Max Is More than Min: Solving Maximization Problems with Heuristic Search

Roni Stern Dept. of ISE Ben Gurion University roni.stern@gmail.com
Scott Kiesel Dept. of Computer Science University of New Hampshire skiesel@cs.unh.edu
Rami Puzis and **Ariel Felner** Dept. of ISE Ben Gurion University puzis,felner@bgu.ac.il
Wheeler Ruml Dept. of Computer Science University of New Hampshire ruml@cs.unh.edu

Abstract

Most work in heuristic search considers problems where a low cost solution is preferred (MIN problems). In this paper, we investigate the complementary setting where a solution of high reward is preferred (MAX problems). Example MAX problems include finding a longest simple path in a graph, maximal coverage, and various constraint optimization problems. We examine several popular search algorithms for MIN problems and discover the curious ways in which they misbehave on MAX problems. We propose modifications that preserve the original intentions behind the algorithms but allow them to solve MAX problems, and compare them theoretically and empirically. Interesting results include the failure of bidirectional search and close relationships between Dijkstra's algorithm, weighted A*, and depth-first search.

1 Introduction

One of the main attractions of the study of combinatorial search algorithms is their generality. But while heuristic search has long been used to solve shortest path problems, in which one wishes to find a path of minimum cost, little attention has been paid to its application in the converse setting, where one wishes to find a path of maximum reward. This paper explores the differences between minimization and maximization search problems, denoted as MIN and MAX problems, respectively.

To facilitate the discussion of MAX problems, consider the *longest simple path* problem (LPP). Given a graph $G = (V, E)$ and vertices $s, t \in V$, the task is to find a simple path from s to t having maximal length. A path is called *simple* if it does not include any vertex more than once. For weighted graphs, the task in LPP is to find a simple path with the highest "cost". LPP has applications in peer-to-peer information retrieval [Wong *et al.*, 2005], multi-robot patrolling [Portugal and Rocha, 2010], VLSI design [Tseng *et al.*, 2010], and coding theory [Kautz, 1958; Östergård and Pettersson, 2014; Singleton, 1966]. Also, some special cases of LPP such as the Snake-in-the-Box problem have important application in error correction codes [Singleton, 1966].

LPP is known to be NP-Hard [Garey and Johnson, 1979] and even hard to approximate [Karger *et al.*, 1997]. By contrast, its MIN variant – finding a shortest path – is a well-known problem that can be solved optimally in time that is polynomial the size of the graph, e.g., using Dijkstra's Algorithm [Dijkstra, 1959].

In this paper, we explore the fundamental differences between MIN and MAX problems and study how existing algorithms, originally designed for MIN problems, can be adapted to solve MAX problems. Surprisingly, this topic has received little attention in the academic literature.

Specific types of MAX problems, such as various constraint optimization problems [Dechter and Mateescu, 2007], oversubscription planning [Domshlak and Mirkis, 2015], and partial satisfaction planning [Benton *et al.*, 2009] have been addressed in previous work. In some cases, problem specific solutions were given and, in other cases, MIN problem algorithms were adapted to the specific MAX problem addressed. Nevertheless, to the best of our knowledge, our work is the first to provide a comprehensive study of uninformed and informed search algorithms for the MAX problem setting.

In this paper, we propose a set of general-purpose search algorithms for MAX problems. Optimal, bounded suboptimal, and unbounded MIN search algorithms are adapted to the MAX problem settings, and their theoretical attributes are discussed. We show that uninformed search algorithms must exhaustively search the entire search space before halting with an optimal solution. With an admissible heuristic, which overestimates future rewards, classical heuristic search algorithms can substantially speed up the search for MAX problems, but unlike MIN problems the search often cannot be stopped when a goal is expanded. We report experimental results for LPP over three types of graphs, demonstrating the importance of using intelligent heuristics in MAX problems.

2 MAX and MIN Search Problems

MIN problems are defined over a search space, which is a directed graph whose vertices are states and weighted edges correspond to operators. The weight of an edge is the cost of the corresponding operator. The task in a MIN problem is to find a path from an initial state s to a goal state whose sum of edge weights is minimal. MAX problems are defined similarly, except that operators have rewards instead of costs and the task is to find a path from s to a goal state whose sum

of edge weights is maximal. Non-additive costs/rewards are not addressed in this paper.

For graph problems like LPP, the input graph (through which we would like to find a longest path) is different from the graph representing the search space. In LPP, the search state must consider not only a single vertex in the input graph, but all vertices that have been included in or excluded from the path so far, in order to enforce simplicity of the path. For example, the search space in an empty $N \times N$ grid from the lower left corner to the upper right one is exponential in N ($O((2N)!)$), while the number of vertices in such a grid is N^2.

