

Planning for Multiple Preferences versus Planning with No Preference

Daniel Bryce
Utah State University
4205 Old Main Hill
Logan, UT 84322
daniel.bryce@usu.edu

Abstract

Many planning applications must address conflicting plan objectives, such as cost, risk, duration, and resource consumption and decision makers want to know the possible trade-offs. Traditionally, such problems are solved by invoking a single-objective algorithm (such as A*) on multiple, alternative preferences of the objectives to identify non-dominated plans. The less-popular alternative is to delay reasoning about preferences and directly optimize multiple plan objectives with a search algorithm like Multi-objective A* (MOA*).

We notice that the relative performance of these two approaches hinges upon the number of f-values computed for individual search nodes. A* may revisit a node several times (once for each preference) and compute a different f-value each time. MOA* visits each node once and may compute some number of f-values (each estimating the value of a different non-dominated solution constructed from the node). While A* does not share f-values between searches for different solutions, MOA* can sometimes find multiple solutions while computing a single f-value per node. However, in doing so, MOA* is often ignorant of alternative solutions. We study several techniques for computing MOA* f-values that seek to lower the per node cost while also seeking multiple alternative solutions. The results of extensive empirical comparison show that i) the performance of multiple invocations of a single-objective A* versus a single invocation of MOA* is often worse in time and quality, and ii) that techniques for balancing per node cost and exploration are promising.

1 Introduction

Most realistic planning problems have multiple competing objectives. It is common in practice to select a preference over the objectives and solve the problem with respect to this preference [7]. It is also common that the preference is *highly subjective*, and the solution is found without knowledge of possibly better, alternative solutions. Instead, by finding multiple solutions, a human can apply their subjective preference over the objectives to the solution set (with full knowledge of the trade-offs available). In this sense, finding multiple solutions can facilitate – or circumvent entirely – preference elicitation. The approach taken in many applications is to apply multi-objective reasoning [3, 13, 21, 10, 23, 19], and in this work we study multi-objective search for planning.

Finding Sets of Plans: There are two ways to find a set of solutions that trade-off the objectives differently: iterate a single objective algorithm over multiple preferences, or use a multi-objective algorithm with no assumptions about the preferences. However, the poor scalability of multi-objective heuristic search algorithms, such as Multi-objective A* (MOA*) [17] has lead to favoring the iteration a single-objective algorithms (such as A*) over different aggregations or bounds (i.e., preferences) on the objectives. Upon closer examination, we notice a fundamental inefficiency with multiple invocations of a single-objective algorithm: each search episode may expand many of the same search nodes and recompute the f-values (a relatively large cost in planning). The high number of redundant node expansions is especially pronounced when the set of non-dominated plans positively interact (i.e., the plans share a common subsequence of actions). Moreover, with multiple invocations of the single-objective algorithm, there may be no guarantee that each solution will be non-dominated with respect to the other solutions.

MOA*: MOA* generalizes A* to find multiple non-dominated solutions in a very straight-forward manner. A* searches for a single “best” solution in terms of one optimization metric captured by each node’s f-value. MOA* searches for multiple “best” solutions by permitting multiple f-values (a vector of f-values) per node. Each MOA* f-value is associated with a different non-dominated solution. If a node is in the open list and at least one of its f-values is non-dominated with respect to the f-values of the other nodes, then the node can be expanded.

Computing f-values: Noticing that having a single non-dominated f-value is enough to keep a node on the non-dominated search frontier is an important observation that we exploit in this work. Computing multiple h-values (to get multiple f-values) for a search node can be both good and bad. Having more h-values improves a node’s chance for being on the non-dominated search frontier, but also increases the per node cost. Ideally, we would like to compute only those h-values that are needed to keep solution-bearing nodes on the non-dominated frontier. Our intuition is that many plans positively interact, and if we can compute a single h-value (or a small set of h-values) for a set of interacting plans, then search will find these plans at a lower cost.

We explore several approaches to computing heuristics for MOA*: i) compute a single h-value per node that estimates the longest solution path, hoping that other solutions will positively interact, ii) compute a uniform grid of h-values per search node to avoid missing solutions that do not positively interact, iii) with probability $1 - p$ compute a uniform grid of h-values (as in ii), and with probability p compute the h-values that are similar to the parent node’s h-values that were non-dominated in the open list. We find (in our experiments) that the third approach is the most useful because it balances exploration with preservation of non-dominated partial solutions. We compare to the baseline approach of computing several solutions with A* by setting different bounds (preferences) on the plan objectives.

Probabilistic Planning: We compare A* and MOA* in probabilistic planning, where the plan length and probability of goal satisfaction are the competing objectives. Probabilistic planning largely simplifies our analysis because in partial solutions, one objective (plan length) is free to change, but the other objective (probability of goal satisfaction) is fixed (i.e., no partial plan collects the probability of goal satisfaction until it is a complete plan). The effect is that there is a single best g-value for each node, and the only way to obtain multiple f-values is to compute multiple h-values. Hence, we can focus solely on how to compute multiple h-values without considering multiple g-values. While we do not study partial satisfaction planning [18] in this work, we note that it largely resembles probabilistic planning from the perspective of a search algorithm, and we believe our techniques are applicable in this problem as well.

