Subset Selection of Search Heuristics

Chris Rayner
University of Alberta
Edmonton, AB, Canada T6G 2E8
rayner@cs.ualberta.ca

Nathan Sturtevant
University of Denver
Denver, CO, USA 80208
sturtevant@cs.du.edu

Michael Bowling
University of Alberta
Edmonton, AB, Canada T6G 2E8
bowling@cs.ualberta.ca

Abstract

Constructing a strong heuristic function is a central problem in heuristic search. A common approach is to combine a number of heuristics by maximizing over the values from each. If a limit is placed on this number, then a subset selection problem arises. We treat this as an optimization problem, and proceed by translating a natural loss function into a submodular and monotonic utility function under which greedy selection is guaranteed to be near-optimal. We then extend this approach with a sampling scheme that retains provable optimality. Our empirical results show large improvements over existing methods, and give new insight into building heuristics for directed domains.

1 Introduction

The heuristic function – which we also simply call the *heuristic* – is an estimate of the cost or distance between two states. When used in search, a strong heuristic can improve solution quality, speed, and can render intractable problems tractable.

The literature describes several approaches to heuristic construction, including pattern databases [Culberson and Schaeffer, 1998], memory-based heuristics [Sturtevant *et al.*, 2009], regressors [Ernandes and Gori, 2004], and metric embeddings [Rayner *et al.*, 2011], each capable of generating multiple different heuristic functions based on input parameters. When multiple heuristics are available, it is common to query each and somehow combine the resulting values into a better estimate. However, under constraints on lookup time or memory, it may be that only a subset of a much larger pool of candidate heuristics can be used during search. Groups of heuristics can interact with each other in subtle ways, which makes selecting the best subset among them challenging.

In this paper, we formulate selecting heuristic subsets as an optimization problem. The loss function we aim to minimize is inspired by search effort, and is independent of the type of heuristics in the candidate set. When the candidate heuristics are admissible, this loss can be translated into a submodular and monotonic utility function; these properties imply that greedy selection is near-optimal. We also introduce a *sample* utility function under which greedy selection retains provable optimality if the heuristics are consistent. An empirical evaluation of our approach shows it to be capable of outperforming existing methods. We further use our approach to accurately benchmark a promising new type of true-distance heuristic, which gives valuable insight into the problem of constructing heuristics for highly directed search graphs.

2 Background

Research spanning the past two decades shows a progression toward building heuristics automatically, often in ways that enforce the key properties of admissibility and consistency. When heuristics are so readily constructed, an accompanying subset selection problem is inevitable.

One common motivation for heuristic construction is to solve otherwise intractable search problems. Here, pattern databases (PDBs) have arguably made the greatest impact [Culberson and Schaeffer, 1998]. The problem of selecting heuristic subsets is prevalent here, with it having been observed that many small PDBs can be more effective together than a single monolithic PDB [Holte *et al.*, 2006].

In other cases, the heuristic function is constructed to expedite the solution of arbitrarily many future problems. These cases are a motivating force behind our work, with example applications including GPS navigation and video game pathfinding. Several algorithms have been proposed in this context for selecting heuristic subsets, but most lack an optimality criterion or are tied to a specific type of heuristic function. For example, one recent approach draws a connection between heuristic construction and manifold learning [Weinberger *et al.*, 2005] and represents heuristic information as distances between points in Euclidean space [Rayner *et al.*, 2011]. Principal components analysis can be used to select an optimal, variance-preserving subset of this distance information, but that approach is exclusive to Euclidean heuristics.

Another popular approach is to precompute true distances to a landmark [Goldberg and Werneck, 2003], which can be thought of as selecting a subset of all distance information [Sturtevant *et al.*, 2009]. These methods can be viewed as special cases of Lipschitz embeddings [Bourgain, 1985] in which distances are computed to the nearest of a *set* of landmarks. Many selection algorithms have been devised for these, both as heuristics [Goldberg and Harrelson, 2005; Fuchs, 2010] and as metric embeddings [Linial *et al.*, 1995], but these too cannot be applied to other types of heuristics.

3 Subset Selection of Search Heuristics

We consider the problem of choosing a good subset H of a set of candidate heuristics $C = \{h_1, \ldots, h_{|C|}\}$. We assume the heuristics in H are to be combined with a set of default heuristics D by maximizing over the values across both H and D. For states i and j, we denote this heuristic lookup as:

$$h^H(i,j) = \max_{h_x \in H \cup D} h_x(i,j) \tag{1}$$

In the simplest case, D contains only the zero heuristic, which gives 0 for any pair of states queried. We further assume any default or candidate heuristic $h_x \in D \cup C$ is non-negative and admissible (i.e., never overestimating),

$$\forall i,j, \quad 0 \leq h_x(i,j) \leq \delta(i,j), \tag{2}$$

where $\delta(i,j)$ is the true distance between states i and j.

3.1 Optimization Problem

We formalize the heuristic subset selection problem as an optimization problem:

$$\begin{aligned} \textbf{minimize} \quad & \mathcal{L}(H) \\ \textbf{subject to} \quad & |H| = d \end{aligned} \tag{3}$$

The constraint $|H| = d$ simply limits the capacity of H to a fixed-size subset of C, and the loss $\mathcal{L}(H)$ is a scalar quantity summarizing the quality of a given subset H.

