Proceedings of the Seventh Annual Symposium on Combinatorial Search (SoCS 2014)

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 the longest simple path in a graph, maximal coverage, and various constraint optimization problems. We examine several popular search algorithms for MIN problems — optimal, suboptimal, and bounded suboptimal — 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 a discovered close relationships between Dijkstra's algorithm, weighted A*, and depth-first search. This work demonstrates that MAX problems demand their own heuristic search algorithms, which are worthy objects of study in their own right.

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 motivate the discussion on MAX problems, consider the *longest simple path* problem (LSP). 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 si called *simple* if it does not include any vertex more than once. For weighted graphs, the task in LSP is to find the simple path with the highest "cost". LSP has applications in peer-to-peer information retreival (Wong, Lau, and King 2005), multi-robot patrolling (Portugal and Rocha 2010), and VLSI design (Tseng, Chen, and Lee 2010).

LSP is known to be NP-Hard (Garey and Johnson 1979) and even hard to approximate (Karger, Motwani, and Ramkumar 1997). By contrast, its MIN variant – finding the shortest path – is a well-known problem that can be solved optimally in polynomial time using Dijkstra's Algorithm (Dijkstra 1959).

LSP can be formulated as a search problem as follows. A state is a simple path in G. The start state is a path containing only s. An operator extends a path by appending a single edge $e \in E$. The reward of applying an operator is the weight of the added edge.

Another class of MAX problems assigns 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 LSP 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 this operator is equal to the difference between rewards of respective states.

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. To our surprise, this topic has received little attention.

Edelkamp et al.(2005) proposed *cost algebras* as a formalism for a wide range of cost optimality notions beyond the standard additive cost minimization. They then showed how common search algorithms can be applied to any problem formulated as a cost algebra. While cost algebras are very general, MAX problems cannot be formulated as one. This is because cost algebra problems must exhibit the "principle of optimality" (Edelkamp, Jabbar, and Lluch-Lafuente 2005), and we show later in this paper that this principle does not hold in MAX problems.

Specific types of MAX problems, such as various constraint optimization problems (Dechter and Mateescu 2007), oversubscription planning (Mirkis and Domshlak 2013), and partial satisfaction planning (Benton, Do, and Kambhampati 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. The main contribution of this work is in laying the theoretical foundation of applying a range of combinatorial search algorithms for MAX problems. We show that uninformed search algorithms must exhaustively search the entire search space before halting with the optimal solution. With an admissible heuristic, which overestimates the marginal reward in MAX problems, classical heuristic search algorithm can still substantially speedup the search. Yet, in MAX problems, the search often cannot be stopped when a goal is expanded. We report experimental results for LSP over three types of graphs. These results show the importance of using intelligent heuristics in MAX problems.

MAX and MIN Search Problems

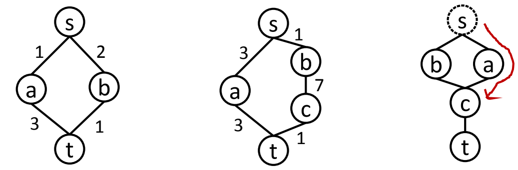
MIN problems are defined over a search space, which is a directed graph whose vertices are states and weighted edges correspond to operators. The weight of an edge is the cost of the corresponding operator. The task in a MIN problem is to find a path from an initial state s to a goal state whose sum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

Note that for graph problems like LSP, the problem graph (e.g.,the graph where we would like to find the longest path in) can be substantially different from the graph representing the search space. In LSP, for example, the search space can be exponentially larger than the underlying graph. For example, the number of simple paths in an empty $N \times M$ grid from the lower left corner to the upper right one is exponential in N and M, while the number of vertices in such a gris is $N \times M$.

It is possible to reduce a MAX problem to a MIN problem by represeting costs with negative or reciprocal rewards. Later, we show that such reduction affects the *monotonicity* of the search space, causing many search algorithms to fail.

Uninformed Search for MAX Problems

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



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

Figure 1: Example of different LSP instances

Dijksra's algorithm

BFS in general is an iterative algorithm, choosing to *expand* a single state in every iteration. Expanding a state v consists of generating all the states that can be reached 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.