Some MAX problems assign rewards to states rather than operations and the objective is to find a state with maximal reward. Consider for example, the maximal coverage problem, where we are given a collection of sets $\mathcal{S} = \{S_1, ..., S_n\}$ and an integer k. The task is to find a subset $\mathcal{S}' \subseteq \mathcal{S}$ such that $|\mathcal{S}'| \leq k$ and $|\bigcup_{S_i \in \mathcal{S}'} S_i|$ is maximized. This problem can be reduced to LPP as follows. A state is a subset of $\mathcal{S}$, where the start state is the empty set. Applying an operator corresponds to adding a member of $\mathcal{S}$, and the reward of this operator is equal to the difference between rewards of respective states. A similar reduction to LPP is possible from any MAX problem with rewards to states.

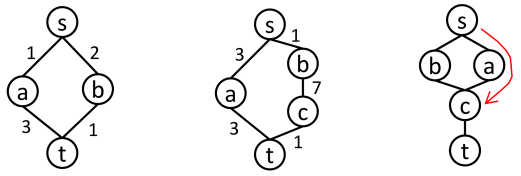
3 Uninformed Search for MAX Problems

First, we discuss several "uninformed search" algorithms that do not require a heuristic function: Dijkstra's algorithm (DA), depth-first branch and bound (DFBnB), and bidirectional search (BDS).

3.1 Dijksra's Algorithm

DA can be viewed as an instance of *best-first search* (BFS) [Felner, 2011]. BFS is an iterative algorithm in which one state is *expanded* in every iteration. Expanding a state v consists of generating all states reachable from v by applying a single operator. Generated states are stored in a list of states called OPEN. OPEN initially contains only s. As the search progresses, generated states enter OPEN and expanded states are removed from it. BFS algorithms differ in how they choose which state to expand. In DA, the state expanded is the one with the lowest g *value*. The g value of the start state is zero and the g value of all other states is initially ∞ (states can be initialized in a lazy manner, when they are generated). When a state u is expanded and a state v is generated, $g(v)$ is updated to be $\min\{g(v), g(u) + c(u, v)\}$, where $c(u, v)$ is the cost of the edge from u to v. In DA, the g-value of an expanded state u is guaranteed to be the lowest cost path found so far from the initial state s to u. When a goal node is chosen for expansion, the search halts and the path to it is guaranteed to be optimal, i.e., having the lowest cost.

How do we apply DA to MAX problems? For MAX problems, the g value as computed above corresponds to the reward collected so far. Expanding the state with the lowest g-value would return the path with the lowest reward. We could define DA for MAX problems to be a BFS that expands the state with the *highest* g-value. This way DA expands the next *best* state — with the lowest g-value (cost) in MIN problems



(a) DA fails (b) BDS fails (c) h_{CC} example

Figure 1: Example of different LPP instances

and with the highest g-value (reward) in MAX problems. Unlike its counterpart in MIN problems, however, this variant of DA does not necessarily find an optimal path. For example, in the graph depicted in Figure 1a, expanding the state with highest g would result in expanding b and then t. The returned path would be $\langle s, b, t \rangle$ while the optimal path is $\langle s, a, t \rangle$.

In order to find an optimal solution, DA for MAX problems must continue expanding states even after the goal is chosen for expansion, as better (higher reward) paths to a goal may still exist. One way to ensure that an optimal solution has been found is to expand states until OPEN is empty. When OPEN is empty, all paths from s to a goal had been considered and an optimal solution is guaranteed. We thus define DA for MAX problems to be a BFS on larger g-values with this simple stopping condition. Other sophisticated reachability mechanisms may exist that avoid this exhaustive exploration.

3.2 Search Space Monotonicity

To gain a deeper understanding of why DA is not particularly effective in MAX problems, we analyze the relation between the objective function (MAX/MIN) and the search space of a given search problem. Let G_S and s be the search space graph and the initial state, respectively.

Definition 1 (Search Space Monotonicity[1]). *A search space is said to be* monotone *w.r.t a state s if for any path P in G_S starting from s it holds that P is not* better *than any of its prefixes, where* better *is defined w.r.t the objective function.*

In MIN problems, a *better* path is a path with a lower cost, while in MAX problems, a better path is a path with a higher reward. It is easy to see that MIN problems have search spaces that are monotone. MAX problems, as we show below, have search spaces that are not monotone.

Next, we establish the relation between search space monotonicity and the performance of BFS in general and DA in particular. Each state v in OPEN represents a prefix of a possible path from s to a goal. When DA expands the best node on OPEN and it is a goal, search space monotonicity implies that its cost is not worse than the optimal solution cost. Thus, when DA expands a goal, it must be optimal and the search can halt. By contrast, in a search space that is not monotone, a prefix P' of a path P may be worse than P itself. Thus, the best g-value in OPEN is not necessarily better than the best solution. In such a case, DA may need to expand all the states in the search space that are on a path to a goal before halting. Moreover, some states may be expanded several times,

[1]This differs from *monotonicity* as proposed by Dechter and Pearl [1985] that describes the relation between a BFS evaluation function and solution costs.

as better paths to them are found. Thus, DA will not expand fewer states than a depth-first search (DFS) that exhaustively expands all states on all paths to a goal. Since DFS has a lower overhead per node than DA, it should perform better.