In the following, we present the MOA* algorithm, provide the intuition for investigating which f-values are computed in MOA*, discuss several approaches to computing h-values for MOA*, describe how to formulate probabilistic planning at A* and MOA* search, study the empirical performance of the techniques on several domains, discuss related work, and conclude with future research directions. Our contribution is to evaluate the relative strengths of single objective and multi-objective search, and show that multi-objective search is preferable when non-dominated solutions share common search nodes.

2 MOA*

MOA* [17] is a search algorithm that finds a set of paths, and relies on established notions of solution (non)dominance in multi-objective problem solving.

Let Π denote a set of solutions for some problem. Each solution $\pi \in \Pi$ is associated with a value vector $v(\pi) = (v_1(\pi), \dots, v_m(\pi))$ defining its quality for each of m objectives. A solution π *dominates* solution π' , denoted $\pi \prec \pi'$, if it is no worse than π' in all objectives, $v_i(\pi) \leq v_i(\pi')$, $i = 1, \dots, m$, and $v(\pi) \neq v(\pi')$. Let $\mathcal{E}(\Pi)$ denote the set of efficient (also called non-dominated) solutions, such that for no pair of solutions $\pi, \pi' \in \mathcal{E}(\Pi)$, $\pi \prec \pi'$ or $\pi \succ \pi'$. As such, we seek efficient solutions that minimize the objectives differently. We characterize the quality of a set of solutions by computing its hypervolume [20], the size of the objective space dominated by the solutions. As the efficient set gains quality, the hypervolume increases. Let $H(\Pi)$ denote the hypervolume (size of the dominated objective space) of a set of solutions $\Pi = \{\pi_1, \dots, \pi_n\}$, calculated as:¹

¹Our implementation uses an optimized, but equivalent computation for two objectives.

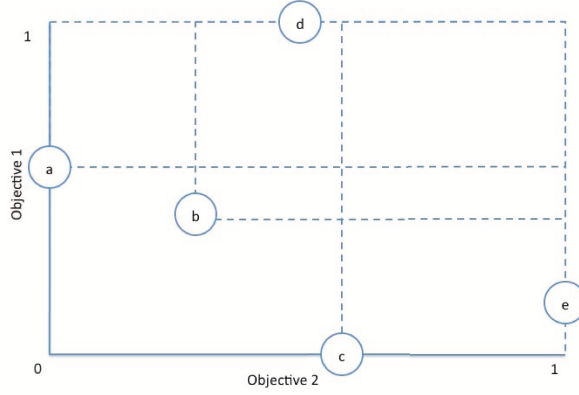


Figure 1: Two objective example (hypervolume depicted).

MOA*

1. Initialize *Open* with the start node. Initialize an empty set of nodes *Solutions*.
2. Find a subset of *Open*, denoted $\mathcal{E}(\textit{Open})$, of nodes with f-values not dominated by an f-value of any node in *Open* or *Solutions*.
3. If $|\mathcal{E}(\textit{Open})| = 0$, then exit returning solutions found by following efficient back-pointers from nodes in *Solutions*.
4. Otherwise, select node *n* from $\mathcal{E}(\textit{Open})$, remove from *Open*, and add to *Closed*.
5. If *n* is a solution node, then add *n* to *Solutions* and Goto 2.
6. Otherwise, Expand *n*, adding successors to *Open* (removing previously expanded successors from *Closed*) and computing successor f-values. Goto 2.

Table 1: MOA* Algorithm.

$$H(\Pi) = \sum_{i=1}^n (-1)^{i+1} \sum_{k_1 < \dots < k_i} \prod_{m=1}^M \left[1 - \max_{j=1, \dots, i} \left(\frac{v_m(\pi_{k_j}) - \underline{\alpha}_m}{\bar{\alpha}_m - \underline{\alpha}_m} \right) \right]$$

where $\bar{\alpha}_m$ and $\underline{\alpha}_m$ denote respective upper and lower bounds on the m^{th} objective (e.g., the lower and upper bounds on a solution's probability of success is zero and one, respectively).

Consider the example in Figure 1, where five solutions $\{a, \dots, e\}$ are shown. This example shows a normalized objective space (with two objectives) so that all solution values fall in the interval $[0, 1]$ in each objective. The solution *d* is dominated by both *a* and *b* because it is greater in both objectives, and, likewise, *e* is dominated by *c*. The efficient set is $\mathcal{E}(\{a, \dots, e\}) = \{a, b, c\}$, containing the non-dominated solutions. The rectangles encompassing the regions that are greater in both objectives than solutions *a*, *b*, and *c* denote the hypervolumes (in this case, areas) dominated by each solution. The union of the hypervolumes dominated by the set is the hypervolume of the set, and its magnitude indicates the quality of the set. Notice that the space dominated by the set $\{a, b, c\}$ is larger than the space dominated by $\{a, b\}$, making the former set a better solution set.