Dijksra's algorithm (DA) (for MIN problems) can be implemented and explained as a best-first search (BFS) that chooses to expand the state with the lowest g-value in OPEN (Felner 2011). The g value of the start state is set to zero, while the g value of all other states is initially set to be ∞ (this initialization can also be done lazily, as states 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 a 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 expanded, the search halts and the best 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. Intuitively, 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 in both MAX and MIN problems — state with the lowest g-value (cost) in MIN problems and the with the highest g-value (reward) in MAX problems. Unfortunately, this variant of DA for MAX problems does not necessarily find the optimal highest reward 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 then be $\langle s, b, t \rangle$ while the weighted longest path is $\langle s, a, t \rangle$.

In order to find an optimal solution, DA for MAX problems is required to continue expanding states even after the goal has been expanded, as better (higher reward) paths to a goal may still exist. One way to ensure that an optimal solution has been found is by continuing to expand states until OPEN is empty. When OPEN is empty, all paths from s to a goal had been considered and the optimal solution is guaranteed. For the remainder of the paper we thus define DA for MAX problems to be a best-first search on larger g-values with this simple stopping condition. We note that other sophisticated reachability mechanisms may exist that avoid this exhaustive exploration.

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 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, while MAX problems 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. In BFS, OPEN contains a set of generated states. Each such state v represents a prefix of a possible path from s to a goal. This prefix is the best path found so far from s to v, and its cost is $g(v)$. OPEN in a BFS contains the best prefixes found so far to all possible paths 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, as better paths to them are found.

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 various dynamic programming algorithms. The shortest path problem has the optimal substructure property as prefixes of a shortest path are also shortest paths. LSP, on the other hand, does not have the optimal substructure property as prefixes of the longest path are not necessarily longest paths. For example, consider LSP 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.

Depth First Branch and Bound

As an alternative to DA, one can use any complete graph search algorithm. For example, one might prefer to run a depth-first search (DFS) to exhaustively expand all states on paths to a goal, as it does not need a priority queue (OPEN) as DA requires. Branch and bound is a popular enhancement to DFS that continues the DFS after a path to a goal is found, pruning paths with cost worse than the incumbent solution (best solution found so far) (Zhang and Korf 1995). DFBnB 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 the optimal solution is returned.

However, applying DFBnB on MAX problems is problematic. In MIN problems, paths with cost higher than the cost of the incumbent solution are pruned. In MAX problems, paths with reward higher than the incumbent are preferred. Pruning paths with a reward lower than the reward of the incumbent solution is also not possible, as these pruned paths may be prefixes of paths with much higher reward, potentially higher than the incumbent solution.

More generally, DFBnB can only prune paths if the search space is monotone. A path can be pruned if one knows that it is not a prefix of a path to a goal that is better than the incumbent. Thus, path pruning is possible if a path can only be a prefix to an equal or worse path. Thus, in uninformed search, pruning with DFBnB only applies to monotone search spaces. Without pruning any paths, DFBnB is plain depth-first search, exhaustively iterating over all paths in the search space. Thus, DFBnB in MAX problem would perform very similarly to DA in MAX problem. In fact, the only difference is the ordering by which paths are traversed and the overhead DA incurs by maintaining OPEN. Since all paths are traversed by both algorithms anyhow, a simple DFS would likely provide an easier and faster (due to less overhead per state) alternative to DA.

Bidirectional search

When there is a single goal state, another popular uninformed search method is bidirectional search, denoted here as BDS. For domains with uniform edge costs, BDS runs a breadth-first search from both start and goal states. When the search frontiers meet, i.e., when a state in the forward search generates a state in the backward search or vise versa, 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 be applied to problems with non-uniform edge costs. DA is executed from the start and goal states. Whenever the search frontiers meet, a solution is found. The incumbent solution is guaranteed to be optimal when the state expanded by the forward search was already expanded by the backward search, or vice versa. Goldberg et al. (2005) provided the following, improved stopping condition. Let $\min_f$ and $\min_b$ denote the lowest g-value in OPEN of the forward and backward search, respectively. The incumbent solution is guaranteed to be optimal when its cost is less than or equal to $\min_f + \min_b$.