Search space monotonicity is related to the "principle of optimality", also known as the "optimal substructure" property. The "optimal substructure" property holds in problems where an optimal solution is composed of optimal solutions to its subproblems [Cormen *et al.*, 2001]. This property is needed for dynamic programming algorithms. The shortest path problem has the optimal substructure property as prefixes of a shortest path are also shortest paths. LPP, on the other hand, does not have the optimal substructure property as prefixes of a longest path are not necessarily longest paths. For example, consider LPP for the graph depicted in Figure 1a. The longest path from s to t passes through a and its cost is 4. The longest path from s to a, however, passes through b and t, and its cost is 6. Not having the "optimal substructure" property in general MAX problems prevents formulating MAX problems as a *cost algebra* [Edelkamp *et al.*, 2005]. Cost algebra is a formalism for representing a wide range of cost optimality notions, and Edelkamp et al. showed how to apply DA on any optimization problems formulated using a cost algebra.

3.3 Depth First Branch and Bound (DFBnB)

Search space monotonicity is also needed for DFBnB to be effective. DFBnB is an enhancement of DFS that prunes paths with cost worse than the incumbent solution (best solution found so far). It is an "anytime" algorithm, i.e., it produces a sequence of solutions of improving quality. When the state space has been completely searched (or pruned), the search halts and an optimal solution is returned.

Applying DFBnB on MAX problems is problematic, since a path can only be pruned if one knows that it is not a prefix of a better path to a goal. Thus, in uninformed search, *pruning with DFBnB only applies to monotone search spaces*. Without pruning any paths, DFBnB is plain DFS, enumerating all paths in the search space.

3.4 Bidirectional Search (BDS)

BFS is another type of uninformed search, in which two searches are performed simultaneously: a *forward search*, from the start state towards a goal, and a *backward search*, from goal states backwards towards the start.

For domains with uniform edge-costs and a single goal state, BDS runs a breadth-first search from both start and goal states. When the search frontiers meet and one of the searches has completed expanding all nodes in the current depth, then the search can halt and the min-cost path from start to goal is found. For MIN problems with uniform edge-costs, the potential saving of using BDS is large, expanding the square root of the number of states expanded by regular breadth-first search (or DA). With some modification, BDS can also be applied to problems with non-uniform edge-costs [Goldberg and Werneck, 2005].

Applying BDS to MAX problems poses several challenges. Consider running BDS on the graph in Figure 1b, searching for the LPP from s to t. For this example assume that the forward and backward searches alternate after every state expansion and both sides use DA for MAX (i.e., expand the state with the highest g). Vertex a is the first vertex expanded by both searches, finding a solution with a reward of 6, while the optimal reward is 9 (following $\langle s, b, c, t\rangle$).

Thus, unlike MIN problems, an optimal solution is not necessarily found when a state is expanded by both sides. Even if both sides used DA for MIN (expanding states with low g) an optimal solution would still not be returned.

The optimality of BDS for MIN problems depends on two related properties that do not hold in MAX problems. First, when a state is expanded, the best path to it has been found. As discussed above, this does not hold for MAX problems, and in general for non-monotone search spaces. Second, the lowest cost path from s to t is composed of an optimal path from s to some state x, concatenated with an optimal path from x to t. This is exactly the optimal substructure property, which as mentioned earlier does not hold for MAX problems.

In summary, in contrast to the MIN setting, DA for MAX problems cannot stop at the first goal, DFBnB offers no advantage over plain DFS, and BDS appears problematic. We are not familiar with an uninformed search algorithm that is able to find optimal solutions to MAX problems without enumerating all the paths in the search space.

4 Heuristic Search for MAX

In many domains, information about the search space can help guide the search. We assume such information is given in the form of a *heuristic function* $h(\cdot)$, where $h(v)$ estimates the remaining cost/reward of the optimal path from v to a goal. For MIN problems, many search algorithms that use such a heuristic function run orders of magnitude faster than uninformed search algorithms. Next, we discuss heuristic search algorithms for MAX problems.

4.1 DFBnB

In MIN problems, $h(v)$ is called *admissible* if for every v, $h(v)$ is a lower bound on the true remaining cost of an optimal path from the start to a goal via v. Given an admissible heuristic, DFBnB can prune a state v if $g(v) + h(v)$ is greater than or equal to the cost of the incumbent solution. This results in more pruning than DFBnB without h.

