Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling

New Optimization Functions for Potential Heuristics

Jendrik Seipp and **Florian Pommerening** and **Malte Helmert**
University of Basel
Basel, Switzerland
{jendrik.seipp,florian.pommerening,malte.helmert}@unibas.ch

Abstract

Potential heuristics, recently introduced by Pommerening et al., characterize admissible and consistent heuristics for classical planning as a set of declarative constraints. Every feasible solution for these constraints defines an admissible heuristic, and we can obtain heuristics that optimize certain criteria such as informativeness by specifying suitable objective functions.

The original paper only considered one such objective function: maximizing the heuristic value of the initial state. In this paper, we explore objectives that attempt to maximize heuristic estimates for all states (reachable and unreachable), maximize heuristic estimates for a sample of reachable states, maximize the number of detected dead ends, or minimize search effort. We also search for multiple heuristics with complementary strengths that can be combined to obtain even better heuristics.

Introduction

Heuristic search with admissible heuristics is commonly used for optimal classical planning. Recently, Pommerening et al. (2015) introduced *potential heuristics*, which associate a numerical potential with each fact of the planning task and calculate a state's heuristic estimate by summing over the potentials of its facts. Linear constraints on the potentials characterize exactly the admissible and consistent potential heuristics. The feasible solutions of these constraints thus form a space of suitable heuristics, and linear programming techniques can easily extract heuristic functions that optimize given criteria from this space.

Pommerening et al. (2015) show that the heuristic optimizing the potentials for every encountered state separately is equal to the state equation heuristic h^{SEQ} (van den Briel et al. 2007; Bonet 2013). In contrast to the state equation heuristic, potential heuristics are only optimized once and then use the same potential function throughout the search. Since optimizing the heuristic for each state separately achieves the maximal heuristic estimates, h^{SEQ} is an upper bound on the heuristic quality we can achieve with any optimization function. Our goal is therefore to find optimization functions that approximate h^{SEQ} as closely as possible and are faster to compute than h^{SEQ}.

Copyright © 2015, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Pommerening et al. use potential heuristics that maximize the heuristic estimate of the initial state. This choice ignores the heuristic values of other states encountered during the search, so other optimization functions probably yield better informed heuristics. Instead of focusing on the initial state, one possibility that we consider is to optimize heuristics for a high average heuristic value on all states. These heuristics significantly outperform the potential heuristic optimized for the initial state.

However, these heuristics are still problematic for two reasons. In the presence of dead ends, the average heuristic value can become arbitrarily high, even if just one state has a high estimate. Also, it is unnecessary to optimize the heuristic values of unreachable states. Therefore, we also try to find potential heuristics that are good at detecting dead ends and have high heuristic values in the reachable part of the search space.

Instead of maximizing heuristic values for a given set of states, we can also look for heuristics that minimize the expected search effort. Different ways of estimating search effort have been suggested; we use a formula by Korf, Reid, and Edelkamp (2001) and optimize heuristics that assume different models of the search space. It turns out that optimizing the average heuristic value of a set of samples corresponds to the assumption that optimal costs of nodes are uniformly distributed, i.e., there are equally many nodes at any distance from the initial state.

Since potential heuristics are extremely fast to evaluate during the search, and are usually fast to precompute, we also experiment with using multiple heuristics simultaneously. As a baseline, we calculate multiple potential heuristics with a randomized approach. We compare this to an algorithm that explicitly tries to find diverse heuristics that have complementary strengths in different parts of the search space.

Definitions

We start by giving some definitions concerning planning tasks and then formally introduce potential heuristics.

Planning Tasks and Heuristics

We consider SAS$^+$ planning tasks with operator costs (Bäckström and Nebel 1995). A planning task Π is a tuple $\langle \mathcal{V}, \mathcal{O}, s_{\text{I}}, s_\star, \mathit{cost} \rangle$ where $\mathcal{V}$ is a finite set of variables V,

each with an associated finite domain $dom(V)$. A *fact* $\langle V, v\rangle$ is a pair of a variable V and one of its values $v \in dom(V)$, and $\mathcal{F}$ is the set of all facts. A partial variable assignment p maps some variables $vars(p) \subseteq \mathcal{V}$ to values in their domains. We say a fact $\langle V, v\rangle$ is true in a variable assignment p if $V \in vars(p)$ and $p[V] = v$. We also consider p to be the set of facts that are true in p where this simplifies things. A *state* is a variable assignment over all variables. We denote the set of all states with $\mathcal{S}$. A partial variable assignment p is *consistent* with a state s if $p[V] = s[V]$ for all $V \in vars(p)$.

The finite set $\mathcal{O}$ contains *operators* o, each with a *precondition* $\text{pre}(o)$ and *effect* $\text{eff}(o)$ which are both partial variable assignments. An operator $o \in \mathcal{O}$ is applicable in state s if its precondition is consistent with s. The result of applying o in s is the state $s[\![o]\!]$ that maps every $V \in vars(\text{eff}(o))$ to $\text{eff}(o)[V]$ and all other variables to $s[V]$. A sequence of operators $\pi = \langle o_1, \ldots, o_n\rangle$ is applicable in a state s_0 if there is a state s_i for every $1 \leq i \leq n$ such that o_i is applicable in s_{i-1} and $s_{i-1}[\![o_i]\!] = s_i$. The resulting state is $s_0[\![\pi]\!] = s_n$. The *cost function* $cost : \mathcal{O} \to \mathbb{R}_0^+$ assigns a cost to each operator. The cost of an operator sequence $\pi = \langle o_1, \ldots, o_n\rangle$ is the sum of its operator costs $cost(\pi) = \sum_{i=1}^{n} cost(o_i)$.

The *goal description* $s_\star$ is a partial variable assignment, and we call states consistent with $s_\star$ *goal states*. An operator sequence π that is applicable in a state s, and results in a goal state $s[\![\pi]\!]$, is called an s-plan. If s is the *initial state* s_I, we refer to s_I-plans as *plans* or *solutions* of the planning task Π. We are interested in finding *optimal plans*, i.e., ones with minimal cost.