We relate loss to eventual search effort. An optimal search algorithm must expand any state encountered whose heuristic is low enough to suggest it may be on an optimal path to the goal [Bagchi and Mahanti, 1983], so a well-suited loss is the weighted sum of the errors between the resulting heuristic values and the true distances, for all pairs across n states:

$$\mathcal{L}(H) = \textstyle\sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} \left| \delta(i,j) - h^H(i,j) \right| \tag{4}$$

The non-negative weight matrix $W \in \mathbb{R}^{n \times n}$ is a free parameter, and it can be flexibly defined to specify the relative importance of each pair of states, perhaps based on knowledge of frequent start and goal locations.

We can rewrite this loss as an equivalent *utility* function $\mathcal{U}$. First, all of the heuristics in $H \cup D$ are admissible, so for all states i and j, $h^H(i,j) \leq \delta(i,j)$. Therefore we can remove the absolute value from line 4 and split the sum:[1]

$$\mathcal{L}(H) = \textstyle\sum_{i,j} W_{ij} \left(\delta(i,j) - h^H(i,j) \right) \tag{5}$$

$$= \textstyle\sum_{i,j} W_{ij}\ \delta(i,j) - \sum_{i,j} W_{ij}\, h^H(i,j) \tag{6}$$

The leftmost term on line 6 does not depend on H, so minimizing $\mathcal{L}(H)$ is equivalent to maximizing the term on the right. We introduce the corresponding utility function,

$$\mathcal{U}(H) = \textstyle\sum_{i,j} W_{ij}\, h^H(i,j) - \alpha, \tag{7}$$

where α is a normalizing constant that has no effect on the choice of H, but ensures that $\mathcal{U}(\emptyset) = 0$. α is defined as a weighted sum of the contributions of the default heuristics:

$$\alpha = \sum_{i,j} W_{ij}\, h^{\emptyset}(i,j) = \sum_{i,j} W_{ij} \max_{h_d \in D} h_d(i,j) \tag{8}$$

[1] Note i and j always iterate from 1 to n as on line 4.

All told, we arrive at a specific utility maximization problem:

$$\begin{aligned} \textbf{maximize} \quad & \mathcal{U}(H) \\ \textbf{subject to} \quad & |H| = d \end{aligned} \tag{9}$$

Unfortunately, there is unlikely to be an efficient algorithm to find a globally optimal solution to this problem.

Proposition 1 *The optimization problem* (9) *is NP-hard.*[2]

Proof. We sketch a reduction from the NP-complete *Vertex Cover* problem over an undirected graph (V, E). This is the problem of finding a subset of d graph vertices $T \subseteq V$ such that all edges in E are incident to at least one vertex in T.

By definition, heuristics describe values between pairs of vertices in a search graph. A special case of this is a function that only returns 1 between a vertex and its neighbors, and 0 for any other query. Thus, to reduce vertex cover, we define C as a set of heuristics $C = \{h_v : v \in V\}$ where each $h_v \in C$ gives a value of 1 between vertex v and its neighbors, and 0 otherwise. If a subset of d such heuristics captures all edge costs, then there is a vertex cover of size d as well. □

3.2 Approximation Algorithm

Despite Proposition 1, greedy selection will yield a solution to the optimization problem (9) with a strong near-optimality guarantee. This is because $\mathcal{U}$ is submodular and monotonic.

Submodularity is a diminishing returns property. It aptly describes settings where marginal gains in utility start to diminish due to saturation of the objective, such as with sensor placement and monetary gain. Let $A \subseteq B \subsetneq S$, let $x \in S \setminus B$, and let ϕ be a function over 2^S. ϕ is submodular if:

$$\phi(A \cup \{x\}) - \phi(A) \geq \phi(B \cup \{x\}) - \phi(B) \tag{10}$$

That is, the same element newly added to a subset and its superset will lead the subset to gain at least as much in value as the superset. Intuitively, the utility function $\mathcal{U}(H)$ and submodularity are a good fit, since adding a new heuristic to H will not newly cover any of the heuristic values that H already covers. We prove this formally in the following lemma.

Lemma 1 *$\mathcal{U}$ is submodular.*

Proof. Let A and B be sets of heuristics with $A \subseteq B$, and let $h_c \in C$ be a particular but arbitrary candidate heuristic function which is in neither A nor B (i.e., $h_c \in C \setminus B$). We can reproduce the inequality on line 10 as follows:

$$\mathcal{U}(A \cup \{h_c\}) - \mathcal{U}(A) \tag{11}$$

$$= \textstyle\sum_{i,j} W_{ij}\, h^{A \cup \{h_c\}}(i,j) - \sum_{i,j} W_{ij}\, h^A(i,j) \tag{12}$$

$$= \textstyle\sum_{i,j} W_{ij} \left(h^{A \cup \{h_c\}}(i,j) - h^A(i,j) \right) \tag{13}$$

$$= \textstyle\sum_{i,j} W_{ij} (h_c(i,j) - h^A(i,j))^+ \tag{14}$$

$$\geq \textstyle\sum_{i,j} W_{ij} (h_c(i,j) - h^{A \cup B}(i,j))^+ \tag{15}$$

$$= \mathcal{U}(B \cup \{h_c\}) - \mathcal{U}(B) \tag{16}$$

Line 12 twice expands the definition of utility (7) with the α terms cancelling, and line 13 rewrites this difference of sums