MOA* [17] (Table 1) and its variations [12] generalize the traditional, single-objective A* heuristic search algorithm by operating on graphs whose edge costs are vectors. Each efficient path from a start node to a terminal node (a solution) has an associated non-dominated value vector (equal to the sum of the edge costs). MOA* finds a set of efficient paths from a source node to one of the terminal nodes by maintaining efficient sets for the familiar, scalar A*

constructs (g-, h-, and f-values, and backpointers). The efficient set of g-values for each node represent the cost of all efficient paths reaching the node, the set of h-values represent the estimated cost of all efficient paths reaching terminal nodes, and the set of f-values contains all efficient members of the cross-product of g- and h-values (i.e., each f-value is the sum of a g-value with an h-value). Nodes in the MOA* *Open* list follow a strict partial order and have potential for expansion if one of their f-values are efficient with respect to f-values of other *Open* list nodes and previously found solutions. Our implementation expands a random efficient node from the *Open* list in each iteration.

MOA* is guaranteed to terminate with an optimal efficient set of solutions (a Pareto optimal set) when: i) edge cost vector components are all greater than zero, ii) the graph is locally finite, iii) the h-values are admissible, and iv) there are no cycles. In this work, we violate two of these assumptions: i) is violated in probabilistic planning because goal satisfaction is zero for non-terminal actions, and iii) is violated because the heuristics employed are inadmissible. As such, it is not guaranteed the MOA* will terminate, nor that it finds the optimal set of solutions. These features are obviously not very theoretically appealing, but we note that they are necessary in practice because scalability requirements and time constraints often prohibit optimality and termination. We adopt an empirical anytime approach to evaluating MOA* and characterize its performance over time. While our case-study violates the requirements for MOA* optimality, these limitations are easily overcome by employing an admissible heuristic and enforcing non-zero edge costs.

3 Computing f-values

The intuition for judiciously selecting which f-values to compute is based on the following observations. A* and MOA* can, and often will, expand the same search nodes to find a set of solutions. To find multiple solutions (each with respect to a different preference over the plan objectives), A* must compute a new f-value for re-expanded search nodes under each new preference. MOA* does not usually² re-expand nodes, and does not necessarily compute a different f-value for each preference over the plan objectives (nor does it require any preference).

Since it is possible for A* and MOA* to expand the same nodes and find the same solutions, whichever algorithm with a lower combined per node and per iteration cost will perform better. A*'s per node cost is dependent upon how many times the node is re-expanded (i.e., its f-value is re-computed for a new preference) and its per iteration cost relies on sorting the *Open* list. MOA*'s per node cost includes the cost of computing multiple f-values and its per iteration cost is predominantly incurred while finding the set of nodes with efficient f-values (its equivalent of sorting the *Open* list).³

With MOA*, it is possible to bootstrap a node's f-values to find multiple efficient solutions. For example, if a node must be expanded to find all efficient solutions, then it is enough that one of its f-values places it within the set of efficient nodes in $\mathcal{E}(Open)$. If an oracle can determine which f-value, among the set of possible f-values, will make the node most competitive within the efficient set (i.e., give the node the best chance of being efficient), then computing only this f-value will lead to a lower per node cost. This single f-value represents a set of positively interacting solutions to be found by expanding the node.

Without an oracle, there are several options for computing the f-values. All options balance two competing desires: lowering the per node cost and giving nodes the best chance at staying on the efficient frontier $\mathcal{E}(Open)$. In the following section, we examine three approaches to computing f-values for MOA* that address these competing desires.

4 Computing MOA* f-values

While there are many ways to compute f-values for MOA*, such as restricting the g-values or the combinations of g- and h-values, we focus solely on methods for computing h-values. These methods include computing i) a single h-value per node (called *M+*), ii) a uniform grid of n h-values (called *Mn*), and iii) a probabilistic choice between computing a uniform grid of h-values or a set of h-values from non-dominated h-values of the parent node (called *MNn*). Each of these methods involves placing an upper bound on all but one objective, and then computing the

²Nodes removed from the *Closed* list in step 6 of Table 1 are technically re-expanded, but f-values are only added, not recomputed.

³Our implementation of MOA* uses Kung's algorithm [4] to find the efficient set in each iteration, with complexity $O(n \log n)$ when there are two objectives and n is the number of f-values of all nodes in *Open*.

value of the free objective with respect to the bounds. We obtain multiple h-values by computing the free objective value with respect to different bounds. The following explores the intuition behind, and computation of, each method.

Single h-value $M+$: Computing a single h-value (i.e., selecting a single bound for all but one objective) is difficult because it must represent the “cost to go” of an efficient solution found via the node under consideration. Our desires from the previous section are to lower the per node cost (which can only decrease here by not computing an h-value) and to compute a most competitive h-value. Without computing additional h-values for comparison (e.g., by comparing the hypervolume dominated by each), it is difficult to determine which is the most competitive. Instead, we take an approach based on the intuition that, by estimating the cost of the solution that is deepest in the search graph, we can maximize potential for other solutions to be found while searching for the deepest solution. Each bound objective in the h-value is bound with its estimated deepest solution value, and the last, free objective is estimated with respect to the bounds. Subsequently all h-values for different nodes are compared solely on the basis of the free objective because the same bounds are used for the h-value of each node.