Similar pruning can be achieved for MAX problems, by adjusting the definition of admissibility.

Definition 2 (Admissibility in MAX problems). *A function h is said to be admissible for MAX problems if for every state v in the search space it holds that $h(v)$ is an* upper bound *on the remaining reward of an optimal (i.e., the highest reward) solution from the start to a goal via v.*

Given an admissible h for a MAX problem, DFBnB can safely prune a state v if $g(v) + h(v) \leq C$ where C is the reward of the incumbent solution. DFBnB with this pruning is very effective for some MAX problems [Kask and Dechter, 2001; Marinescu and Dechter, 2009].

4.2 A*

A* is probably the most well-known heuristic search algorithm [Hart *et al.*, 1968]. It is a BFS that uses an evaluation

function $f(\cdot) = g(\cdot) + h(\cdot)$. Let min_f and max_f denote $\min_{n \in OPEN} f(n)$ and $\max_{n \in OPEN} f(n)$, respectively. In every iteration, A* for MIN problems expands a state with f-value equal to min_f. Similiarly, we define A* for MAX problems to expand a state with f-value equal to max_f.

Lemma 1. *If h is admissible then for any BFS that stores all generated states in OPEN, min_f lower bounds and max_f upper bounds the cost/reward of an optimal solution.*

For MIN problems, A* is guaranteed to find an optimal solution when the first goal is chosen for expansion. This is because in MIN problems, an admissible h function must return zero for any goal state t and thus if t is expanded $f(t) = g(t) = min_f$ and thus $g(t)$ is optimal. This is not the case for MAX problems, where $h(t)$ may be larger than 0, e.g., when t is on a higher reward path to another goal. Therefore, while the $f(t)$ upper bounds the optimal solution (Lemma 1), $g(t)$ may not. To preserve optimality, A* for MAX problems should be modified to halt either when OPEN is empty, or when $max_f \leq C$. A similar variant of A* was proposed under the names Anytime A* or Best-First Branch and Bound (BFBB) in the context of partial satisfaction planning [Benton *et al.*, 2009] and over subscription planning [Domshlak and Mirkis, 2015].

The *consistency* property of a heuristic can also be adapted for MAX problems in a way that preserves its positive properties on search algorithms. See Stern et al. [2014] for a more detailed discussion.

5 Suboptimal Search for MAX

Solving problems optimally is often infeasible and suboptimal algorithms are often used in practice. Next, we investigate how to adapt classic suboptimal search algorithms to MAX problems.

5.1 Greedy Best First Search and Speedy Search

Greedy BFS (GBFS), also known as pure heuristic search, is a BFS that expands the state with the lowest h value. In some MIN problem domains, GBFS quickly returns a solution of reasonable quality. Can GBFS be adapted to MAX problems?

First, we analyze why GBFS is often effective for MIN problems. In MIN problems, h is expected to decrease as we advance towards the goal. Thus, expanding first states with low h value is expected to lead the search quickly towards a goal. In addition, states with low h value are estimated to have less remaining cost to reach the goal. Thus, choosing to expand states with low h values is somewhat related to finding a better, i.e., lower cost, solution. Therefore, in MIN problems, by expanding the state with the lowest h value, GBFS attempts to both reach a high quality goal, and also to reach it quickly, as desired for a suboptimal search algorithm. But what would be a proper equivalent in MAX problems?

In MAX problems, GBFS does not have this dual positive effect. Expanding the state with the *lowest* h value is expected to lead to a goal supposedly quickly, but for MAX problems a short path to the goal is expected to have low solution quality (if length is correlative to the amount of collected reward). The alternative of expanding the state with the *highest* h value would result in a breadth-first search behavior. Even if goal states had $h = 0$ (not necessarily true in MAX problems), then they would be chosen for expansion last, after all other states. That would make GBFS extremely slow, and much slower than A*! This was also supported in a set of preliminary experiments we performed, where a GBFS that expands the highest h was extremely inefficient. We thus use the term GBFS for both MAX and MIN problems to denote a BFS that always expands the state with lowest h in OPEN.

If edges in the state space have different costs, the length of a path to a goal may be different from the cost of that path. In such state spaces, *Speedy search* was proposed as an alternative to GBFS when the task is to find a goal as fast as possible [Ruml and Do, 2007]. It is a BFS on d, which is an estimate of the shortest path to a goal. Speedy search is well suited to MAX problems: d is defined in MAX problems exactly like in MIN problems, and Speedy for both MIN and MAX problems always expands the state with the lowest d. In the experimental section later, we show that Speedy search in MAX problems is substantially faster than GBFS.