[2] A distinct result is the NP-hardness of ALT's preprocessing phase under an edge-covering objective [Bauer *et al.*, 2010].

as a sum of differences between h_c and h^A together versus h^A alone. Line 14 equates this to a sum of the *positive* differences between h_c and h^A, where $(x)^+ = \max\{0, x\}$. Line 15 holds since h_c's individual gains over $h^{A\cup B}$ cannot exceed its gains over h^A. But $A \cup B = B$, proving submodularity. □

Monotonicity is the notion that adding an element to a set never leads to a decrease in value. Let $A \subseteq B \subseteq S$ be sets and let ϕ be a function over 2^S. ϕ is monotonic if $\phi(A) \leq \phi(B)$. Since $\mathcal{U}(H)$ only measures the sum of the heuristics – and not, for example, the cost per heuristic lookup or the memory consumed – no heuristic added to H can decrease utility.

Lemma 2 *$\mathcal{U}$ is monotonic.*

Proof. Let A and B be sets of heuristics with $A \subseteq B$. We must show that $\mathcal{U}(A) \leq \mathcal{U}(B)$.

$$\mathcal{U}(A) = \textstyle\sum_{i,j} W_{ij}\, h^A(i,j) - \alpha \tag{17}$$

$$\leq \textstyle\sum_{i,j} W_{ij}\, h^{A\cup B}(i,j) - \alpha \tag{18}$$

$$= \textstyle\sum_{i,j} W_{ij}\, h^B(i,j) - \alpha = \mathcal{U}(B) \tag{19}$$

Where line 18 assumes non-negativity of the heuristics and the entries in W; thus proving monotonicity. □

Together, submodularity and monotonicity are exploitable properties that reveal simple approximation algorithms to hard problems. A reference including details for speeding up greedy selection under such functions is by Krause and Golovin [2012]. In particular, Lemmas 1, 2, and a key result by Nemhauser *et al.* [1978] lead us to the following result:

Theorem 1 *Initialize $H_0 = \emptyset$ and incrementally add heuristics by greedy selection from a set of admissible heuristics C,*

$$H_t = H_{t-1} \cup \left\{ \arg\max_{h \in C} \mathcal{U}(H_{t-1} \cup \{h\}) \right\}. \tag{20}$$

$\mathcal{U}(H_d)$ is greater than a factor of $(1-1/e) \approx 0.63$ of optimal.

A similar bound has been observed alongside the problem of selecting heuristics [Fuchs, 2010]. However, it applies to an edge covering measure of utility that can only be used with a specific type of heuristic, and does not incorporate an arbitrary default heuristic. While we compare to such an edge covering objective later, we stress that Theorem 1 applies to any – possibly heterogeneous – set of candidate heuristics C for which our measure of utility can be efficiently determined.

4 Sample Utility

Greedy selection can be further sped up by measuring utility with a simpler approximation to $\mathcal{U}$. The approach we consider is to sum the heuristic values between only a sample of the states. Intuitively, if this sample is well distributed, then the heuristics between sample states should be strongly correlated with the heuristics between all states.

4.1 A Partitioning Approach

One of the many possible ways to implement sampling is to partition the state space into m mutually exclusive and collectively exhaustive regions, represented by sets of state indices, $Z_1, \ldots, Z_m$. Within each region, a single sample state is nominated, $z_i \in Z_i$. From these we define *sample utility* as

$$\overline{\mathcal{U}}(H) = \textstyle\sum_{p=1}^m \sum_{q=1}^m \overline{W}_{pq}\, h^H(z_p, z_q) - \overline{\alpha}, \tag{21}$$

where the weight between sample states p and q is the sum of the weights between the states in partitions Z_p and Z_q,

$$\overline{W}_{pq} = \textstyle\sum_{r \in Z_p} \sum_{s \in Z_q} W_{rs}, \tag{22}$$

or $\overline{W}_{pq} = |Z_p|\,|Z_q|$ if W specifies a uniform weighting, and

$$\overline{\alpha} = \textstyle\sum_{p=1}^m \sum_{q=1}^m \overline{W}_{pq}\, h^\emptyset(z_p, z_q) \tag{23}$$

is a normalizing constant ensuring $\overline{\mathcal{U}}(\emptyset) = 0$.

Choosing the best partitioning is an optimization problem unto itself, and a study of different ways to do so is left to future work. When we use sampling in our experiments, we incrementally select sample states to *cover* the largest number of uncovered states; a state is covered if it is within t steps of a sample. When all states are covered, we define partitions by assigning states to the nearest sample state as measured in an undirected version of the search graph, where the costs between states are replaced with $\delta(i,j) \leftarrow \min\{\delta(i,j), \delta(j,i)\}$. We note the similarity of this approach to specifying canonical heuristics [Sturtevant *et al.*, 2009] and choosing pathfinding subgoals [Bulitko *et al.*, 2012].