Applying BDS to MAX problems poses several challenges. Consider running BDS on the graph in Figure 1b, searching for the LSP from s to t. For this example assume that the forward and backward searches alternate after state

[1] This differs from *monotonicity* as proposed by Dechter and Pearl (1985), as their notion is between the BFS evaluation function f and the solution cost function.

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, the 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) the 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. The second property which BDS is implicitly based on is that the lowest cost path from s to t is composed of the optimal path from s to some state x, concatenated with the 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 a uninformed search algorithm that is able to find optimal solutions to MAX problems without enumerating all the paths in the search space.

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 many MIN problems, 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.

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 every state whose $g + h$ cost 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 $h(v)$* upper bounds *the remaining reward of the 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 cost of the incumbent solution. DFBnB with this pruning is very effective for some MAX problems (Kask and Dechter 2001; Marinescu and Dechter 2009).

A*

A* is probably the most well-known heuristic search algorithm (Hart, Nilsson, and Raphael 1968). It is a BFS that uses an evaluation function $f(\cdot) = g(\cdot) + h(\cdot)$. By the minimal f-value in OPEN, we refer to the f-value of the state in OPEN with the lowest f-value (we define maximal f-value in OPEN similarly). For MIN problems, in every iteration, the state with the lowest f-value in OPEN is expanded. When a goal is expanded, the search halts and the best path found to so far that goal is returned. If h is admissible, then the minimal f-value in OPEN is guaranteed to lower bound all solution costs. Thus, when A* expands a goal state, that goal has the minimal f-value in OPEN and is therefore optimal.

With some minor modification, A* can be successfully applied to MAX problems. In MAX problems, the optimal solution upper bounds all other solutions. Therefore, we wish to define the f-values of states such that the maximal f-value in OPEN upper bounds all solution costs. To do this, we assume h is admissible for MAX problems (Definition 2) and we define A* for MAX problem to expand the node with the *highest* f-value in OPEN. Let max_f denote the highest f value in OPEN.

Lemma 1. *If h is admissible for MAX problems, then for any BFS that stores all generated states in OPEN, max_f upper bounds the reward of the optimal solution.*

The proof is completely analogous to the MIN case and is omited due to space limitations.

The first significant difference between A* for MAX and for MIN arises when considering goals. In MIN problems, an admissible h function must return zero for a goal state. Thus for any goal state t, $f(t) = g(t)$. This is not the case for MAX problems as $h(t)$ may be larger than 0. This can be a result of an overestimating h function. Alternatively, $h(t)$ can be accurate and t is on a path to another goal state with a higher reward. Therefore, while the $f(t)$ upper bounds the optimal solution (Lemma 1), $g(t)$ may not, since $h(t)$ may be larger than 0. Thus, unlike A*for MIN problems, expanding a goal with A*in MAX problems does not guarantee that the optimal solution has been found.

To preserve the optimality of A*for MAX problems, its stopping condition can be modified as follows: A*will halt either when OPEN is empty, or when max_f is not larger than the reward of the incumbent solution. 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, Do, and Kambhampati 2009) and over subscription planning (Mirkis and Domshlak 2013).

Trivial Heuristic Another difference between A* for MAX and MIN problems arising from the definition of admissibility is how the *trivial heuristic* is defined. The trivial heuristic, which is mostly used as a theoretical and pedagogical construct, is an admissible heuristic function that ignores the state it evaluates. It is usually denoted by h_0, because in MIN problems, h_0 simply maps every state to zero. This lower bounds the cost of all paths to a goal, and is thus admissible. A* with this heuristic is equivalent to DA.

In MAX problems, h_0 would be ∞, upper bounding the cost of all paths to a goal. Alternatively, if an upper bound U on the reward of the optimal solution is known, then a more accurate trivial heuristic would be $h_0(n) = U - g(n)$. A* using these trivial heuristics would end up assigning all states with the same f value (either ∞ or U). As a result states could be expanded in any order (depending on tie breaking), unlike DA for MAX. Moreover, goal states would also recieve the same f value. Thus, A* would continue to expand states exhaustively until OPEN is empty or, for $h_0 = U - g$, until all paths with reward U are expanded. Thus, A* with the trivial heuristic ultimately expands the same states as DA for both MIN and MAX problems.