A *heuristic* $h : \mathcal{S} \to \mathbb{R} \cup \{\infty\}$ estimates the cost of optimal s-plans. The *optimal heuristic* $h^*(s)$ maps each state s to the exact cost of an optimal s-plan or ∞ if no s-plan exists. An *admissible* heuristic never overestimates the optimal solution cost, i.e., $h(s) \leq h^*(s)$ for all states $s \in \mathcal{S}$. The A* algorithm (Hart, Nilsson, and Raphael 1968) finds optimal solutions if its heuristic is admissible. We call a heuristic *goal-aware* if it maps goal states to values less than or equal to zero. We call it *consistent* if it satisfies the triangle inequality $h(s) \leq h(s[\![o]\!]) + cost(o)$ for all states s and operators o that are applicable in s. Goal-aware and consistent heuristics are also admissible, and admissible heuristics are goal-aware. This implies that when restricting attention to consistent heuristics, a heuristic is admissible iff it is goal-aware.

Potential Heuristics

Potential heuristics (Pommerening et al. 2015) are a family of heuristics that assign numerical values to each fact. We call this value the fact's *potential*. The heuristic value of a state is the sum of potentials for all facts that are true in the state. Potentials have to satisfy constraints that guarantee that the resulting heuristic is consistent and goal-aware. The constraints form a linear program (LP) that can be optimized for any linear combination of potentials.

Like Pommerening et al., we assume that if an operator has an effect on a variable, it also has a precondition on it ($vars(\text{eff}(o)) \subseteq vars(\text{pre}(o))$) and that every variable occurs in the goal description ($vars(s_\star) = \mathcal{V}$).[1] This simplifies the linear program as follows:

$$\text{Maximize } opt \text{ subject to}$$
$$\sum_{V \in \mathcal{V}} \mathsf{P}_{\langle V, s_\star[V]\rangle} \leq 0$$
$$\sum_{V \in vars(\text{eff}(o))} (\mathsf{P}_{\langle V, \text{pre}(o)[V]\rangle} - \mathsf{P}_{\langle V, \text{eff}(o)[V]\rangle}) \leq cost(o)$$
$$\text{for all } o \in \mathcal{O}$$

The first constraint guarantees that the goal state has a non-positive heuristic value (goal-awareness) while the second constraint guarantees that no operator violates consistency. Every heuristic that is consistent and goal-aware satisfies the constraints, so they are necessary and sufficient conditions that characterize the set of consistent and goal-aware potential heuristics exactly.

The objective function *opt* can be chosen arbitrarily, and we discuss different choices in the next section.

Optimization Functions

The heuristic value of any state s is a simple sum over potentials:

$$opt_s = \sum_{V \in \mathcal{V}} \mathsf{P}_{\langle V, s[V]\rangle}.$$

Maximizing opt_s subject to the potential function constraints yields a heuristic with maximal value for state s.

Initial State

Pommerening et al. (2015) used potential functions that optimize the heuristic value of the initial state in their experiments, i.e., they maximized opt_{s_I}.

One obvious disadvantage of maximizing the heuristic value of only one state is that there is no incentive to optimize potentials of facts that do not occur in the initial state. We therefore introduce alternative optimization functions that consider more than one state.

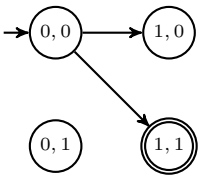
All Syntactic States

Instead of maximizing the heuristic value of one state, we can maximize the average heuristic value of *all* states. In general, the average heuristic value for any set of states S is a weighted sum over potentials:

$$opt_S = \frac{1}{|S|} \sum_{s \in S} \sum_{V \in \mathcal{V}} \mathsf{P}_{\langle V, s[V]\rangle}.$$

Note that we can generally eliminate any linear transformation of the objective function since we are not interested in the objective value itself. For example, it makes no difference if we optimize the average heuristic value, or the sum of heuristic values, in S.

[1] We make this assumption for simplicity of presentation only. In our experiments we use an implementation that can handle arbitrary tasks.



$$
\begin{aligned}
opt_{\mathcal{S}} &= \frac{1}{4}(h(s_{0,0}) + h(s_{1,0}) + \\
&\qquad h(s_{0,1}) + h(s_{1,1})) \\
&= \frac{1}{4}\left(2\mathsf{P}_{\langle X,0\rangle} + 2\mathsf{P}_{\langle X,1\rangle} + \right. \\
&\qquad \left. 2\mathsf{P}_{\langle Y,0\rangle} + 2\mathsf{P}_{\langle Y,1\rangle}\right) \\
&= \frac{1}{2}\left((\mathsf{P}_{\langle X,0\rangle} + \mathsf{P}_{\langle Y,0\rangle}) + \right. \\
&\qquad \left. (\mathsf{P}_{\langle X,1\rangle} + \mathsf{P}_{\langle Y,1\rangle})\right) \\
&= \frac{1}{2}\left(h(s_{0,0}) + h(s_{1,1})\right) < \infty
\end{aligned}
$$

Figure 1: Example for a task where the set of all syntactic states $\mathcal{S}$ contains dead ends, but the LP maximizing $opt_{\mathcal{S}}$ is bounded. There are two variables X and Y, each with the domain $\{0, 1\}$. A state $s_{x,y}$ assigns x to X and y to Y. Independently of the dead ends' reachability, the average heuristic value is bounded by finite heuristic values.