4.2 Optimality Analysis

In this section we analyze the effect sampling has on optimality with respect to the true utility function (7). The foundation of this analysis is heuristic consistency. If each default and candidate heuristic $h_x \in D \cup C$ is consistent,[3] i.e.,

$$\forall i, j, k, \quad h_x(i,k) \leq h_x(j,k) + \delta(i,j), \tag{24}$$

then we can use local distance information in the search graph to bound the difference between $\mathcal{U}(H)$ and $\overline{\mathcal{U}}(H)$ for any H.

Lemma 3 *The sample utility's error is bounded by the weighted sum of the distances between samples and states in the same partition, i.e., $\forall H \in 2^C$, $|\mathcal{U}(H) - \overline{\mathcal{U}}(H)| \leq \epsilon$ with:*

$$\epsilon = 2 \sum_{p=1}^m \sum_{q=1}^m \sum_{r \in Z_p} \sum_{s \in Z_q} W_{rs}(\delta(r, z_p) + \delta(z_q, s)) \tag{25}$$

Proof. Since the partitions are mutually exclusive and collectively contain all states, we can write $\mathcal{U}(H)$ equivalently as a sum over pairs of states between pairs of partitions:[4]

$$\mathcal{U}(H) = \textstyle\sum_{i,j} W_{ij}\, h^H(i,j) - \alpha \tag{26}$$

$$= \textstyle\sum_{p,q,r,s} W_{rs}\, h^H(r,s) - \sum_{p,q,r,s} W_{rs}\, h^\emptyset(r,s) \tag{27}$$

Since the heuristics in H are consistent, it must be true for arbitrary indices p, q, r, and s that:

$$h^H(r,s) \leq h^H(z_p, s) + \delta(r, z_p) \tag{28}$$

$$\leq h^H(z_p, z_q) + \delta(r, z_p) + \delta(z_q, s) \tag{29}$$

Reversing the consistency inequality, we similarly have:

$$h^\emptyset(r,s) \geq h^\emptyset(z_p, s) - \delta(z_p, r) \tag{30}$$

$$\geq h^\emptyset(z_p, z_q) - \delta(z_p, r) - \delta(s, z_q) \tag{31}$$

[3] Consistency also implies admissibility.

[4] Note p and q always iterate from 1 to m, and r and s always iterate over the state indices in Z_p and Z_q as on line 25.

Substituting lines 29 and 31 into line 27 establishes an upper bound on $\mathcal{U}(H)$ as follows:

$$\mathcal{U}(H) \quad (32)$$

$$\leq \sum_{p,q,r,s} W_{rs}(h^H(z_p, z_q) + \delta(r, z_p) + \delta(z_q, s)) - \sum_{p,q,r,s} W_{rs}(h^{\emptyset}(z_p, z_q) - \delta(z_p, r) - \delta(s, z_q)) \quad (33)$$

By separating the terms that refer to the heuristics from those that do not refer to the heuristics, we can rewrite line 33 and recover the definitions of $\overline{W}$, $\overline{\mathcal{U}}$ and ϵ:

$$\sum_{p,q,r,s} W_{rs}\, h^H(z_p, z_q) - \sum_{p,q,r,s} W_{rs}\, h^{\emptyset}(z_p, z_q) + \sum_{p,q,r,s} W_{rs}(\delta(r, z_p) + \delta(z_q, s)) + \sum_{p,q,r,s} W_{rs}(\delta(z_p, r) + \delta(s, z_q)) \quad (34)$$

$$= \sum_{p,q} \overline{W}_{pq}\, h^H(z_p, z_q) - \sum_{p,q} \overline{W}_{pq}\, h^{\emptyset}(z_p, z_q) + \epsilon \quad (35)$$

$$= \overline{\mathcal{U}}(H) + \epsilon \quad (36)$$

where the last two terms of line 34 are identical but with transposed indices, together equating to ϵ; thus proving $\mathcal{U}(H) \leq \overline{\mathcal{U}}(H) + \epsilon$. The lower bound proceeds similarly, and together these bounds give $|\mathcal{U}(H) - \overline{\mathcal{U}}(H)| \leq \epsilon$. □

Lemma 3 suggests we could use a *modified* greedy algorithm to optimize $\overline{\mathcal{U}}$ and still retain an approximation guarantee to optimal under $\mathcal{U}$ [Krause and Guestrin, 2005]. However, this modification entails a polynomial increase in the number of iterations over the candidate set, which may be unacceptable when the set is large. Fortunately, solutions of *known* optimality under $\overline{\mathcal{U}}$ are *provably* optimal under $\mathcal{U}$.

By duplicating the reasoning in Lemmas 1 and 2, we assert that the sample utility function $\overline{\mathcal{U}}$ is both submodular and monotonic. It follows as in Theorem 1 that greedy maximization under $\overline{\mathcal{U}}$ will find a solution that scores greater than a factor of $(1-1/e)$ of $\overline{\mathcal{U}}$'s optimal value. Together with Lemma 4 (described in Appendix A) we obtain the following result:

Theorem 2 *Initialize $H_0 = \emptyset$, and incrementally add heuristics by greedy selection from a set of consistent heuristics C,*

$$H_t = H_{t-1} \cup \left\{ \arg\max_{h \in C} \overline{\mathcal{U}}(H_{t-1} \cup \{h\}) \right\}. \quad (37)$$

$\overline{\mathcal{U}}(H_d)$ is within a factor of $(1 - 1/e)$ of optimal, implying $\mathcal{U}(H_d)$ is within a factor of $(1-1/e)\frac{\overline{\mathcal{U}}(H_d)-\epsilon}{\overline{\mathcal{U}}(H_d)+\epsilon-\epsilon/e}$ of optimal.