Consistency In MIN problems, a heuristic h is said to be *consistent* if, for any two states x and y with a path between them, it holds that $h(x) \leq c_{min}(x, y) + h(y)$ where $c_{min}(x, y)$ is the cost of the least-cost path from x to y (Hart, Nilsson, and Raphael 1968). Running A* with a consistent heuristic causes the f values of states to be *monotonic non-decreasing* along any path in the search space. When a state v is expanded by A* with a monotonic f function, then it is guaranteed that $g(v)$ is the lowest-cost path from the start to v. This has the positive impact of not needing to reopen states that have already been expanded, causing the worst-case time complexity of A* to be linear in the number of states in the state space.

An equivalent definition exists for MAX problems:

Definition 3 (Consistency in MAX problems). *A heuristic function h is said to be consistent in MAX problems if, for any two states x and y with a path from x to y it holds that $h(x) \geq r_{max}(x, y) + h(y)$, where $r_{max}(x, y)$ is the reward of the maximum-reward path from x to y.*

This definition preserves the usual properties of A*

Theorem 1. *The following properties are guaranteed when running A*with a consistent heuristic in a MAX problem:*

- *f is* monotonically non-increasing, *i.e., generated states always have f values not greater than their parent..*
- *When a state v is expanded by A* $g(v)$ is the highest-reward path from the start to v.*
- *After a state is expanded, it is never reopened, resulting in a linear time complexity for A**

Proof. Let v' and v be states such that v' generated v and let $r(v', v)$ denote the reward on the edge from v' to v. By definition $r_{max}(v', v) \geq r(v', v)$ and thus

$$\begin{aligned} f(v') = g(v') + h(v') &\geq g(v') + r_{max}(v', v) + h(v) \\ &\geq g(v) + h(v) = f(v), \end{aligned}$$

establishing that f is monotonic.

Next, we prove that if v is expanded by A* then $g(v)$ is optimal, i.e.,. no better path exists to v. By contradiction, assume that there exists a path π from start to v with reward higher than $g(v)$. This means that there exists a state v' in OPEN through which π passes on the way to v. As v was expanded by A* we have that $f(v) \geq f(v')$. Also, since $g(v)$ is not optimal and the optimal path to v passes through v', it holds that $g(v') + r_{max} > g(v)$. Therefore,

$$\begin{aligned} f(v) &\geq f(v') = g(v') + h(v') \\ &> g(v) - r_{max}(v', v) + r_{max}(v', v) + h(v) = f(v), \end{aligned}$$

resulting in the contradicting $f(v) > f(v)$.

The last part of the theorem is a direct consequence of $g(v)$ being optimal when expanded – no better path to it will ever be expanded. □

The last part of Theorem 1 states that the complexity of A*is linear in the size of the search space if h is consistent. Note that for LSP on a graph G , the size of the search space may be exponential in the size of G, as the search space consists of all the possible paths in G.

In the experimental results given later in the paper, we also observe that the heuristic search algorithms preserve their substantial advantage over uninformed search.

Suboptimal Search for MAX

So far, we have discussed optimal search algorithms, i.e., algorithms that are guaranteed to return optimal (maximal, in MAX problems) solutions. As solving problems optimally is often infeasible, suboptimal algorithms are often used in practice. Next, we investigate how classic suboptimal search algorithms can be adapted to MAX problems.

Greedy Best First Search

Greedy BFS (GBFS), also known as *pure heuristic search*, is a BFS that expands the state with the lowest h value in every iteration. 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 (especially at the beginning of the search) to expand states with low h values is somewhat related to finding better 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. For admissible h values it is reasonable to expect that as the search advances towards a goal, the h value would decrease as the upper bound on future rewards should decrease as the sum of future possible rewards decreases. However, unlike in MIN problems low h values suggest low quality solutions – solutions with a low reward.