If we consider the set of all syntactic states $\mathcal{S}$, the coefficient of each potential equals the fraction of states in which the corresponding fact is true:

$$
\begin{aligned}
opt_{\mathcal{S}} &= \frac{1}{|\mathcal{S}|}\sum_{s\in\mathcal{S}}\sum_{V\in\mathcal{V}}\mathsf{P}_{\langle V,s[V]\rangle} = \sum_{V\in\mathcal{V}}\sum_{s\in\mathcal{S}}\frac{1}{|\mathcal{S}|}\mathsf{P}_{\langle V,s[V]\rangle} \\
&= \sum_{V\in\mathcal{V}}\sum_{v\in dom(V)}\frac{|\{s\in\mathcal{S}\mid s[V]=v\}|}{|\mathcal{S}|}\mathsf{P}_{\langle V,v\rangle} \\
&= \sum_{\langle V,v\rangle\in\mathcal{F}}\frac{1}{|dom(V)|}\mathsf{P}_{\langle V,v\rangle}
\end{aligned}
$$

Maximizing $opt_{\mathcal{S}}$ yields a potential heuristic with the highest possible average heuristic value, but there are two problems in practice.

First, if $\mathcal{S}$ contains dead ends, heuristic values can become arbitrarily large and the linear program can become unbounded. This can even happen if all dead ends are unreachable. When the linear program is unbounded, we usually cannot extract a useful heuristic function. Unfortunately, the LP is not always unbounded if $\mathcal{S}$ contains a dead end so we cannot use this to test for dead ends in a set of states. See Figure 1 for an example of a task where the LP is bounded even though there are dead ends.

Second, the heuristic values of unreachable states influence the solution. This is problematic since unreachable states are never encountered during the search. Thus it is pointless to optimize their heuristic value. Moreover, they can be fundamentally different from reachable states. For example, an invariant analysis can detect facts that can never occur together in a reachable state, i.e., they are *mutex*. The set of all states $\mathcal{S}$ also includes states that violate such mutex information, and maximizing $opt_{\mathcal{S}}$ requires considering them. We would like to only consider reachable states, but we cannot classify this set of states concisely. (If this could be done efficiently, it would also present an efficient test for plan existence as we could use it to check if the goal is reachable.) While we could exclude all states that violate a single mutex, excluding multiple (potentially overlapping) mutexes is more complicated and we therefore leave it as future work.

Handling Dead Ends

Dead-end states (those with $h^*(s) = \infty$) are problematic for the optimization function for several reasons. First, with an optimization function that sums or averages over heuristic values of many states, a dead end can outweigh the contribution of an arbitrarily large number of states with finite goal distance. Therefore, the optimization function offers little incentive to derive high heuristic values for non-dead-end states.

Second, dead ends that are "recognized" by the constraints can be assigned arbitrarily high finite heuristic values, which is likely to lead to an unbounded objective function.

Both issues can be finessed to a certain extent by introducing an upper bound M on potentials. We can extend the linear program with the constraints $\mathsf{P}_{\langle V,v\rangle} \leq M$ for every fact $\langle V, v\rangle \in \mathcal{F}$. With these additional constraints, the heuristic value of every state s is bounded by $\min(h^*(s), |\mathcal{V}| \cdot M)$. An LP that maximizes the average heuristic value for a set of states is therefore always bounded. If the bound is large ($|\mathcal{V}| \cdot M \gg h^*(s_{\mathrm{I}})$), potential heuristics can achieve a higher average value by assigning high values to recognized dead ends. Heuristic values larger than $h^*(s_{\mathrm{I}})$ thus contribute the most to the objective of a maximizing solution. States with such heuristic values are never expanded during the search.

Focusing on Reachable States

As mentioned before, ideally one would focus the optimization of the heuristic function on the set of all reachable states. This is usually infeasible, but we can use samples that approximate the set. Drawing a uniformly distributed sample from the reachable part of the state space is a hard task in itself. We adopt the sampling procedure from Haslum et al. (2007): we first calculate a heuristic value for the initial state using the potential heuristic optimized for this value. The heuristic can underestimate, so we multiply this estimate by 2 before we divide the result by the average operator cost to get a rough estimate of the optimal solution depth. We then collect samples using random walks with a binomially distributed length centered around the estimated solution depth. If a random walk gets stuck in a state without successors, it is restarted in the initial state. We then optimize the average heuristic value $opt_{\hat{S}}$ for a set $\hat{S}$ of sample states. The sampled states can still contain dead ends, so the linear program can become unbounded.

One way of handling dead-end samples is to bound the potential of each fact as described above.

If the number of samples is small enough, we can also explicitly check for each state whether it is a recognized dead end. For each sample $s \in \hat{S}$, we optimize the LP with the objective opt_s. The LP is unbounded iff s is a recognized dead end.

Minimizing Search Effort

A high heuristic value for a state s is useless if s is never generated during the search. Maximizing the average heuristic value of all reachable states also maximizes heuristic values of states that the search never considers. It also might be more important to have a heuristic value of 1 for n states than to have a heuristic value of n for 1 state.

Ideally, we would like to minimize the effort of an A* search with the resulting heuristic. Korf, Reid, and Edelkamp (2001) estimate the search effort of an IDA* search up to a bound of c using an admissible and consistent heuristic. Their estimate is based on the number N_i of nodes with a g value of i and the probability $P(k)$ that a node drawn uniformly at random from the search tree has a heuristic value less than or equal to k (the heuristic's *equilibrium distribution*). In the limit for large c the number of expanded nodes is

$$\sum_{k=0}^{c} N_{c-k} P(k).$$

The formula can also be used for best-first search if the probability $P(k)$ is based on nodes that are drawn uniformly at random from the the search space (not the search tree) with a cost up to c. If we use a set of samples $\hat{S}$ to estimate the value of $P(k)$ the formula for search effort can be rewritten as:

$$\begin{aligned}
\sum_{k=0}^{c} N_{c-k} \hat{P}(k) &= \sum_{k=0}^{c} N_{c-k} \frac{|\{s \in \hat{S} \mid h(s) \leq k\}|}{|\hat{S}|} \\
&= \frac{1}{|\hat{S}|} \sum_{k=0}^{c} N_{c-k} \sum_{s \in \hat{S}} [h(s) \leq k] \\
&= \frac{1}{|\hat{S}|} \sum_{s \in \hat{S}} \sum_{k=0}^{c} N_{c-k} [h(s) \leq k] \\
&= \frac{1}{|\hat{S}|} \sum_{s \in \hat{S}} \sum_{i=0}^{c-h(s)} N_i
\end{aligned}$$

In other words, the search effort is estimated to be the average cumulative distribution of g values at $c-h(s)$. We can estimate the distribution of N and the value of c in different ways resulting in different cumulative distribution functions.