6 Bounded Suboptimal Search for MAX

Bounded suboptimal search algorithms are suboptimal algorithms that accept a parameter w and guarantee to return a solution whose cost is bounded by w times the cost of an optimal solution. Formally, let C denote the cost of the solution returned by a suboptimal search algorithm and let C^* denote the cost of an optimal solution. A bounded suboptimal search algorithm for MIN problems guarantees that $C \leq w \cdot C^*$. Since C^* is the lowest-cost solution, bounded suboptimal algorithms can only find a solution for $w \geq 1$. We define a bounded suboptimal search algorithm for MAX problems similarly, as an algorithm that is guaranteed to return a solution whose reward is at least w times the highest-reward solution, i.e., that $C \geq w \cdot C^*$, where $0 \leq w \leq 1$.

Weighted A* [Pohl, 1970] is perhaps the first and most well-known bounded suboptimal search algorithm. It is a BFS, expanding in every iteration the state in OPEN with the lowest $f_w(\cdot) = g(\cdot) + w \cdot h(\cdot)$. When a goal is chosen for expansion, the search halts and the cost of the found solution is guaranteed to be at most $w \cdot C^*$. To achieve a similar guarantee for MAX problems, we modify WA* in two ways. First, WA* for MAX problems expands the state with the *highest* f_w in OPEN. Second, instead of halting when a goal is chosen for expansion, WA* for MAX problems halts only when the maximal f_w in OPEN (denoted max_{f_w}) is not greater than the reward of the incumbent solution.

Theorem 1 (w-Admissibility of WA* in MAX problems). *For any $0 \leq w \leq 1$, when $max_{f_w} \leq C$ then $C \geq w \cdot C^*$.*

Proof is omitted due to lack of space, and is basically similar to the equivalent proof in MIN problems.

Consider the behavior of WA* as w changes. When $w = 1$, WA* is equivalent to A*İn MIN problems, increasing w means that the evaluation function f_w depends more on h than on g. In general [Wilt and Ruml, 2012], this results in finding solutions faster but with lower quality (i.e., higher cost). In the limit, $w = \infty$ and WA* becomes GBFS.

To analyze the behavior of WA* in MAX problems, consider first the special case where $w = 0$. This is the equivalent

Problem	MIN	MAX
w=0	DA	DA that halts early = DFS
0<w<1	N/A	Worse quality, faster search
w=1	A*	A*
1<w<∞	Worse quality, faster search	N/A
w=∞	GBFS	N/A

Table 1: Weighted A* in MAX and MIN problems

of $w = \infty$ in MIN problems, having unbounded suboptimality. When $w = 0$ WA* becomes a BFS expanding the node with highest g. Thus WA* with $w = 0$ expands states in the same order as DA (as opposed to GBFS in WA* for MIN problem). There is, however a key difference between DA and WA* with $w = 0$. DA is intended to find optimal solutions and thus it halts only when OPEN is empty (as discussed earlier). By contrast, WA* for MAX problems halts when the incumbent is larger than or equal to the highest f_w value in OPEN. As a result, WA* for MAX problems with $w = 0$, assuming a reasonable tie-breaking policy, is exactly DFS that halts when the first solution is found! We explain this next.

Since $w = 0$, then for every state v, we have $f_w(v) = g(v)$. Let v be the most recent state expanded. By definition, v had the highest g value in OPEN. Its children have g values that are the same or higher and, therefore, one of these children will be expanded next. This continues until either a goal is chosen for expansion, having the highest g value in OPEN, and thus the search halts, or alternatively, a dead-end is reached, and then the best state in OPEN would be one of its immediate predecessors. An earlier predecessor cannot be better than one further along the path as long as edge weights are non-negative. Note that this is exactly the backtracking done by DFS. DFS is known to find solutions quickly in MAX problems. Thus, WA* is fast for MAX problems, as it is for MIN problems, but for a very different reason.

More generally, decreasing w from one to zero has two effects. First, WA* behaves more similarly to DFS (converging to it when $w = 0$). This in general results in finding a solution faster. Second, lowering w allows lower quality solutions to be returned. The behavior of WA* for MIN and MAX problems is summarized in Table 1.

The A^*_ϵ algorithm represents a different class of bounded suboptimal search algorithms [Pearl and Kim, 1982]. In addition to OPEN, A^*_ϵ maintains another list, called FOCAL. FOCAL contains a subset of states from OPEN, specifically, those states with $g + h \leq w \cdot min_f$. While states in OPEN are ordered according to their f-value as in A*, the states in FOCAL are ordered according to d. This FOCAL-based approach was taken further by Thayer et al. [2011] in their EES algorithm, which uses an additional third ordering according to an inadmissible heuristic. We denote by *focal search* the general search algorithm framework where one ordering function is used to select FOCAL, and another ordering function is used to select which node to expand from FOCAL. A^*_ϵ and EES are thus instances of focal search.

