9	9	14	14	12	10	11
Sokoban	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
Tidybot	17	16	16	16	16	16	15	17	15	17	16	16	16	14	9
Transport	14	13	10	11	13	11	9	10	10	11	7	8	6	8	7
Visitall	18	18	18	15	18	18	18	18	12	12	16	16	10	16	16
Woodworking	16	15	15	12	16	16	16	15	20	20	15	15	15	9	9
Total	219	214	202	200	204	207	199	210	204	208	203	204	180	185	175

Table 2: Coverage of different planning systems on the 2011 IPC benchmarks.

We compare the coverage of the GHS approaches with the following state-of-the-art planners, RIDA* [Barley *et al.*, 2014], two variants of StoneSoup (StSp1 and StSp2) [Helmert *et al.*, 2011], two versions of Symba (SY1 and SY2) [Torralba, 2015]. We also independently run A* with the following heuristics: maximum of all heuristics in ζ (Max), iPDB, LMCut, and Merge & Shrink (M&S) as described by Nissim *et al.* [2011]. Table 2 presents the results.

7.4 Discussion of the Results

The system that solves the largest number of instances is Hybrid (219 in total). Its combination of the strengths of both SS and CS has proven particularly effective on the Barman domain where Hybrid's first selection contains good subsets of GA-PDBs and its second selection recognizes when it must not add the iPDB and LMCut heuristics to the first selection. As a result, Hybrid solves more problems on this domain than any other GHS approach.

Time+CS also performs well—it solves 214 problems. Hybrid and Time+CS are superior to all other approaches tested. When minimizing $\hat{J}$ or maximizing Sum, GHS tends to add accurate heuristics to the selected subset, irrespective of their evaluation time. Thus, GHS frequently selects LMCut which is often the heuristic that most reduces the search tree size and most increases the sum of heuristic values. However, LMCut is computationally expensive, and often the search is faster if LMCut is not in ζ'. Both Hybrid and Time+CS often recognize when LMCut should not be in ζ' because they account for the heuristics' evaluation time.

Interestingly, while Time+CS solves 214 instances, Time+SS solves only 200. We conjecture that this is due to SS not detecting duplicate nodes during sampling and thus substantially overestimating A*'s running time. As a result, similarly to the Size and Sum approaches, Time+SS often mistakenly adds the accurate but expensive LMCut heuristic in cases where A* would be faster without LMCut.

RIDA* is the most similar system to GHS; it selects a subset of heuristics by using an evaluation method similar to CS. Starting with an empty subset, it evaluates all subsets of size i before evaluating subsets of size $i+1$. This limits RIDA* to considering only tens of heuristics in its pool. Specifically, RIDA* uses 42 GA-PDBs, iPDB, and LMCut in its pool. By contrast, GHS may consider thousands of heuristics.

Selecting from large sets of heuristics can be helpful, even if most of the heuristics in the set are redundant with each other—as is the case with the GA-PDBs. The process of generating GA-PDBs is stochastic, thus one increases the chances of generating a helpful heuristic by generating a large number of them. GHS is an effective method for selecting a small set of informative heuristics from a large set of mostly uninformative ones. This is illustrated in Table 2 on the Transport domain. Compared to systems which use multiple heuristics (StSp 1 and 2, and RIDA*), Hybrid solves the largest number of Transport instances, which is due to the selection of a few key GA-PDBs.

The best GHS approach, Hybrid, substantially outperforms Max; Hybrid solves 20 more instances than Max. Finally, Hybrid and Time+CS outperform all other approaches tested, with RIDA* being the closest competitor with 210 instances solved.

8 Concluding Remarks

In this paper we presented GHS, a greedy optimization algorithm that chooses a subset from a pool of thousands of heuristics. We used GHS to select a subset of heuristics while minimizing an approximation of A*'s search tree size ($\hat{J}$) and an approximation of A*'s running time ($\hat{T}$)—two objective functions we proposed in this paper. In addition to the two objective functions we proposed, we also experimented with an objective function that accounts for the sum of heuristic values in the state space (Sum). We tested two methods, CS and SS, for estimating the values of the objective functions tested. Several previous works used Sum (or variations of it) for constructing heuristics (*e.g.,* [Haslum *et al.*, 2007; Edelkamp, 2007; Rayner *et al.*, 2013; Seipp *et al.*, 2015]). Our experiments on optimal domain-independent problems showed that GHS using a hybrid approach that minimizes a combination of $\hat{J}$ and $\hat{T}$ outperformed all other approaches tested, including the approach that maximizes Sum.