For example, if we assume N_i to be constant for all i, then the estimated search effort is $\frac{1}{|\hat{S}|} \sum_{s \in \hat{S}} \sum_{i=0}^{c-h(s)} N_0 = \frac{1}{|\hat{S}|} \sum_{s \in \hat{S}} N_0(c - h(s) + 1)$, which is a linear transformation of the negative average heuristic value $-\frac{1}{|\hat{S}|} \sum_{s \in \hat{S}} h(s)$. Thus we implicitly assume N_i to be constant if we maximize the average heuristic value over some samples. This explains the result by Haslum et al. (2007), who show that the average heuristic value is not the best predictor for search effort.

A slightly more realistic assumption might be that N develops linearly ($N_i = ib$), which results in the search effort estimate $\frac{b}{2|\hat{S}|} \sum_{s \in \hat{S}} (c-h(s))(c-h(s)+1)$. This formula is quadratic in $h(s)$ so it can no longer be optimized with linear programming. Quadratic programming (QP) is the problem of optimizing a quadratic objective function ($\frac{1}{2} x^\top Q x + c^\top x$) subject to linear constraints. QP problems with a minimization objective can be solved in polynomial time if the matrix Q is positive semidefinite. If we remove constant factors and rewrite the objective as $\sum_{s \in \hat{S}} (d_s^2 + d_s)$ with the additional constraints $d_s = c - h(s)$, it is easy to see that this produces a positive semidefinite matrix Q which has entries of 2 on the diagonal and is 0 everywhere else.

A common assumption is that N develops exponentially ($N_i = b^i$) which has an estimated search effort of $\frac{1}{|\hat{S}|(b-1)} \sum_{s \in \hat{S}} (b^{c-h(s)+1} - 1)$. By removing constant factors, we end up with the objective $\sum_{s \in \hat{S}} b^{-h(s)}$. Since optimizing this objective requires more advanced optimization techniques, we leave it as future work.

Using Multiple Heuristics

The main advantage of potential heuristics is that they are very fast to evaluate during the search. Instead of computing one heuristic, we can easily compute multiple heuristics, evaluate them all during the search, and use their maximum.

Results by Pommerening et al. (2015) show that even maximizing over two potential heuristics that were optimized for the same objective can have a significant benefit. In general, LP solutions are not unique so there may be multiple potential functions optimizing the same objective. The resulting heuristics can be quite different on most states.

We expect even better results if the different component heuristics are very diverse and have strengths and weaknesses in different parts of the search space. The remaining question is how best to select diverse component heuristics.

For sample-based heuristics an obvious approach is to just compute many heuristics, each with its own set of independently drawn samples. In the extreme case, where each set of samples contains exactly one sample, we get one heuristic per sample which maximizes the heuristic value of this sample. On the other end of the scale, we can use a single heuristic for a larger set of samples. This is a trade-off between heuristic accuracy, precomputation time and evaluation speed during the search. Alternatively, we can look for diverse heuristics explicitly.

Automatic Diversification

Similar to some of the approaches above, our automatic diversification algorithm starts by sampling a fixed number of states. Instead of calculating a single potential heuristic for all of the samples however, we try to find a small ensemble of potential heuristics that together give the maximal heuristic value for each of the samples.

We can precompute the maximal value for each sample s by optimizing for s individually, i.e., computing h_s^{pot}. For brevity we say that a heuristic h *covers* a sample s if it gives the maximal value for s, i.e., $h(s) = h_s^{\text{pot}}(s)$. We iteratively try to find a heuristic that covers as many samples as possible and focus on the remaining samples in subsequent iterations.

Algorithm 1 describes the procedure in detail. First, we sample a fixed number of states $\hat{S}$ and initialize an empty set

Algorithm 1 Sample a fixed number of states and find a small ensemble of potential heuristics that together give the maximal heuristic value for each of the samples.

function AUTOMATICDIVERSIFICATION
  $\hat{S} \leftarrow$ SAMPLESTATES
  $H \leftarrow \emptyset$
  while $\hat{S} \neq \emptyset$ **do**
    if $\exists s \in \hat{S}$ with $h^{\mathrm{pot}}_{\hat{S}}(s) = h^{\mathrm{pot}}_{s}(s)$ **then**
      $h \leftarrow h^{\mathrm{pot}}_{\hat{S}}$
    else
      $s \leftarrow$ RANDOM($\hat{S}$)
      $h \leftarrow h^{\mathrm{pot}}_{s}$
    $H \leftarrow H \cup \{h\}$
    $\hat{S} \leftarrow \{s \in \hat{S} \mid h(s) < h^{\mathrm{pot}}_{s}(s)\}$
  return H

of potential heuristics H. Then we iteratively calculate the potential heuristic $h = h^{\mathrm{pot}}_{\hat{S}}$ maximizing the average heuristic value for all samples that are not yet covered. If this heuristic doesn't cover any additional samples, we choose a random sample s and set $h = h^{\mathrm{pot}}_{s}$. We add h to H and remove all samples from $\hat{S}$ that are now covered, before continuing with the next iteration. The algorithm stops once $\hat{S}$ becomes empty, i.e. once the heuristics in H together cover all samples.