Adapting focal search to MAX problems is easy. First, the states in FOCAL are now those states with f values greater than or equal to $w \cdot max_f$, where max_f is the largest f value in OPEN. Second, as in WA* for MAX problems, finding a goal is not sufficient to guarantee that the incumbent solution is w-admissible, and a different stopping condition is needed. Such a condition that guarantees w-admissibility can be derived from Lemma 1: halt if $w \cdot max_f \leq C$.

An alternative to the previously discussed BFS-based frameworks is to apply DFBnB and prune states that cannot lead to a solution that would improve on the incumbent solution by more than a factor of $\frac{1}{w}$. The corresopnding pruning rule is that a state v can be pruned if $f_w(v) \leq C$.

7 Empirical Evaluation

As a preliminary study of how the algorithms discussed in this paper behave, we performed a set of experiments on the LPP domain. We do not presume to claim that any of the evaluated LPP solvers is a state-of-the-art LPP solver. Different classes of LPP have different solvers, where some subclasses of LPP can be even be solved in polynomial time [Ioannidou and Nikolopoulos, 2013]. The theoretical computer science literature also studied different approaches to solve LPP, such as using randomized algorithms [Williams, 2009] to find paths of length k with high probability. We use LPP as an example MAX problem on which we investigate the general purpose MAX search algorithms described in this paper. Thus, we did not compare with

Three types of LPP domains were used: (1) **uniform grids** are 4-connected $N \times N$ grids with 25% random obstacles, where traversing each edge costs one, (2) **life grids** are the same grids but traversing an edge into a grid cell (x, y) costs $y + 1$ [Thayer and Ruml, 2008], and (3) **roads** are subgraphs of the US road network graph. From this relatively large graph (approx. 20 million vertices), we chose a random vertex and performed a breadth-first search around it up to a predefined number of vertices.

Heuristics

For uniform grids, we used the following admissible heuristic, denoted by h_{CC}. Let $v = \langle v_1, v_2, .., v_k \rangle$ be a state, where $v_1, .., v_k$ are the vertices on the path in G it represents. Let G_v be the subgraph containing exactly those nodes on any path from v_k to the target that do not pass through any other vertex in v. G_v can be easily discovered with a DFS from v_k. h_{CC} returns $|G_v| - 1$. It is easy to see that h_{CC} is admissible. As an example, consider the Figure 1c. $h_{CC}(\langle s \rangle) = 4$ while $h_{CC}(\langle s, a, c \rangle) = 1$. A similar heuristic was used for life grids. The heuristic computes G_v and count every vertex (x, y) as $y + 1$. For roads, we used the maximal weight spanning tree for G_v. This is the spanning tree that has the highest weight and covers all the states in G_v. This heuristic is admissible as every path can be covered by a spanning tree. The distance heuristic d (used by Speedy and focal searches) used was the shortest path to a goal, which was computed once for all states at the beginning of the search.

Optimal Algorithms

We compared A*, DFBnB with pruning using an admissible heuristic, and a DFS enumeration of all paths in the search

Algorithm	Network size 100	300	500	700	1000
Speedy	100	100	100	100	100
DFS	100	83	78	73	64
GBFS	100	81	65	52	52

Table 2: Solved LPP instances on roads

space. DFS was chosen to represent uninformed search algorithms, as its computational overhead is very small and all uninformed search algorithms we discussed had to search through all paths in the search space to guarantee optimality.

Figure 2a show the average runtime as a function of the domain size in roads. As expected, exhaustive DFS performs poorly compared to A* and DFBnB. The differences between A* and DFBnB is very small. DFBnB is faster, while they expand almost exactly the same number of nodes (not shown). Thus, the slight advantage of DFBnB in runtime is due to the overhead of A* such as the need to maintain OPEN. Very similar trends were observed for uniform and life grids.

Suboptimal Algorithms

Next, we compared three unbounded suboptimal search algorithms: GBFS (denoted as "Greedy"), Speedy search, and plain DFS. All algorithms halt when the first goal was found.

Table 2 shows the number of instances, out of a hundred, solved by each algorithm under 5 minutes for different sizes of road networks. As the network grows larger, Speedy is able to solve many more instances than both DFS and Greedy, as d perfectly estimates the shortest distance to a goal. The poor performance of GBFS emphasizes that its heuristic (h_{CC}) is ill-suited to guide a search quickly to the goal. Furthermore, the computation of h_{CC} is more costly than the one-time computation of the shortest path required for computing d.

Complementing the view above, Figure 2b shows the reward achieved by each algorithm (averaged over the instances solved by all). Interestingly, DFS finds better solutions. This is because DFS does not aim to find a solution of low or high reward. By contrast, both Speedy and Greedy aim to find a solution quickly, which in LPP results in a short solution with small reward. Speedy finds worse solutions compared to Greedy, because it uses a perfect d, leading it to the shortest path to a goal. By contrast, Greedy uses h_{CC}, which is not focused on short paths, and thus the solutions it finds are better than those found by Speedy. The trends reported above for both success rate and achieved reward were also observed in uniform and life grids.