9 Acknowledgements

The authors gratefully thank an anonymous referee of an earlier version of this paper for suggesting important improvements to the theoretical contributions of this work.

Financial support for this research was in part provided by Brazil's CAPES (Science Without Borders), FAPEMIG, CNPq, and Canada's NSERC (Discovery Grant program and Canada Research Chairs program).

This material is based upon work supported by the Air Force Office of Scientific Research, Asian Office of Aerospace Research and Development (AOARD) under award number FA2386-15-1-4069. Any opinions, finding, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force.

References

[Alcázar and Torralba, 2015] V. Alcázar and A. Torralba. A reminder about the importance of computing and exploiting invariants in planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 2–6, 2015.

[Bäckström and Nebel, 1995] C. Bäckström and B. Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11:625–656, 1995.

[Barley *et al.*, 2014] M. Barley, S. Franco, and P. Riddle. Overcoming the utility problem in heuristic generation: Why time matters. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 38–46, 2014.

[Chen, 1992] P.-C. Chen. Heuristic sampling: A method for predicting the performance of tree searching programs. *SIAM Journal on Computing*, 21:295–315, 1992.

[Culberson and Schaeffer, 1998] J. C. Culberson and J. Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

[Domshlak *et al.*, 2012] C. Domshlak, E. Karpas, and S. Markovitch. Online speedup learning for optimal planning. *Journal of Artificial Intelligence Research*, 44:709–755, 2012.

[Edelkamp, 2001] S. Edelkamp. Planning with pattern databases. In *Proceedings of the European Conference on Planning*, pages 13–24, 2001.

[Edelkamp, 2007] S. Edelkamp. Model checking and artificial intelligence. In *MoChart*, chapter Automated Creation of Pattern Database Search Heuristics, pages 35–50. Springer-Verlag, 2007.

[Hart *et al.*, 1968] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(2):100–107, 1968.

[Haslum *et al.*, 2007] P. Haslum, A. Botea, M. Helmert, B. Bonet, and S. Koenig. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1007–1012. AAAI Press, 2007.

[Helmert *et al.*, 2011] M. Helmert, G. Röger, and E. Karpas. Fast downward stone soup: A baseline for building planner portfolios. In *In Proceedings of the Workshop on Planning and Learning*, pages 28–35. AAAI, 2011.

[Helmert, 2006] M. Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[Korf *et al.*, 2001] R. E. Korf, M. Reid, and S. Edelkamp. Time complexity of Iterative-Deepening-A*. *Artificial Intelligence*, 129(1-2):199–218, 2001.

[Lelis *et al.*, 2013] L. H. S. Lelis, S. Zilles, and R. C. Holte. Predicting the size of IDA*'s search tree. *Artificial Intelligence*, pages 53–76, 2013.

[Lelis *et al.*, 2014a] L. H. S. Lelis, L. Otten, and R. Dechter. Memory-efficient tree size prediction for depth-first search in graphical models. In *Proceedings of the Principles and Practice of Constraint Programming*, pages 481–496, 2014.

[Lelis *et al.*, 2014b] L. H. S. Lelis, R Stern, and N. R. Sturtevant. Estimating search tree size with duplicate detection. In *Proceedings of the Seventh Annual Symposium on Combinatorial Search*, pages 114–122. AAAI Press, 2014.

[Nissim *et al.*, 2011] R. Nissim, J. Hoffmann, and M. Helmert. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1983–1990. AAAI, 2011.

[Pommerening and Helmert, 2013] F. Pommerening and M. Helmert. Incremental lm-cut. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 162–170, 2013.

[Pommerening *et al.*, 2015] F. Pommerening, M. Helmert, G. Röger, and J. Seipp. From non-negative to general operator cost partitioning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 3335–3341, 2015.

[Rayner *et al.*, 2013] D. C. Rayner, N. R. Sturtevant, and M. Bowling. Subset selection of search heuristics. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 637–643. AAAI Press, 2013.

[Seipp *et al.*, 2015] J. Seipp, F. Pommerening, and M. Helmert. New optimization functions for potential heuristics. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 193–201, 2015.

[Torralba, 2015] A. Torralba. *Symbolic Search and Abstraction Heuristics for Cost-Optimal Planning*. PhD thesis, Universidad Carlos III de Madrid, 2015.