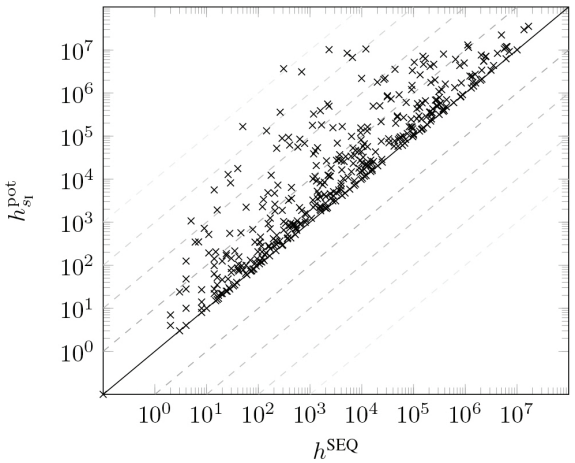


Figure 2: Number of expansions excluding the last f layer for the potential heuristic optimized for the initial state ($h^{\mathrm{pot}}_{s_\mathrm{I}}$) and the potential heuristic optimized for every state s encountered during the search individually (h^{SEQ}) on commonly solved tasks. There are no points below the diagonal since the latter heuristic is an upper bound for the former. Points above the diagonal represent tasks for which we can hope to improve upon $h^{\mathrm{pot}}_{s_\mathrm{I}}$.

Evaluation

We implemented our heuristics in the Fast Downward planning system (Helmert 2006) and evaluated them experimentally using benchmarks from the international planning competitions (IPC) 1998–2011. For solving linear programs, we used IBM CPLEX v12.5.1. Each task was run on a single core of Intel Xeon E5-2660 processors with 2.2 GHz. We limited runtime to 30 minutes and memory usage to 2 GB.

Initial State

We first investigate how much room for improvement there is for potential heuristics with any optimization function. As a baseline for this comparison we use the potential heuristic $h^{\mathrm{pot}}_{s_\mathrm{I}}$ which optimizes the heuristic value for the initial state, as presented by Pommerening et al. (2015). We compare this to the best heuristic that we can aim for, i.e., one that optimizes the potential function for the given state in every evaluation. As mentioned above this heuristic is equal to the state equation heuristic h^{SEQ}. To measure the relative heuristic quality of $h^{\mathrm{pot}}_{s_\mathrm{I}}$ and h^{SEQ}, we compare the number of expansions for commonly solved tasks in Figure 2. As usual, in this and all following plots, we exclude states expanded in the last f layer because their number depends on tie-breaking. We can see that for many tasks the number of expansions for h^{SEQ} is orders of magnitude lower than the one for $h^{\mathrm{pot}}_{s_\mathrm{I}}$, which shows that there is still room for improvement for other optimization functions.

In total, $h^{\mathrm{pot}}_{s_\mathrm{I}}$ solves 21 fewer tasks than h^{SEQ} (see Table 1). The faster evaluation time of this single potential heuristic cannot make up for its lower heuristic quality. This matches the result by Pommerening et al. (2015).

All Syntactic States

The third column in Table 1 shows the number of solved tasks when maximizing the average heuristic value of *all* states ($h^{\mathrm{pot}}_{\mathcal{S}}$). As mentioned above, we deal with dead ends in the state space by bounding the values of potentials. We use a limit of 10^8 for each potential, which is high enough for dead ends to have heuristic values different from regular states (the ParcPrinter domain has operator costs on the order of 10^6). The limit is also low enough to avoid numerical problems in the LP solver.

We see that overall $h^{\mathrm{pot}}_{\mathcal{S}}$ with bounded potentials solves significantly more tasks than $h^{\mathrm{pot}}_{s_\mathrm{I}}$ (659 vs. 611). It also has a higher coverage in 20 out of 44 domains, while the same is true in the opposite direction only in 4 domains.

Average of Sample States

Our next set of experiments evaluates the optimization function $h^{\mathrm{pot}}_{\hat{S}}$ that maximizes the average heuristic value of a set of samples $\hat{S}$. To distinguish between the version that bounds the potentials (*B*) and the version that filters dead ends (*F*), we denote the two variants by $h^{\mathrm{pot}}_{\hat{S},B}$ and $h^{\mathrm{pot}}_{\hat{S},F}$.

In order to find a good value for the number of samples we evaluated $h^{\mathrm{pot}}_{\hat{S},B}$ for sample sizes between 1 and 1 000 000. Table 2 shows that the number of solved tasks increases significantly when going from 1 sample to 500 samples. Be-