Under the assumption that both $\overline{\mathcal{U}}(H)$ and ϵ can be efficiently computed, it is therefore easy to determine an optimality bound on H *post factum*. Note that this bound improves as $\overline{\mathcal{U}}(H)$ increases (i.e., as we add more heuristics to the set H).

5 Experiments

We test our approach of greedy utility maximization under $\mathcal{U}$ and $\overline{\mathcal{U}}$ on optimal search with A^* [Hart *et al.*, 1968]. We focus on selecting true-distance heuristics, using the number of nodes expanded during search as a general performance measure. First, we compare our approach to existing methods on two undirected search domains. Next, we use our approach to accurately benchmark a new type of heuristic for *directed* domains, showing encouraging results over alternative heuristics. In each case our approach requires less than three hours to complete, even on the largest search graphs we consider.

5.1 Undirected Search Graphs

We first apply our approach to the selection of differential heuristics, or DHs [Sturtevant *et al.*, 2009], for undirected search graphs. A single DH consists of the precomputed distances from each state to a specific landmark state p and, from these distances, $h_p(i, j) = |\delta(i, p) - \delta(j, p)|$ gives consistent heuristics. A search graph with n states defines n candidate DHs. Since every DH occupies the same amount of memory, a cardinality constraint doubles as a constraint on memory.

Word Search

In these search problems, states are four letter words from an English dictionary. A word can be changed into another by substituting one letter at a time, resulting in $54,752$ edges across $4,820$ states. To turn *coal* into *gold*, for instance, one could take the path ⟨coal; goal; goad; gold⟩. Euclidean heuristics were recently shown to outperform DHs that used the *Farthest* selection algorithm [Goldberg and Harrelson, 2005] on precisely this search graph [Rayner *et al.*, 2011]. However, it was unclear whether this was a failure of DHs, or of the particular algorithm that was used to select them. This distinction is to be emphasized: a strong class of heuristics may be undervalued in the absence of a good selection algorithm.

This motivates testing whether DHs are powerful enough to outperform Euclidean heuristics when driven by our approach of greedy utility maximization under $\mathcal{U}$ (note we do not sample utility with $\overline{\mathcal{U}}$ here). We reproduce the experimental parameters of the earlier work by testing sets of 6 and 18 DHs, and use the same Euclidean heuristics of dimension 6 and 18. The default heuristic is the zero heuristic.

The results across 10,000 problems are shown in Figure 1. Greedy utility maximization (*MaxU*) gives a set of 6 DHs that are still not competitive with a 6-dimensional Euclidean heuristic. However, 18 greedily chosen DHs dominate their Euclidean counterpart across all solution lengths. Meanwhile, the DHs placed using the *Farthest* algorithm excel only on a minority of the longest paths. This is expected of *Farthest*, since such paths tend to start or end on the landmark states.

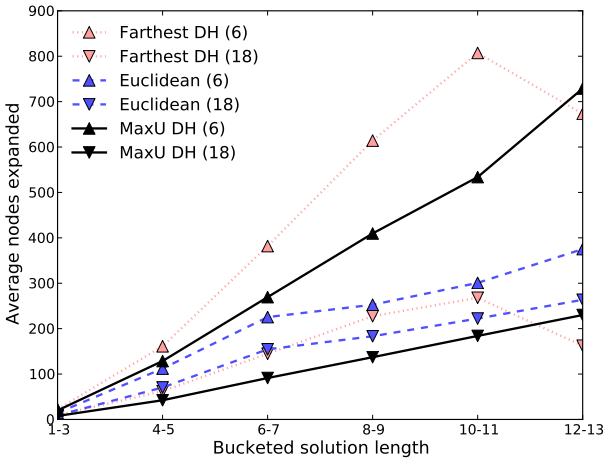

Figure 1: *Word Search* results averaged over 10,000 problems. Through greedy utility maximization, a set of 18 differential heuristics dominates an 18-dimensional Euclidean heuristic.

Game Maps

Next we consider standard pathfinding benchmarks [Sturtevant, 2012] defined on grid-based maps from BioWare's *Dragon Age: Origins*. The agent can move cardinally at cost 1 or diagonally at cost 1.5 among open grid cells, as long as doing so does not cut the corner of any closed grid cell. We consider all benchmark problems on fully connected maps with between 168 and 18,890 states, using the travel cost assuming no closed cells as the default heuristic.

We compare three selection algorithms. The first is greedy utility maximization under $\overline{\mathcal{U}}$, where the partitions are defined using a radius of 1 when a map has less than 10,000 states, and a radius of 2 otherwise. Second, we consider a greedy edge-covering approach [Fuchs, 2010]. This measure of utility is also submodular monotonic, so a comparison is warranted. To improve its efficiency and facilitate an even comparison, we modify this approach to use the same sample states as used by $\overline{\mathcal{U}}$. Third, we use the *Farthest* algorithm, which tends to be very effective on the benchmark problems. Each approach is used to define sets of 3 and 10 DHs.