Thus, in MAX problems expanding the state with the lowest h value would lead to a goal supposedly quickly, but that would also lead to extremely low solution quality. However, 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 expanded 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.

Domains with Non-Unit Edge Costs

In domains with non-unit edge costs, it can be the case that the length of a path to a goal is not equal to the cost of that path. This occurs if the edges in the state space have different costs. In such cases, one can define two additional heuristic functions: $d_{nearest}$ or $d_{cheapest}$ Both heuristics estimate the shortests path (in edges, rather than as a sum of costs) to goal, but $d_{cheapest}$ estimate the shortest path to the lowest-cost goal, while $d_{cheapest}$ estimate to the shortest path to the closest (in terms of number of edges, not cost) goal. Previous work showed that $d_{nearest}$ and $d_{cheapest}$ can be used to find solutions quickly in problems with non-unit costs (Thayer and Ruml 2009; 2011). In particular, *Speedy search*, a best-first search on $d_{nearest}$, was proposed as an alternative to GBFS when the task is to find a goal as fast as possible (Ruml and Do 2007).[2] Note that the solution quality of Speedy search is often lower than GBFS. Speedy search is well suited to MAX problems. $d_{nearest}$ is defined in MAX problems exactly like in MIN problems. Furthermore, because the search algorithm is defined without respect to cost, it remains exactly the same for MIN and MAX problems: always search first the state with the lowest $d_{nearest}$ In the experimental section given later in this paper, we show that Speedy search in MAX problems is substantially faster than GBFS, which uses h.

Bounded Suboptimal Search for MAX

Bounded suboptimal search algorithms are a special class of suboptimal algorithms in which the algorithm accepts a parameter w and guarantees to return a solution whose cost is bounded by w times the cost of the optimal solution. Formally, let C denote the cost of the solution returned by a suboptimal search algorithm and let C^* denote the cost of the 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 $0 \leq w \leq 1$. Next, we investigate how common bounded suboptimal search algorithms can be adapted to MAX problems.

Weighted A*

Weighted A* (Pohl 1970) is perhaps the first and most well-known bounded suboptimal search algorithm. It is a best-first search, expanding in every iteration the state in OPEN with the lowest $f_w(\cdot) = g(\cdot) + w \cdot h(\cdot)$. When a goal is expanded, the search halts and the cost of the found solution is guaranteed to be at most $w \cdot C^*$.

[2] It is not clear from previous work whether Speedy was defind on $d_{nearest}$ or $d_{cheapest}$ We assume here $d_{nearest}$ as it follows best the logic behind Speedy: find a goal as fast as possible.

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

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 expanded, WA* for MAX problems halts only when the maximal f_w in OPEN is not greater than the reward of the incumbent solution.

Theorem 2 (w-Admissibility of WA* in MAX problems). *For any $0 \leq w \leq 1$, when the maximal f_w value in OPEN is less than or equal to the reward of the incumbent solution C, then it is guaranteed that $C \geq w \cdot C^*$.*

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* In 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 $w = 0$. Note that this the equivalent of $w = \infty$ in MIN problems. When $w = 0$ WA* becomes a best-first search expanding the node with highest g. Thus WA* with $w = 0$ expands states in the same order as DA (as appose 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 depth-first search that halts when the first solution is found! We explain this in more detail 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 expanded, 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 similar to DFS (convergin to it when $w = 0$). This in general results in finding a solution faster. Second, as in MAX problems, lowering w allows lower quality solutions to be returned. The behavior of WA* for MIN and MAX problems is summarized in Table 1.

Focal Search

There are many bounded suboptimal BFS algorithms. The A^*_ϵ algorithm uses a different approach to bound the returned solution (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 $g + h \leq w \cdot min_f$, where min_f is the smallest f-value in OPEN. While states in OPEN are ordered according to their f-value, as in A*, the states in FOCAL are ordered according to the estimated number of operators to reach the nearest goal (e.g., using $d_{nearest}$ or $d_{cheapest}$). This general approach of using OPEN and FOCAL lists 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 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 relatively straightforward. 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, the stopping condition is different. As in WA* for MAX problems, finding a goal is not sufficient to guarantee that the incumbent solution is w-admissible.