Coverage	h^{SEQ}	$h^{\text{pot}}_{s_1}$	$h^{\text{pot}}_{\mathcal{S}}$	$h^{\text{pot}}_{\hat{S},B}$	$h^{\text{pot}}_{\hat{S},F}$	$h^{\text{pot}}_{effort,F}$	$h^{\text{pot}}_{diverse}$
airport (50)	**23**	**23**	22	**23**	**23**	**23**	**23**
barman-11 (20)	4	4	4	4	4	4	**7**
blocks (35)	**28**	18	**28**	**28**	**28**	**28**	**28**
depot (22)	**7**	4	**7**	**7**	**7**	**7**	**7**
driverlog (20)	**12**	10	**12**	**12**	**12**	**12**	**12**
elevators-08 (30)	9	**11**	**11**	**11**	**11**	**11**	**11**
elevators-11 (20)	7	**9**	**9**	**9**	**9**	**9**	**9**
floortile-11 (20)	**4**	2	**4**	**4**	2	**4**	**4**
freecell (80)	40	39	39	53	53	41	**54**
grid (5)	1	1	**2**	**2**	1	1	**2**
gripper (20)	**7**	**7**	**7**	**7**	**7**	**7**	**7**
logistics-00 (28)	**16**	13	**16**	**16**	**16**	**16**	**16**
logistics-98 (35)	**4**	2	**4**	**4**	**4**	**4**	**4**
miconic (150)	**52**	**52**	51	**52**	**52**	51	**52**
mprime (35)	20	**23**	**23**	**23**	**23**	**23**	**23**
mystery (30)	15	16	**17**	**17**	16	**17**	16
nomystery-11 (20)	10	8	**12**	**12**	**12**	**12**	**12**
openstacks-06 (30)	**7**	**7**	**7**	**7**	**7**	**7**	**7**
openstacks-08 (30)	16	**20**	**20**	**20**	**20**	**20**	**20**
openstacks-11 (20)	11	**15**	**15**	**15**	**15**	**15**	**15**
parcprinter-08 (30)	**28**	22	22	25	13	12	27
parcprinter-11 (20)	**20**	15	16	**20**	9	8	**20**
parking-11 (20)	3	0	**7**	**7**	1	1	**7**
pathways (30)	**4**	**4**	**4**	**4**	**4**	**4**	**4**
pegsol-08 (30)	28	28	**29**	27	28	28	28
pegsol-11 (20)	18	18	**19**	17	18	18	18
pipesworld-nt (50)	15	21	21	**22**	21	**22**	20
pipesworld-t (50)	11	15	14	**16**	15	15	**16**
psr-small (50)	**50**	49	**50**	**50**	**50**	**50**	**50**
rovers (40)	**6**	5	**6**	**6**	**6**	**6**	**6**
satellite (36)	**6**	5	**6**	**6**	**6**	**6**	**6**
scanalyzer-08 (30)	**14**	13	13	12	12	12	13
scanalyzer-11 (20)	**11**	10	10	9	9	9	10
sokoban-08 (30)	19	22	**24**	**24**	23	23	23
sokoban-11 (20)	16	18	**19**	**19**	18	18	18
tidybot-11 (20)	7	**13**	**13**	**13**	**13**	11	**13**
tpp (30)	**8**	6	6	7	7	7	7
transport-08 (30)	**11**	**11**	**11**	**11**	**11**	**11**	**11**
transport-11 (20)	**6**	**6**	**6**	**6**	**6**	**6**	**6**
trucks (30)	9	9	9	**12**	9	7	**12**
visitall-11 (20)	**17**	**17**	16	13	13	13	**17**
wood-08 (30)	**14**	8	12	12	11	12	**14**
wood-11 (20)	**9**	3	7	7	6	7	**9**
zenotravel (20)	**9**	**9**	**9**	8	8	8	**9**
Sum (1396)	632	611	659	679	639	626	**693**

Table 1: Number of solved tasks for the state equation heuristic h^{SEQ} and different potential heuristics. We compare the heuristics obtained by maximizing the heuristic value of the initial state ($h^{\text{pot}}_{s_1}$), all syntactic states ($h^{\text{pot}}_{\mathcal{S}}$) and a set of 1000 samples $\hat{S}$ where $h^{\text{pot}}_{\hat{S},B}$ bounds potentials and $h^{\text{pot}}_{\hat{S},F}$ filters dead ends. The heuristic $h^{\text{pot}}_{effort,F}$ tries to minimize the search effort and $h^{\text{pot}}_{diverse}$ uses automatic diversification to find an ensemble of potential heuristics that achieve the maximal heuristic estimate for each of a set of sample states.

$\lvert\hat{S}\rvert$	**1**	**5**	**10**	**50**	**100**	**500**	**1K**	**5K**	**10K**	**50K**	**100K**	**500K**	**1M**
Coverage	567	630	643	659	672	679	679	679	**681**	679	680	676	673

Table 2: Number of tasks solved by the heuristic that maximizes the average heuristic estimates of a set of samples $\hat{S}$ while bounding the potentials ($h^{\text{pot}}_{\hat{S},B}$) for different sizes of $\hat{S}$.

tween 500 and 100 000 samples the number of solved tasks remains roughly the same. Only when using even more samples does coverage decrease. The time it takes to generate the sample states is negligible for small sample sizes, but can be very large for bigger sample sizes and more difficult tasks. We therefore use a default value of 1000 samples in the experiments below.

The results in Table 1 allow us to compare the $h^{\text{pot}}_{\hat{S},B}$ and $h^{\text{pot}}_{\hat{S},F}$. The total number of solved tasks is higher for the method that bounds the potentials (679) than for the variant that filters dead ends (639). Except for two tasks in the Peg-Solitaire domain the former method also solves as many or more tasks in each domain. We can explain the performance difference by the advantages that bounding the potentials has over filtering dead ends.

First, filtering dead ends takes time since we need to solve a linear program for each of the samples. This is fast enough to run within a minute in most cases, but it can be critical and even lead to timeouts in some domains or on large tasks.

Second, the sampling method often returns dead ends since the $h^{\text{pot}}_{s_1}$ heuristic that we use for setting the lengths of the random walks does not detect dead ends during the walk. For $h^{\text{pot}}_{\hat{S},F}$ this leads to the problem that very few samples remain for the optimization. Since the method that bounds the potentials does not need to filter dead ends, it can always consider all samples during the optimization.

Third, filtering dead ends lets the LP solver optimize for non-dead-end states only. Therefore, heuristic estimates for dead-end states are likely to be on the same order of magnitude as the estimates for non-dead-end states and the search will be likely to expand the dead-end states.

Given the results of this comparison, we will bound the potentials for all methods below, except where stated otherwise.

Search Effort

Next, we evaluate the effect of optimizing for minimal search effort. If we assume that the g layer sizes N_i grow linearly ($N_i = ib$ for some factor b) the resulting QP problems are still solvable in polynomial time. We used 1000 samples as input for this optimization and compare the obtained heuristic $h^{\text{pot}}_{effort,F}$ with $h^{\text{pot}}_{\hat{S},F}$ maximizing the average heuristic values of the same set of samples. For both variants we explicitly filter dead ends from the samples before optimizing since bounding the potentials as above led to numerical difficulties for the search effort estimation.