Multiple h-values Mn : Extending the $M+$ method to compute several h-values can improve the competitiveness of a search node, albeit at increased cost. While one h-value may never lead to an efficient f-value, another computed for the same node might. Similar to $M+$, Mn bounds the value of all but one objective and computes an h-value with respect to the bounds. Each of the n h-values bounds the values of the objectives differently. We evaluate this method by spacing the bounds uniformly, but note that other approaches that bias the spacing may be viable as well.

Probabilistically Non-Dominated h-values MNn : The $M+$ and Mn methods are two extremes that either use very few h-values to keep per node cost low or use many h-values to seek out many different plans. A simple way to combine the approaches is to probabilistically select between them when computing the h-values for a node. While $M+$ will save some effort expanding the node, it does not account for which h-values caused the current node’s parent to be expanded. There are likely to be only a very few efficient f-values that caused the parent to be expanded, so instead of computing a single value, we can compute a set of h-values similar to the parent’s efficient h-values (i.e., those h-values whose associated f-values were non-dominated with respect to all other f-values of nodes in the *Open* list). The h-values are similar to the parent h-values because they use the same bounds on the fixed objectives and recompute the free objective. In this manner, we can compute h-values that are likely to be most competitive. By probabilistically selecting between Mn and computing h-values similar to the parent node, we can both explore and preserve efficient h-values.

5 Case Study of Probabilistic Planning

Probabilistic planning is a naturally multi-objective problem, where at its simplest, the plan objectives are the plan length and probability of goal satisfaction. More precisely, we use the risk (one minus the probability of goal satisfaction) so that we can minimize each objective. We choose to study probabilistic planning because it has one property that largely simplifies our analysis and otherwise boosts empirical performance when comparing MOA* and A*. As previously noted, MOA* associates cost vectors with search graph edges; in probabilistic planning, all edges leading to non-terminal nodes incur unit cost with respect to plan length and zero cost with respect to risk, however, edges leading to terminal nodes incur zero cost for the plan length and a possibly non-zero risk cost. The effect of this property is that there is a single efficient g-value for each non-terminal node, meaning that multiple f-values for a search node can only arise from computing multiple h-values.

The following subsections define the conformant probabilistic planning problem, its formulation as a graph search problem in both A* and MOA*, and discussion of an existing reachability heuristic used in the search algorithms.

Conformant Probabilistic Planning: A conformant probabilistic planning problem is given by the tuple $CPP = (P, A, b_I, G)$, where P is a set of propositions, A is a set of actions, b_I is an initial belief state (probability distribution over initial states), and G is the goal description (a conjunctive set of propositions).

A belief state is a probability distribution over all states, where each state is a set of propositions. The probability of a state s in a belief state b , $P(s|b)$, is denoted $b(s)$. We say that a state s is in b ($s \in b$) if $b(s) > 0$. The probability that a belief state b satisfies the goal G , is the sum of the probabilities of states where the goal is satisfied ($G \subseteq s$).

An action $a \in A$ is a tuple $(\rho_e(a), \Phi(a))$, where $\rho_e(a)$ is an enabling precondition, and $\Phi(a)$ is a set of outcomes. The enabling precondition $\rho_e(a)$ is a conjunctive set of propositions that determines action applicability. An action a is applicable $appl(a, b)$ in belief state b if it is applicable in each state in the belief state, $\forall s \in b \rho_e(a) \subseteq s$. The causative outcomes $\Phi(a)$ are a set of tuples $(w_i(a), \Phi_i(a))$ representing possible outcomes (indexed by i), where $w_i(a)$ is the probability of outcome i being realized, and $\Phi_i(a)$ is a mutually-exclusive and exhaustive set of conditional effects (indexed by j). Each conditional effect $\varphi_{ij}(a) \in \Phi_i(a)$ is of the form $\rho_{ij}(a) \rightarrow (\varepsilon_{ij}^+(a), \varepsilon_{ij}^-(a))$, where both the antecedent (secondary precondition) $\rho_{ij}(a)$ and positive $\varepsilon_{ij}^+(a)$ and negative $\varepsilon_{ij}^-(a)$ consequents are conjunctive sets of propositions. This representation of effects follows the 1ND normal form presented by Rintanen [15]. As outlined in the probabilistic PDDL (PPDDL) standard [22], it is possible to use the effects of every action to derive a state transition function $T(s, a, s')$ that defines a probability that executing a in state s will result in state s' . Executing action a in belief state b , denoted $exec(a, b) = b_a$, defines the successor belief state such that $b_a(s') = \sum_{s \in b} b(s)T(s, a, s')$.

A sequence of actions (a_1, \dots, a_m) , executed in belief state b , results in a state b' , where $b' = exec(a_m, exec(a_{m-1}, \dots, exec(a_1, b) \dots))$ and each action is executable in the appropriate belief state. The probability that the sequence of actions satisfies the goal is the probability that the final belief state satisfies the goal. The number of actions in the sequence is the length. As long as the sequence of actions respects the definition of applicability of each action, the sequence (including the empty sequence) is a feasible plan, but not necessarily an efficient plan.