Figure 2 shows our approach achieving a marked improvement over both algorithms on the vast majority of problems. Even though all three approaches have access to the default heuristic during search, our approach benefits from explicitly incorporating it into how the heuristic subset is chosen.

5.2 Directed Search Graphs

Since our approach can be applied to *any* class of heuristics, an important practical use is to compare competing classes of heuristics on a level playing field. In the following, we compare three kinds of heuristics for directed search graphs.

A search graph is *directed* if, for any two states i and j, $\delta(i,j) \neq \delta(j,i)$. DHs can still generate consistent heuristics on such graphs if the distances are computed on an undirected version of the graph with $\delta(i,j) \leftarrow \min\{\delta(i,j), \delta(j,i)\}$, but we will also consider a new approach of *directed* differential heuristics (DDHs). DDHs use a pivot state p too, but each pivot defines two distinct heuristic functions. If distances are computed *to* the pivot, $h_{\overleftarrow{p}}(i,j) = (\delta(i,p) - \delta(j,p))^+$. If they are computed *from* the pivot, $h_{\overrightarrow{p}}(i,j) = (\delta(p,j) - \delta(p,i))^+$. A search graph with n states defines $2n$ DDHs, all consistent. DDHs are essentially a decoupling of the heuristics used by the ALT algorithm [Goldberg and Harrelson, 2005], which always stores both the *to* and *from* distances for each pivot. However, given fixed memory, the space of possible DDH subsets encapsulates and is exponentially larger than the space of possible ALT subsets. Nevertheless, the greedy algorithm will only experience a doubling in effort.

Each of these – DHs, DDHs, and ALT heuristics – has a defining drawback. A DH built on the undirected graph risks being based on inaccurate distances; a DDH suffers from returning 0 for at least half of the state pairs; and a single ALT heuristic occupies twice the memory of any one DH or DDH. We stress that the optimality bound on our approach readily applies in each case, despite these major idiosyncrasies.

Turning in Place

In our first comparison of these three heuristics, we add a state variable to the *Dragon Age* map, LAK101D, to describe the agent's current heading. The agent can advance along its current heading or turn left and right in 45 degree increments, as depicted in Figure 3. This results in 7,342 directed edges across 2,544 states. The cost to advance is 1, and the cost to turn is a variable which offers control over the difference in cost between an action and its reverse. The default heuristic is the travel cost assuming no closed cells and no cost to turn.

We explore how the cost to turn affects the performance of sets of 1 and 2 ALT heuristics, and sets of 2 and 4 DH and DDH heuristics. Each set is selected using greedy util-

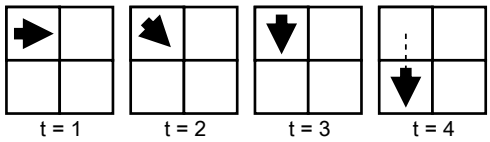

Figure 3: An east-facing agent turns before it can go south.

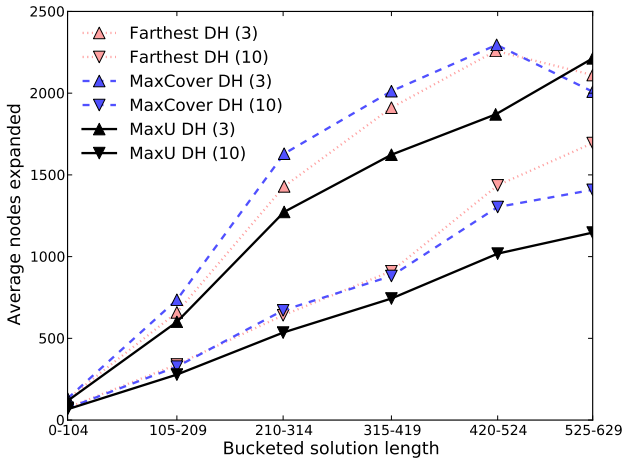

Figure 2: *Game Maps* results averaged across *Dragon Age: Origins* benchmarks using sets of 3 and 10 differential heuristics. Utility maximization proves to be an effective approach.

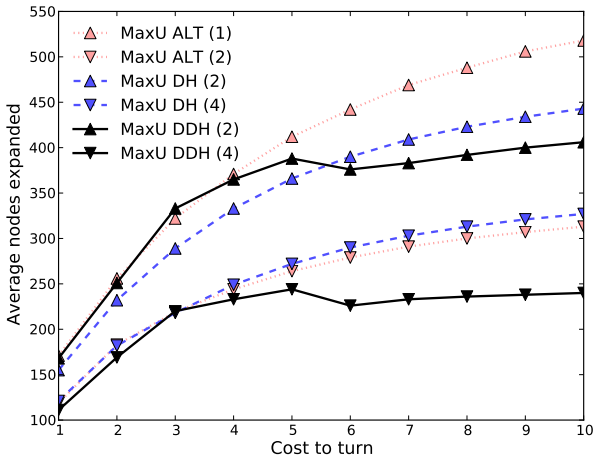

Figure 4: *Turning in Place* results across 10,000 problems. Differential heuristics are competitive at low turning costs, but higher costs lead to a preference for directed heuristics.