In Figure 3 we compare the number of expansions of the two heuristics. As we can see, there is no clear winner,

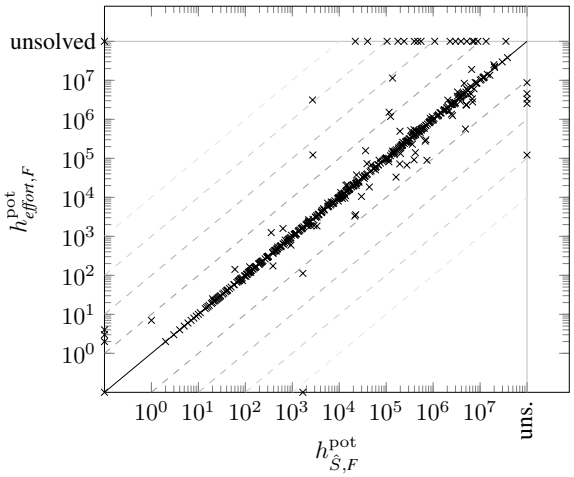


Figure 3: Number of expansions excluding the last f layer for the potential heuristic optimized for the average heuristic value of a set of samples $\hat{S}$ ($h^{\text{pot}}_{\hat{S},F}$) and the potential heuristic that tries to minimize search effort ($h^{\text{pot}}_{effort,F}$).

Heuristics	**1**	**5**	**10**	**50**	**100**	**500**	**1000**
Coverage	567	597	610	642	**647**	594	569

Table 3: Number of solved tasks by the combination of n h^{pot}_s heuristics for single randomly sampled states s for different values of n.

since both heuristics need the same number of expansions for most of the commonly solved tasks. Table 1 shows that $h^{\text{pot}}_{effort,F}$ solves more tasks in 5 domains and less tasks in 6 domains. The total coverage of $h^{\text{pot}}_{effort,F}$ is 13 tasks less than that of $h^{\text{pot}}_{\hat{S},F}$, mainly because $h^{\text{pot}}_{effort,F}$ solves 12 tasks less in the Freecell domain. Judging by these results, trying to minimize the search effort looks like a promising research direction, but bounding the potentials seems to be even more important for good performance. Since numerical difficulties prohibit us from doing so for h^{pot}_{effort}, we do not pursue this idea further in this work.

Maximum over Multiple Heuristics

A straightforward idea for obtaining multiple potential functions is to repeatedly use a randomized approach that yields different potential functions for each invocation. The heuristics $h^{\text{pot}}_{s_\text{I}}$ and $h^{\text{pot}}_{\mathcal{S}}$ are not randomized. They will therefore always return the same heuristic unless the underlying LP solver uses different strategies for solving the optimization problem. There are techniques to bias the LP solver to return different solutions, but we treat the solver as a black box here.

Instead, we calculate h^{pot}_s for each state s in a set of random samples $\hat{S}$ and use all obtained heuristics together. In Table 3 we can see that the number of solved tasks increases from 567 to 647 when going from 1 to 100 heuristics. Only when using even more heuristics does coverage decrease again. Overall, none of these configurations solves as many tasks as the single potential heuristic $h^{\text{pot}}_{\hat{S},B}$ optimized for a set of 1000 samples (679 solved tasks).

When comparing the total number of solved tasks by $h^{\text{pot}}_{s_\text{I}}$ (611) and the heuristic optimized for a single sample (567), we can conclude that it is better to use the initial state than a random state. This is expected since the initial state will have to be expanded in any case by the search, whereas a random state might never be visited.

The obvious next step is to create multiple heuristics for bigger sample sizes. Since our sampling method is randomized, the set of states each heuristic optimizes for is always different. As above, we use a fixed sample size of 1000 and create 1 to 1000 heuristics. Table 4 shows that the total number of solved tasks does not change much (678–681) for 1 to 100 potential heuristics optimized for a fixed number of samples. Only when using 500 or more heuristics does coverage decrease significantly. We hypothesize that using multiple $h^{\text{pot}}_{\hat{S},B}$ heuristics has no real advantage over using a single one because the heuristics are too similar.

Heuristics	**1**	**5**	**10**	**50**	**100**	**500**	**1000**
Coverage	679	**681**	**681**	679	678	623	584

Table 4: Number of solved tasks by the combination of n $h^{\text{pot}}_{\hat{S},B}$ heuristics optimized for randomly sampled state sets $\hat{S}$ ($|\hat{S}| = 1000$) for different values of n.

Sufficient # Heuristics	**1**	**5**	**10**	**50**	**100**	**500**	**1000**
# Tasks	852	140	29	88	42	156	89

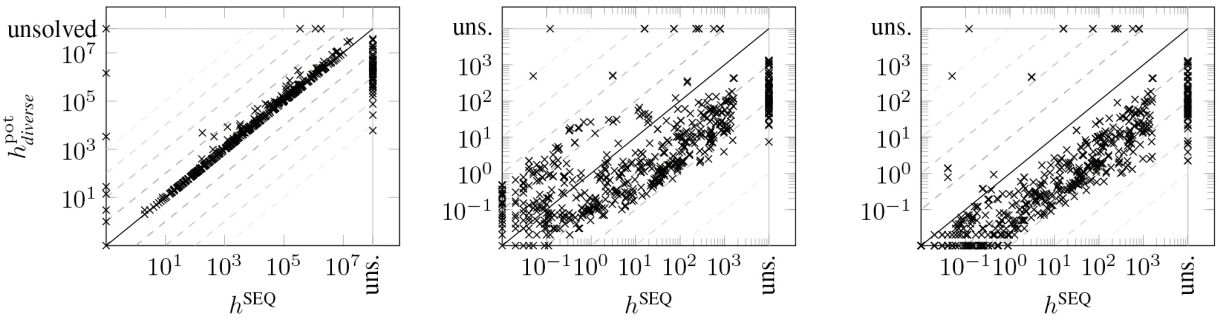
Table 5: Number of heuristics that are sufficient for achieving the maximal heuristic value for 1000 samples with automatic diversification $h^{\text{pot}}_{diverse}$.

Automatic Diversification

Our automatic diversification algorithm tries to address this problem by iteratively calculating potential functions that together achieve the maximal heuristic values for all samples. As a result, we no longer have to manually specify the number of potential heuristics manually. In our experiment to evaluate the algorithm, we use 1000 samples for comparability with the other methods.