Lemma 2. *In any search that maintains all generated states in OPEN, and for any* $0 \leq w \leq 1$, *if* $w \cdot max_f \leq C$ *then* $w \cdot C^* \leq C$.

Lemma 2 is a simple derivation from Lemma 1.

The practical application of Lemma 2 is that focal search algorithms can halt when the cost of the incumbent solution is greater than or equal to $w \cdot max_f$. Note that $w \cdot max_f$ is smaller than $f_w(best_{f_w})$, so the stopping condition for focal searches is stronger than for WA*.

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 $f_w(v) \leq C$. We omit a formal proof due to space limitations.

Empirical Evaluation

As a preliminary study of how the algorithms discussed in this paper behave, we performed a set of experiments on the LSP domain. Three types of LSP 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 $h_{CC}(\langle s, a, c \rangle) = 1$.

The complexity of computing h_{CC} for a given state $v = \langle v_1, .., v_k \rangle$ is mostly affected by the complexity of constructing G_v. We implemented this by performing a depth-first search from the goal vertex, adding to G_v all states except those on the path to v_k. The worst case complexity is thus the number of edges in the underlying graph G. Note that G can be exponentially smaller than the state space, and thus computing this heuristic can be worthwhile. Improvements to h_{CC} are discussed below under future work, as our focus here is not to propose a state-of-the-art LSP solver.

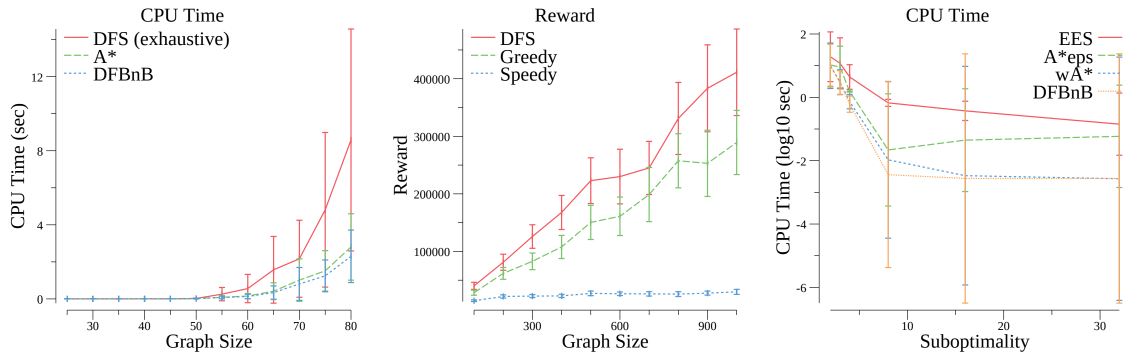
A similar heuristic was used for life grids. The heuristic computes G_v and count every vertex (x, y) as $y + 1$. For roads, we used the maximal weight spanning tree for G_v. This is the spanning tree that has the highest weight and covers all the states in G_v. This heuristic is admissible as every path can be covered by a spanning tree.

Optimal Algorithms First, we experimented with algorithms that return optimal solutions. We experimented with A*, DFBnB with pruning using an admissible heuristic, and plain DFS that enumerates all paths in the search 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, where DFBnB showing a slight advantage. When considering the number of nodes expanded (not shown), A* and DFBnB expand almost exactly the same number of nodes in this domain. 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 experimented with unbounded suboptimal search algorithms. We implemented GBFS (denoted as "Greedy") and Speedy search. In LSP, we can easily compute prefect $d_{nearest}$, which is the shortest path to a goal. $d_{nearest}$ was computed once for all states at the beginning of the search. Both Speedy and Greedy halt when the first goal was found. We also compared DFBnB with the same stopping condition, which is effectively DFS (with node ordering), as DFBnB only prunes states after the first solution is found.