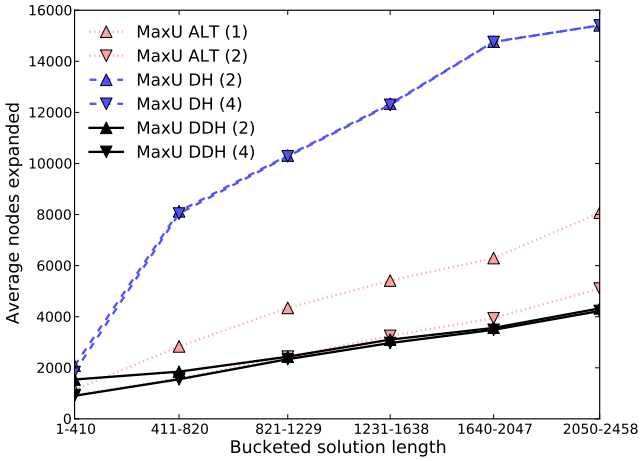

Figure 5: *Platformer* results averaged over 10,000 problems. Differential heuristics are ineffective, but *directed* differential heuristics excel at representing the distances in this domain.

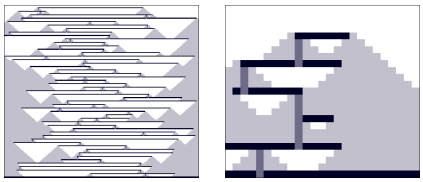

Figure 6: *Platformer* examples. Dark cells are platforms and ladders, grey cells are mid-air, and white cells are out of reach. The platformer on the left is used in our experiments.

ity maximization without sampling (that is, we directly measure utility using $\mathcal{U}$), and each set uses the same amount of memory. The results across 10,000 benchmark problems are shown in Figure 4. The search problems become more expensive to solve as the cost to turn increases, but DDHs seem least affected by this. A tradeoff between DHs and DDHs is also evident: when the cost to turn is small, 2 DHs are preferred over 2 DDHs, but this changes as the cost increases.

Platformer

The last test domain is defined on a grid of three types of cells: platforms, ladders, and mid-air. Examples are shown in Figure 6. If the agent is on a platform or ladder, it can move left and right, or up and down onto ladders. If the agent occupies a mid-air state, it automatically moves *down* one cell per turn, but can optionally control its descent left or right. The default heuristic is the number of steps assuming the agent can travel freely between any adjoining cells. This domain is modelled on a popular genre of video games which has received relatively little attention as a pathfinding benchmark, but is an extreme example of a cost structure that occurs in domains with actions that are difficult to reverse.

We define a random platformer on a 200×200 grid with 16,248 accessible states. The states are partitioned into regions whose radius is approximately 2, and greedy maximization of $\overline{\mathcal{U}}$ is used to choose sets of 1 and 2 ALT heuristics, and sets of 2 and 4 DH and DDH heuristics. Figure 5 shows the results on 10,000 random problems. These results reveal DHs to be ineffective at representing costs in this domain, due to the disparity between the cost of an action and its reverse. Meanwhile, we also see the DDH subsets consistently outperforming their ALT counterparts. With the knowledge that both are near-optimal under the same objective, we can infer that DDHs are well worth considering as an alternative class of search heuristics in domains with highly directed costs.

6 Conclusion

This paper introduced a novel approach to the widespread problem of selecting search heuristics. We showed that greedy selection is nearly optimal under a natural measure of utility, and furthermore that provable optimality is not sacrificed when utility is measured on just a sample of the states. These two points rely on the admissibility and consistency of the heuristics respectively, properties that are commonly satisfied by many existing classes of heuristic functions.

Our empirical study emphasizes the importance of optimality under a well-motivated objective. In particular, our method of approximate utility maximization redeemed an underperforming class of heuristics in one domain, outcompeted existing selection algorithms in another, and was instrumental in a pilot study of a promising new type of heuristic which excels in directed domains. The work in this paper has the potential to be extended in several new directions, such as by defining new measures of utility, more efficient sampling methods, and wider application to domains where heuristic subset selection defies domain expertise.

Acknowledgments

We are grateful to several anonymous reviewers who provided valuable feedback. This research was supported by NSERC and Alberta Innovates Technology Futures.

A *Post Factum* Bound on Optimality

Lemma 4 *Let θ and ϕ be set functions over 2^S with*

$$\min_{A:|A|=d} \theta(A) = 0, \quad \min_{A:|A|=d} \phi(A) = 0, \tag{38}$$

$$\forall A \in 2^S \quad |\theta(A) - \phi(A)| \leq \epsilon, \tag{39}$$

for some ϵ, and finite maxima at cardinality d denoted

$$\theta^* = \max_{A:|A|=d} \theta(A), \quad \phi^* = \max_{B:|B|=d} \phi(B). \tag{40}$$

For any set G with $|G| = d$, G's optimality with respect to ϕ is tied to its optimality with respect to θ:

$$\frac{\phi(G)}{\phi^*} \geq b \Rightarrow \frac{\theta(G)}{\theta^*} \geq b\frac{\phi(G) - \epsilon}{\phi(G) + b\epsilon} \tag{41}$$