Bounded Suboptimal Algorithms

Next, we compared the bounded suboptimal variant of DFBnB presented earlier, WA*, A*$_\epsilon$, and EES. Following recent work, we implemented simplified versions of A*$_\epsilon$ and EES which perform iterative deepening instead of maintaining OPEN and FOCAL [Hatem and Ruml, 2014]. These simplified version were shown to be more efficient in most cases than the original versions of the algorithms.

Figure 2c presents performance on 13x13 life grids. The x-axis represents the desired suboptimality bound and the y-axis depicts runtime in seconds (in log scale, to better differentiate the algorithms). DFBnB and WA* performs best with some small advantage to DFBnB. The focal searches perform worse. Similar trends were shown in roads and uniform grids, where the advantage of DFBnB and WA* over the focal searches was slightly larger. The relatively poor performance of the focal search algorithms is caused by the overhead of maintaining FOCAL (even with the iterative deepening scheme mentioned earlier), and the poor accuracy of the additional d heuristic. Devising better focal searches for MAX is left for future work.

Concluding, we observed from the results the expected benefit of heuristic search algorithms for MAX problems. If the task is to find a solution as fast as possible, Speedy search is the algorithm of choice. For finding optimal solutions DFBnB with a heuristic performs best, with A* close behind. For bounded suboptimal the best performing algorithms were DFBnB with a suitable heuristic ($w \cdot h$) and WA*, while focal searches performed worse. DFBnB performed well in our domain (LPP) because the search space has no duplicates (since a state is a path), finding an initial solution is easy, and the proposed heuristics were relatively accurate. All of these are known weak points of DFBnB: it does not detect duplicates, it performs no pruning until the first solution is found, and a bad heuristic may cause it to commit early to an unpromising branch. Evaluating the performance of DFBnB against the other heuristic algorithms on MAX problem domains without these properties is left for future work.

8 Conclusion and Future Work

We explored a range of search algorithms for solving MAX problems. Using the notion of search space monotonicity, we showed how classical uninformed search algorithm for MIN problems cannot be applied efficiently to MAX problems, often requiring exhaustive search of the entire search space. Heuristic search algorithms, however, can be effective in MAX problems. We reestablished several key properties of searching in MAX problems, and showed how these algorithms can speed up the search, thus demonstrating the potential of heuristic search for MAX problems. Applying advanced search techniques and heuristics are exciting directions for future work.

9 Acknowledgments

This research was supported by the Israel Science Foundation (ISF) under grant #417/13 to Ariel Felner and by the US NSF under grant #1150068 to Wheeler Ruml.

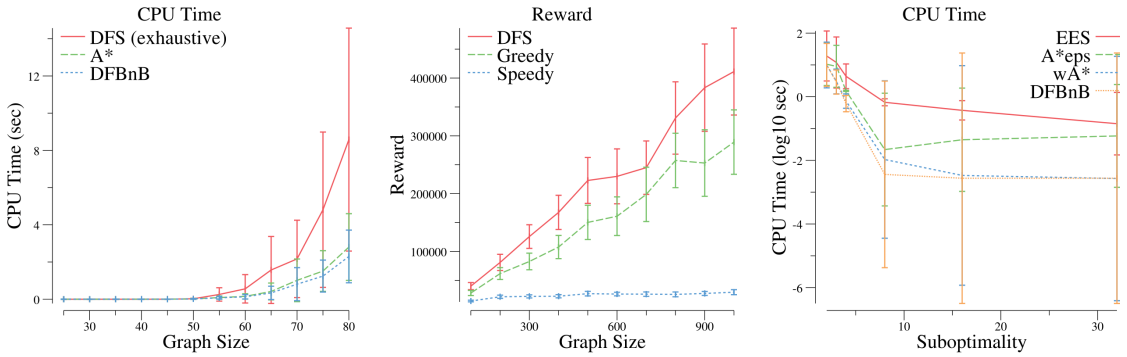
References

[Benton *et al.*, 2009] J Benton, Minh Do, and Subbarao Kambhampati. Anytime heuristic search for partial satisfaction planning. *AIJ*, 173(5):562–592, 2009.

[Cormen *et al.*, 2001] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2 edition, 2001.

[Dechter and Mateescu, 2007] Rina Dechter and Robert Mateescu. And/or search spaces for graphical models. *Artificial intelligence*, 171(2):73–106, 2007.

[Dechter and Pearl, 1985] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of a*. *Journal of the ACM (JACM)*, 32(3):505–536, 1985.