Formulating CPP as Graph Search: We formulate CPP as both A* and MOA* search over the belief state space. Each search node is a belief state, and each edge is an action.

A* search associates a unit cost with each edge, and defines terminal nodes as those nodes where the belief state’s probability of goal satisfaction is no less than a given threshold τ . The actions associated with edges leading to the terminal node identify the plan. We find multiple solutions with A* by invoking A* with different values for τ .

MOA* search associates a cost vector with each edge. The first component of the vector is the action execution cost (as in A*), and the second component is the risk. Risk is only incurred when transitioning to terminal nodes, and only action execution cost is incurred when transitioned to non-terminal nodes. MOA* treats terminal nodes differently than A* because it does not exit upon finding a solution (i.e., MOA* does not use a goal satisfaction threshold τ and pursues multiple solutions). MOA* adds a single, unique terminal node and unique edges for “quit” actions that allow the search to choose to transition from any belief state to the terminal node. The quit actions i) are applicable to all belief states, ii) incur zero action execution cost, iii) incur a risk cost equal to one minus the probability of goal satisfaction of the belief state where applied, and iv) are not added to the plan extracted from the edges leading to the terminal node. By using quit actions, MOA* can continue searching through nodes that satisfy the goal (with some probability), but retain the solutions.

Heuristics for CPP: The most effective heuristics for CPP involve estimating the cost to achieve the goal with probability no less than τ [2, 6]. We employ the McLUG technique described by Bryce et al. [2] to compute relaxed plans, using the number of actions as the heuristic. The approach taken by Bryce et al. [2] to compute the heuristic for a belief state and value of τ is to compute k deterministic planning graphs and extract a relaxed plan that achieves the goals in at least $k\tau$ of the planning graphs. Each planning graph is deterministic because it is built with respect to a state sampled from the belief state and sampled outcomes of each probabilistic action in each action layer. Symbolic techniques make the construction of multiple planning graphs and the relaxed plan efficient.

In MOA*, the heuristic is computed once for the bound $\tau = 1.0$ in $M+$. For example, when $\tau = 1.0$, the relaxed plan might contain ten actions; in this case, MOA* would add an h-value vector $(10, 0.0)$ (i.e., $1.0 - \tau = \text{risk}$). The Mn method computes the heuristic for each bound $\tau \in \{1/n, 2/n, \dots, n/n\}$, adding the set of h-value vectors $(c_1, 1 - 1/n), (c_2, 1 - 2/n), \dots, (c_n, 0.0)$, where c_1, c_2, \dots, c_n are the numbers of actions in the relaxed plans for each value of τ . With probability $p = 0.5$, the MNn heuristic uses the MN heuristic and, with probability $1 - p$ computes the relaxed plan heuristic for the same values of τ that the non-dominated parent h-values computed.

6 Empirical Evaluation

We compare A* to multiple versions of MOA* (using different heuristic computation strategies) on several CPP problems across four domains. The questions that we attempt to answer are:

- Will multiple invocations of A* with different preferences or one invocation of MOA* with no preferences find a better set of solutions, find the set faster, or both?
- Which method for computing h-values in MOA* will perform best?

The following describes the evaluation metrics, domains, the test environment, and results.

Evaluation Metrics: As previously mentioned, we measure the quality of a set of solutions by its hypervolume. We can measure the hypervolume over time with MOA*, but because A* finds a single solution at a time, only measuring its hypervolume over time is not as appealing. Thus, we compute a set of solutions using A* and also compare the total time taken by MOA* to find a plan set with the same or better hypervolume. We also compare the maximum hypervolume found by each technique, within twenty minutes for MOA*, and for the fixed number of solutions found by A* (where each invocation to find a solution is given a twenty minute limit).

Domains: The evaluation domains include an artificial domain, called Grid and Ladder (GL), and several domains from the CPP literature, including Logistics, Gripper, and Sand Castle.

The GL domain is an adaptation of the Grid domain presented by Hyafil and Bacchus [8] that adjusts the degree of positive interaction and independence among the non-dominated plans. GL includes five actions: move-right, move-left, move-up, move-down, and climb. The move actions have the effect of moving along the intended axis with 0.8 probability and the effect of moving laterally along the other axis with 0.1 probability in each direction. The initial state starts the agent at one corner of the grid (with certainty), and the goal is to reach the opposite corner and reach the top of the ladder. The deterministic climb ladder action can only be performed in the destination corner once for each rung of the ladder and, once executed, prevents further move actions (it is possible to climb the ladder, but not go down and move about the grid). The probability of achieving the goal is equal to the probability of reaching the corner with the ladder because the climb action does not change the probability of goal satisfaction (assuming the goal of being at the top of the ladder is attained by repeated climb actions). Alternative solutions perform different sequences of grid moves to reach the corner with different probabilities and plan lengths, and all plans must climb the ladder. The problems vary the size of the grid and the height of the ladder. A larger grid equates to longer positively interacting plan prefixes, and a ladder with more rungs equates to longer independent plan suffixes. GL presents challenges to MOA* because computing too many h-values during the grid traversal phase of the plan is costly, but additional h-values are required prior to the ladder climbing phase to push search to find alternative plans. GL also challenges A* because it must recompute the f-value for many of the same nodes within the plan prefix.