Proof. The bound given on line 39 implies that θ^* and ϕ^* must also have bounded difference: $\theta^* - \epsilon \leq \phi(\arg(\theta^*)) \leq \phi^*$. This reveals an inequality relating θ^* to $\phi(G)$,

$$\phi(G) \geq b\phi^* \geq b(\theta^* - \epsilon) \Leftrightarrow \theta^* \leq \frac{\phi(G) + b\epsilon}{b}, \tag{42}$$

which lets us bound the optimality of $\theta(G)$ as

$$\frac{\theta(G)}{\theta^*} \geq \frac{\phi(G) - \epsilon}{\theta^*} \geq b\frac{\phi(G) - \epsilon}{\phi(G) + b\epsilon}, \tag{43}$$

thus proving the inequality. □

References

[Bagchi and Mahanti, 1983] A. Bagchi and A. Mahanti. Search Algorithms Under Different Kinds of Heuristics - A Comparative Study. *Journal of the ACM (JACM)*, 30(1):1–21, January 1983.

[Bauer *et al.*, 2010] Reinhard Bauer, Tobias Columbus, Bastian Katz, Marcus Krug, and Dorothea Wagner. Preprocessing Speed-Up Techniques is Hard. In *Proceedings of the 7th International Conference on Algorithms and Complexity*, CIAC'10, pages 359–370, Berlin, Heidelberg, 2010. Springer-Verlag.

[Bourgain, 1985] J. Bourgain. On Lipschitz Embedding of Finite Metric Spaces in Hilbert Space. *Israel Journal of Mathematics*, 52:46–52, 1985.

[Bulitko *et al.*, 2012] Vadim Bulitko, Chris Rayner, and Ramon Lawrence. On Case Base Formation in Real-Time Heuristic Search. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment conference (AIIDE)*, 2012.

[Culberson and Schaeffer, 1998] Joseph Culberson and Jonathan Schaeffer. Pattern Databases. *Computational Intelligence*, 14(3):318–334, 1998.

[Ernandes and Gori, 2004] Marco Ernandes and Marco Gori. Likely-admissible and Sub-symbolic Heuristics. In *ECAI*, pages 613–617, 2004.

[Fuchs, 2010] Fabian Fuchs. On Preprocessing the ALT-Algorithm. Master's thesis, Institute for Theoretical Informatics, Faculty of Computer Science, University of Karlsruhe, 2010.

[Goldberg and Harrelson, 2005] Andrew V. Goldberg and Chris Harrelson. Computing the Shortest Path: A* Search Meets Graph Theory. In *Symposium on Discrete Algorithms (SODA)*, pages 156–165, 2005.

[Goldberg and Werneck, 2003] Andrew V. Goldberg and Renato F. Werneck. Computing Point-to-Point Shortest Paths from External Memory. In *Proceedings of the 2005 SIAM Workshop on Algorithms Engineering and Experimentation*, 2003.

[Hart *et al.*, 1968] Peter Hart, Nils Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. on Sys. Sci. and Cybernetics*, 4(2):100–107, 1968.

[Holte *et al.*, 2006] Robert Holte, Ariel Felner, Jack Newton, Ram Meshulan, and David Furcy. Maximizing over Multiple Pattern Databases Speeds up Heuristic Search. *Artificial Intelligence Journal*, 170:1123–1136, 2006.

[Krause and Golovin, 2012] Andreas Krause and Daniel Golovin. Submodular Function Maximization. In *Tractability: Practical Approaches to Hard Problems*. Cambridge University Press, 2012.

[Krause and Guestrin, 2005] Andreas Krause and Carlos Guestrin. A Note on the Budgeted Maximization of Submodular Functions. Technical Report CMU-CALD-05-103, Carnegie Mellon University, June 2005.

[Linial *et al.*, 1995] Nathan Linial, Eran London, and Yuri Rabinovich. The Geometry of Graphs and Some of Its Algorithmic Applications. *Combinatorica*, 15:215–245, 1995.

[Nemhauser *et al.*, 1978] G. Nemhauser, L. Wolsey, and M. Fisher. An Analysis of Approximations for Maximizing Submodular Set Functions. *Mathematical Programming*, 14(1):265–294, 1978.

[Rayner *et al.*, 2011] Chris Rayner, Michael Bowling, and Nathan Sturtevant. Euclidean Heuristic Optimization. In *Proceedings of the Twenty-Fifth National Conference on Artificial Intelligence (AAAI)*, pages 81–86, San Francisco, CA, USA, 2011.

[Sturtevant *et al.*, 2009] Nathan R. Sturtevant, Ariel Felner, Max Barrer, Jonathan Schaeffer, and Neil Burch. Memory-Based Heuristics for Explicit State Spaces. *IJCAI-09*, pages 609–614, 2009.

[Sturtevant, 2012] Nathan R. Sturtevant. Benchmarks for Grid-Based Pathfinding. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(2):144–148, 2012.

[Weinberger *et al.*, 2005] Kilian Q. Weinberger, Benjamin D. Packer, and Lawrence K. Saul. Nonlinear Dimensionality Reduction by Semidefinite Programming and Kernel Matrix Factorization. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, pages 381–388, 2005.