(a) Optimal, roads, CPU time (b) Unbounded, roads, reward (c) Bounded, life grids, CPU time

Figure 2: Optimal, bounded, and unbounded LPP solvers

[Dijkstra, 1959] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[Domshlak and Mirkis, 2015] Carmel Domshlak and Vitaly Mirkis. Deterministic oversubscription planning as heuristic search: Abstractions and reformulations. *JAIR*, pages 97–169, 2015.

[Edelkamp *et al.*, 2005] Stefan Edelkamp, Shahid Jabbar, and Alberto Lluch-Lafuente. Cost-algebraic heuristic search. In *AAAI*, pages 1362–1367, 2005.

[Felner, 2011] Ariel Felner. Position paper: Dijkstra's algorithm versus uniform cost search or a case against dijkstra's algorithm. In *SOCS*, pages 47–51, 2011.

[Garey and Johnson, 1979] Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. Freeman San Francisco, 1979.

[Goldberg and Werneck, 2005] Andrew V Goldberg and Renato Fonseca F Werneck. Computing point-to-point shortest paths from external memory. In *ALENEX/ANALCO*, pages 26–40, 2005.

[Hart *et al.*, 1968] Peter E. Hart, Nils J. Nilsson, and Bertram 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.

[Hatem and Ruml, 2014] Matthew Hatem and Wheeler Ruml. Simpler bounded suboptimal search. In *AAAI (to appear)*, 2014.

[Ioannidou and Nikolopoulos, 2013] Kyriaki Ioannidou and Stavros D. Nikolopoulos. The longest path problem is polynomial on cocomparability graphs. *Algorithmica*, 65(1):177–205, 2013.

[Karger *et al.*, 1997] David Karger, Rajeev Motwani, and GDS Ramkumar. On approximating the longest path in a graph. *Algorithmica*, 18(1):82–98, 1997.

[Kask and Dechter, 2001] Kalev Kask and Rina Dechter. A general scheme for automatic generation of search heuristics from specification dependencies. *Artificial Intelligence*, 129(1):91–131, 2001.

[Kautz, 1958] William H. Kautz. Unit-distance error-checking codes. *IRE Transactions on Electronic Computers*, 7:177–180, 1958.

[Marinescu and Dechter, 2009] Radu Marinescu and Rina Dechter. And/or branch-and-bound search for combinatorial optimization in graphical models. *Artificial Intelligence*, 173(16):1457–1491, 2009.

[Östergård and Pettersson, 2014] Patric RJ Östergård and Ville H Pettersson. Exhaustive search for snake-in-the-box codes. *Graphs and Combinatorics*, pages 1–10, 2014.

[Pearl and Kim, 1982] Judah Pearl and J.H. Kim. Studies in semi-admissible heuristics. *IEEE Trans. on PAMI*, 4(4):392–400, 1982.

[Pohl, 1970] Ira Pohl. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1(3–4):193–204, 1970.

[Portugal and Rocha, 2010] David Portugal and Rui Rocha. Msp algorithm: multi-robot patrolling based on territory allocation using balanced graph partitioning. In *ACM Symposium on Applied Computing*, pages 1271–1276, 2010.

[Ruml and Do, 2007] Wheeler Ruml and Minh Binh Do. Best-First Utility-Guided Search. In *IJCAI*, pages 2378–2384, 2007.

[Singleton, 1966] Richard C Singleton. Generalized snake-in-the-box codes. *Electronic Computers, IEEE Transactions on*, (4):596–602, 1966.

[Stern *et al.*, 2014] Roni Stern, Scott Kiesel, Rami Puzis, Ariel Felner, and Wheeler Ruml. Max is more than min: Solving maximization problems with heuristic search. In *Symposium on Combinatorial Search (SoCS)*, 2014.

[Thayer and Ruml, 2008] Jordan Tyler Thayer and Wheeler Ruml. Faster than weighted A*: An optimistic approach to bounded suboptimal search. In *ICAPS*, pages 355–362, 2008.

[Thayer and Ruml, 2011] Jordan T. Thayer and Wheeler Ruml. Bounded suboptimal search: A direct approach using inadmissible estimates. In *AAAI*, pages 674–679, 2011.

[Tseng *et al.*, 2010] I-Lun Tseng, Huan-Wen Chen, and Che-I Lee. Obstacle-aware longest-path routing with parallel milp solvers. In *World Congress on Engineering and Computer Science (WCECS)*, volume 2, 2010.

[Williams, 2009] Ryan Williams. Finding paths of length k in o(k2) time. *Information Processing Letters*, 109(6):315–318, 2009.

[Wilt and Ruml, 2012] Christopher Makoto Wilt and Wheeler Ruml. When does weighted A* fail? In *SOCS*, pages 137–144, 2012.

[Wong *et al.*, 2005] Wan Yeung Wong, Tak Pang Lau, and Irwin King. Information retrieval in p2p networks using genetic algorithm. In *WWW*, pages 922–923, 2005.