The other domains are not modified from their original versions to help gauge the MOA* approaches in problems without clear structure. Logistics [8] involves probabilistic (un)load actions and initial belief states where package locations are uncertain, and we use the instance p2-2-2, with two cities, two packages, and two possible locations for each package. Gripper [9] involves several machining operations to manufacture widgets that work probabilistically. Sand Castle [11] involves two actions to build a castle or dig a moat, both of which are probabilistic and effect the success of the other. GL and Logistics are relatively challenging domains that cause state of the art CPP planners to struggle, whereas in Gripper and Sand Castle are fairly simple.

Environment: All experiments were conducted on a 3Ghz Xeon processor running Linux with 8GB of RAM. All code was written in C++ and is based on the POND planner [2], and all hypervolume computations were done offline after planning. The McLUG heuristic computed for the GL domain used 32 planning graphs per h-value, and 128 planning graphs per h-value in the other domains. The comparisons of hypervolume across instances were with respect to the same lower and upper bounds on each objective (computed from the planner output). The results for A* are averaged over five runs of computing plans for the set of thresholds and the MOA* results are averaged over five invocations on each problem. A* is invoked with thresholds {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0} for each problem. Invocations not returning a solution are not counted in the results.

Results: Figure 2 shows the hypervolume achieved over time for nine problems in the GL domain, where each is a grid of size three, six, or ten, and has a ladder of height two, six, or ten. Each row of plots in the figure is a common grid size (increasing in size with each row), and each column shares a common height ladder (increasing from left to right). There are four methods shown in each plot: A*, MOA* with *M10*, MOA* with *MN2*, and MOA* with *MN10*. Some methods described in the accompanying Table 2 are not shown in the plots because they were not as

G / L	A* T(s)	A* HV	M+ T(s)	M+ HV	M2 T(s)	M2 HV	M10 T(s)	M10 HV	MN2 T(s)	MN2 HV	MN10 T(s)	MN10 HV
3 / 2	2.55	0.60	2.14	0.64	4.66	0.63	45.36	0.63	1.95	0.63	10.94	0.63
3 / 6	9.12	0.40	1211.68	0.41	10.48	0.43	109.56	0.43	27.24	0.42	37.64	0.42
3 / 10	42.08	0.27	-	-	24.58	0.29	208.48	0.28	61.90	0.27	82.44	0.27
6 / 2	34.67	0.47	47.33	0.50	322.80	0.47	-	0.39	21.07	0.48	45.23	0.47
6 / 6	69.47	0.29	470.21	0.29	107.82	0.30	-	0.26	44.75	0.28	58.99	0.30
6 / 10	238.14	0.23	-	-	207.63	0.24	-	0.17	137.30	0.23	90.50	0.24
10 / 2	481.47	0.38	139.32	0.39	349.38	0.38	-	-	122.49	0.38	319.03	0.38
10 / 6	562.31	0.26	-	0.18	753.27	0.26	-	-	361.63	0.26	588.06	0.26
10 / 10	867.51	0.18	-	-	845.73	0.18	-	-	205.28	0.18	719.37	0.20

Table 2: A* versus MOA* in GL Domain

	A* T(s)	A* HV	M+ T(s)	M+ HV	M2 T(s)	M2 HV	M10 T(s)	M10 HV	MN2 T(s)	MN2 HV	MN10 T(s)	MN10 HV
Log p2-2-2	32.01	0.15	129.00	0.41	98.03	0.16	-	-	54.30	0.28	132.64	0.28
Gripper	1.74	0.87	2.47	0.94	0.73	0.95	3.07	0.95	0.46	0.95	1.00	0.95
Sand Castle	2.33	0.88	0.91	0.89	1.34	0.96	4.79	0.96	1.34	0.96	2.59	0.95

Table 3: A* versus MOA* in Logistics, Gripper, and Sand Castle Domains

competitive. Table 2 includes results for the $M+$ and $M2$ methods, and the data is the average time taken in seconds to exceed the average hypervolume found by A* (T(s)) and the average maximum hypervolume found by each method (HV). The T(s) and HV results for A* are the average time taken to find the average hypervolume using A*, and the T(s) results for all other methods are the time taken to exceed the A* HV. The HV for all other methods is the maximum hypervolume found before the timeout. The “-” entries indicate that either no solution set could be found that exceeds the A* hypervolume, in the case of T(s), or that no single solution was found, in the case of HV.

The plots that show the hypervolume over time indicate that the MNn method tends to find the most hypervolume of the MOA* methods, outperforming $M+$ and Mn , especially as the problems become larger. We also see that as the problems become larger, using $MN10$ is required to find more hypervolume. A* tends to find its first solutions very quickly, and it takes longer for the MOA* methods to find their first solutions. However, MOA* tends to attain quite a bit of hypervolume earlier than A*. As the problems become more difficult, MOA* takes increasingly longer over A* to find initial solutions, but finds considerably more hypervolume within a short period of time thereafter. Note that small improvements in hypervolume later in the search are not insignificant because they represent new and improved solutions that may be difficult to find despite the small hypervolume they add.