We experimented on different sizes of road networks and



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

Figure 2: Optimal, bounded, and unbounded LSP solvers

	Network size				
Algorithm	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 LSP instances on roads

counted the number of instances, out of a hundred, solved by each algorithm under 5 minutes. The results are given in Table 2. As can be seen, as the graph grows larger, Speedy is able to solve many more instances than both DFS and Greedy, as its heuristic perfectly estimates the shortests 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 $d_{nearest}$.

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 LSP results in a short solution with small reward. Speedy finds worse solutions compared to Greedy, because it uses a perfect $d_{nearest}$, leading it to the shortest path to a goal. By contrast, Greedy uses h_{CC}, which is not focussed on short paths, and thus the solutions it finds are better than those found by Speedy. The same trends observed for both success rate and achieved reward were also observed in uniform and life grids.

Bounded Suboptimal Algorithms Our final category is bounded suboptimal search algorithms. We experimented with 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 effiecient 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 $d_{cheapest}$ 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 soultion 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, and for bounded suboptimal the best performing algorithms were DFBnB with a suitable heuristic ($w \cdot h$) and WA*, while focal searches performed worse. The good performance of DFBnB in our domain (LSP) is understandable, as the search space has no duplicates (since a state is a path). Additionally, finding an initial solution in LSP is easy and the proposed heuristics were relatively accurate for the tested domain. 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.

Conclusion and Future Work

In this work we explored a range of uninformed and heuristic 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 resetablish several key properties of searching in MAX problems, and show theoretically and empirically how these algorithms speed up the search. This work demonstrates the potential of heuristic search methods in MAX problems. Thus, applying advanced search techniques such as pattern databases and heirarchical search would be an exciting direction for future work.

Acknowledgments

This research was supported by the Israel Science Foundation (ISF) under grant #417/13 to Ariel Felner.

References

Benton, J.; Do, M.; and Kambhampati, S. 2009. Anytime heuristic search for partial satisfaction planning. *Artificial Intelligence* 173(5):562–592.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 2001. *Introduction to Algorithms*. The MIT Press, 2 edition.

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

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

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

Edelkamp, S.; Jabbar, S.; and Lluch-Lafuente, A. 2005. Cost-algebraic heuristic search. In *AAAI*, 1362–1367.

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

Garey, M. R., and Johnson, D. S. 1979. *Computers and intractability*, volume 174. Freeman San Francisco.

Goldberg, A. V., and Werneck, R. F. F. 2005. Computing point-to-point shortest paths from external memory. In *ALENEX/ANALCO*, 26–40.

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* SSC-4(2):100–107.

Hatem, M., and Ruml, W. 2014. Simpler bounded suboptimal search. In *AAAI (to appear)*.

Karger, D.; Motwani, R.; and Ramkumar, G. 1997. On approximating the longest path in a graph. *Algorithmica* 18(1):82–98.

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

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

Mirkis, V., and Domshlak, C. 2013. Abstractions for oversubscription planning.

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

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

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

Ruml, W., and Do, M. B. 2007. Best-First Utility-Guided Search. In *IJCAI*, 2378–2384.

Thayer, J. T., and Ruml, W. 2008. Faster than weighted A*: An optimistic approach to bounded suboptimal search. In *ICAPS*, 355–362.

Thayer, J. T., and Ruml, W. 2009. Using distance estimates in heuristic search. In *ICAPS*, 382–385.

Thayer, J. T., and Ruml, W. 2011. Bounded suboptimal search: A direct approach using inadmissible estimates. In *AAAI*, 674–679.

Tseng, I.-L.; Chen, H.-W.; and Lee, C.-I. 2010. Obstacle-aware longest-path routing with parallel milp solvers. In *World Congress on Engineering and Computer Science (WCECS)*, volume 2.

Wilt, C. M., and Ruml, W. 2012. When does weighted A* fail? In *SOCS*, 137–144.

Wong, W. Y.; Lau, T. P.; and King, I. 2005. Information retrieval in p2p networks using genetic algorithm. In *WWW (Special interest tracks and posters)*, 922–923.

Zhang, W., and Korf, R. E. 1995. Performance of linear-space search algorithms. *Artificial Intelligence* 79(2):241–292.