Table 5 shows the number of potential heuristics that automatic diversification needs to achieve the maximal heuristic estimate for each of the 1000 samples. Interestingly, for many tasks a single potential heuristic is sufficient. There are, however, tasks for which we need to store a separate potential heuristic for almost all samples.

The experiments above show that it is beneficial to limit the number of potential heuristics used during search. We



(a) Number of expansions. (b) Runtimes. (c) Runtimes without preprocessing time.

Figure 4: Comparison of the state equation heuristic h^{SEQ} and the automatic diversification heuristic $h^{\text{pot}}_{diverse}$. We compare the number of expansions excluding the last f layer and the runtimes with and without the preprocessing time that $h^{\text{pot}}_{diverse}$ needs to find the heuristic ensemble. In Figure 4a points above the diagonal represent tasks for which $h^{\text{pot}}_{diverse}$ needs more expansions than h^{SEQ}. There are no points below the diagonal since h^{SEQ} is an upper bound on $h^{\text{pot}}_{diverse}$. In Figure 4b points below the diagonal correspond to tasks for which $h^{\text{pot}}_{diverse}$ finds a solution faster than h^{SEQ}. The same holds for Figure 4c, but here we exclude $h^{\text{pot}}_{diverse}$'s preprocessing time.

do the same here and stop the automatic diversification once it has found 100 heuristics. Table 1 (rightmost column) tells us that $h^{\text{pot}}_{diverse}$ solves more tasks (693) than all previously evaluated methods. On many domains $h^{\text{pot}}_{diverse}$ is even competitive with the state-of-the-art $h^{\text{LM-cut}}$ heuristic (Helmert and Domshlak 2009). While $h^{\text{LM-cut}}$ solves more tasks than $h^{\text{pot}}_{diverse}$ in 23 domains, the opposite is true in 11 domains.

In Figure **??** we compare the number of expansions made by $h^{\text{pot}}_{diverse}$ and h^{SEQ}. Since h^{SEQ} is an upper bound on $h^{\text{pot}}_{diverse}$ no task solved by both methods can be below the diagonal. Not only does $h^{\text{pot}}_{diverse}$ require the same number of expansions for almost all commonly solved tasks, it also solves many tasks for which h^{SEQ} fails to find a solution. When we compare the runtimes of the two methods in Figure **??** we see that there are some tasks for which h^{SEQ} finds a solution faster than $h^{\text{pot}}_{diverse}$, but this is mainly due to the preprocessing time that automatic diversification needs for calculating the ensemble of heuristics (see Figure **??**). For almost all tasks that are not solved by h^{SEQ} in 20 seconds, $h^{\text{pot}}_{diverse}$ finds solutions much faster, often by two orders of magnitude. This explains why $h^{\text{pot}}_{diverse}$ solves significantly more tasks than h^{SEQ}.

Conclusion

We have seen that potential heuristics are a flexible class of admissible and consistent heuristics. Multiple interesting objectives can be specified as linear combinations over fact potentials, and a heuristic maximizing them can be extracted in polynomial time.

We formulated and evaluated several optimization functions, and we found that it is beneficial to consider multiple states instead of only focusing on a single state such as the initial state. For example, we tried to maximize the average heuristic value of all syntactic states. However, the drawback is that the values for unreachable states and dead ends influence the resulting heuristic too much. Using samples to approximate the reachable state space, and maximizing their average heuristic estimate, led to better results. Dead ends can be explicitly removed from these samples, but we found that bounding the fact potentials yields better heuristics. This is because such heuristics have high estimates for dead ends and avoid visiting them during the search.

Trying to minimize the search effort is another promising idea. However, our experiments show no real advantage of a model of search spaces with a linear number of states for each cost layer over a model with a constant number. Numerical problems make the evaluation complicated, and more advanced models of the search space require linear optimization with quadratic or exponential objective functions. We did not explore using such objectives.

Since potential heuristics can be evaluated extremely fast during the search, we can easily compute multiple heuristics and maximize over them. We get the best results when the component heuristics have complementary strengths. With automatic diversification, we can find a small set of potential heuristics that are sufficient to give maximal estimates for each of a set of samples. The heuristic values very closely approximate the state equation heuristic, which is an upper bound on any potential heuristic. However, our automatic diversification heuristic is much faster to compute, and consequently, it solves significantly more tasks.

Nonetheless, looking for other objective functions or other combinations of potential heuristics is still promising in our opinion. The resulting heuristics will not reduce the number of expanded states by much, compared to the result of our automatic diversification approach. However, if they use fewer component heuristics, they might be faster to evaluate. One way of accomplishing this might be to search for more accurate search space models and formulate optimization functions that try to minimize the search effort. Another promising research direction is the study of potential heuris-

tics that assign potential not just to facts, but also to other formulas over the state variables.

Acknowledgments

This work was supported by the Swiss National Science Foundation (SNSF) as part of the project "Abstraction Heuristics for Planning and Combinatorial Search" (AHPACS).

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS^+ planning. *Computational Intelligence* 11(4):625–655.

Bonet, B. 2013. An admissible heuristic for SAS^+ planning obtained from the state equation. In Rossi, F., ed., *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 2268–2274.

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

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007)*, 1007–1012. AAAI Press.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169. AAAI Press.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Korf, R. E.; Reid, M.; and Edelkamp, S. 2001. Time complexity of iterative-deepening A^*. *Artificial Intelligence* 129:199–218.

Pommerening, F.; Helmert, M.; Röger, G.; and Seipp, J. 2015. From non-negative to general operator cost partitioning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3335–3341. AAAI Press.

van den Briel, M.; Benton, J.; Kambhampati, S.; and Vossen, T. 2007. An LP-based heuristic for optimal planning. In Bessiere, C., ed., *Proceedings of the Thirteenth International Conference on Principles and Practice of Constraint Programming (CP 2007)*, volume 4741 of *Lecture Notes in Computer Science*, 651–665. Springer-Verlag.