Figure 3 and Table 3 present results for the other domains that show trends similar to those seen in the GL domain. The MOA* approaches tend to find solutions later than A*, but often find much better solutions as measured by maximum hypervolume. The difference in hypervolume is especially pronounced in the Gripper and Sand Castle problems because these problems have many solutions whose probability of goal satisfaction falls in the interval [0.9, 1.0]; and, because we only use the extreme points of this interval as values for τ in A*, it misses many solutions in the interval. MOA* does not use bounds on the objectives and can seek out all the solutions within intervals that are created by the bounds – leading to more hypervolume. By selecting an *a priori* set of preferences on the objectives, A* cannot find some solutions.

Summary: From our analysis, we have seen the following trends in comparing MOA* and A*:

- MOA* improves the quality of solution sets over A* because it is not limited to finding a predetermined number of solutions, and it continues to search and improve upon the solutions.
- The MNn method for computing h-values is the most effective MOA* technique because it balances exploration with the retention of efficient partial solutions.

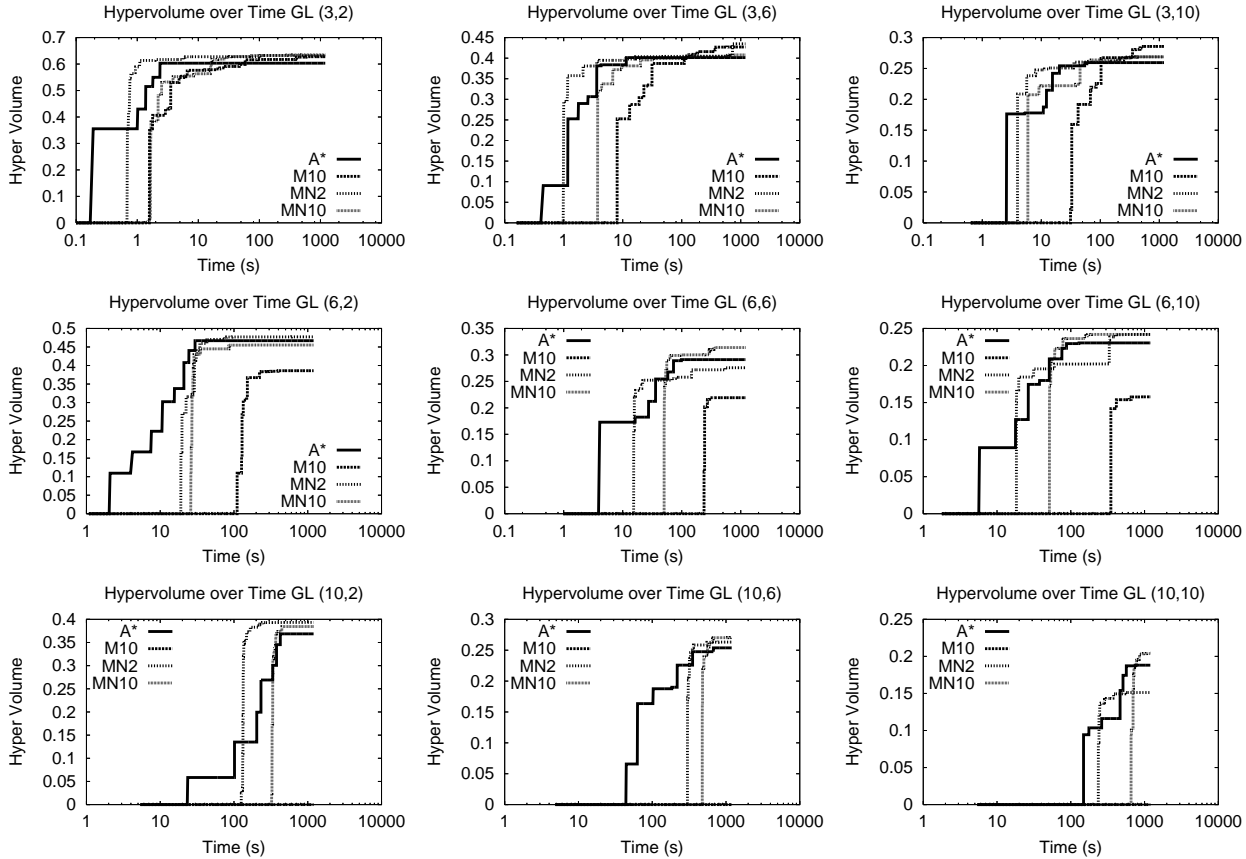


Figure 2: Hypervolume Over Time for GL Domain

- The $M+$ method does not pursue enough different solutions to attain high hypervolume, or the domains tested did not have a large degree of positive interaction among the solution sets.
- The Mn method was useful for finding high hypervolume in smaller problems, but failed to scale well.
- A^* is fast to find a first solution, but much slower than MOA^* at finding a set with large hypervolume.

7 Related Work & Discussion

Multi-objective problem solving has been previously studied in planning [7, 1, 14, 5, 18]. However, to our knowledge, all prior works study how to best formulate preferences over the objectives and solve a single objective problem. As discussed in the previous section, iteratively solving the problem with different preferences as a single objective problem can not only miss solutions, but may take considerably longer, or fail, to find a set of solutions with comparable quality.

The work of Van den Briel et al. [18] on partial satisfaction planning (PSP) is especially close to our study of probabilistic planning. PSP allows for the satisfaction of a subset of goals with utilities, where CPP allows for a probabilistic satisfaction of all goals. Thus, both problems have the same interesting property in MOA^* : there is only one g -value for each search node, and the only way to obtain multiple f -values is by computing multiple h -values. We expect that applying MOA^* to PSP would attain results similar to those presented in this work.

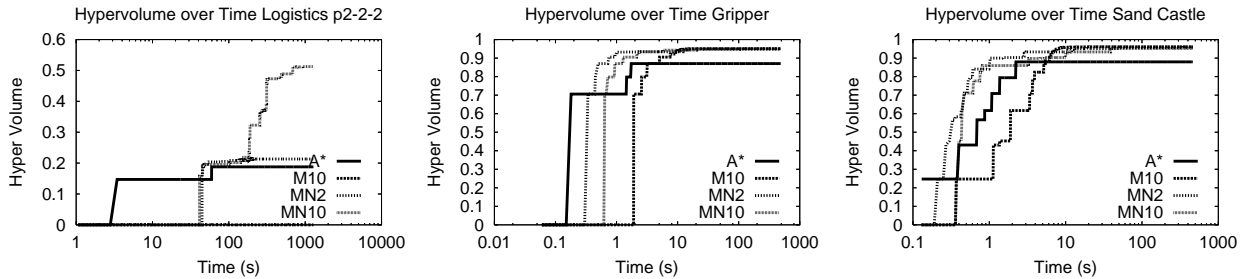


Figure 3: Hypervolume Over Time for Logistics, Gripper, and Sand Castle Domains

Finding a set of diverse solutions to planning problems is an important problem recently studied in planning [16]. Srivastava et al. [16] construct solutions that are diverse with respect to a distance measure, defined in terms of the causal structure of the plans. We take a different approach, where we find plans that are diverse in their objectives. These two approaches can be seen as complementary, the former finding diversity in the decision space, and the latter finding diversity in the objective space.

MOA* was originally studied by Stewart and White [17], showing conditions for optimality, and termination. More recently, Mandow and de la Cruz [12] modified the algorithm to expand the different f -values of a node within each iteration, where the original algorithm expanded the node itself. Our implementation is based on the original algorithm, and our work explores which f -values to compute, a topic not discussed by prior work on MOA*.

8 Conclusion and Future Work

We have shown that MOA* can find a better *set* of solutions than A* by balancing the explorative capability attained by increasing the number of f -values per node with the decreased per node cost associated with computing fewer f -values. The most effective technique for computing a node's f -values randomly chooses between computing a uniform grid of f -values and recomputing the f -values proven efficient by its parent node. By finding a better set of solutions more quickly, MOA* is a viable choice for computing multiple solutions for a problem. Moreover, the efficient solutions are naturally diverse in the objective space. The limitations of MOA* are that when users have clear preferences it is rendered unnecessary, and as found by our evaluation, it can be more efficient to perform multiple A* searches when the non-dominated solutions share few common search nodes.

In future work, we intend to explore additional techniques for computing h -values, methods for managing multiple g -values per search node, other types of planning problems (such as PSP), and planning with more objectives. We are also interested in combining our approach with techniques for finding plans that are diverse in the decision space (i.e., causally diverse).

References

- [1] D. Aberdeen, S. Thiébaux, and L. Zhang. Decision-theoretic military operations planning. In *Proceedings of ICAPS'04*, pages 402–412, 2004.
- [2] D. Bryce, S. Kambhampati, and D.E. Smith. Sequential Monte Carlo in probabilistic planning reachability heuristics. *Artificial Intelligence*, 172(6-7):685–715, 2008.
- [3] C Cheng, C Ou, and K Chau. Combining a fuzzy optimal model with a genetic algorithm to solve multi-objective rainfallrunoff model calibration. *Journal of Hydrology*, 268:72–86, 2002.
- [4] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley-Interscience Series in Systems and Optimization. John Wiley & Sons, Chichester, 2001.

- [5] M.B. Do and S. Kambhampati. Sapa: A scalable multi-objective metric temporal planner. *Journal of Artificial Intelligence Research*, 20:155–194, 2003.
- [6] C. Domshlak and J. Hoffmann. Fast probabilistic planning through weighted model counting. In *Proceedings of ICAPS’06*, pages 243–251, 2006.
- [7] P. Haddawy and S. Hanks. Representations for decision-theoretic planning: Utility functions for deadline goals. In *Proceedings of KR’92*, pages 71–82, 1992.
- [8] N. Hyafil and F. Bacchus. Utilizing structured representations and CSPs in conformant probabilistic planning. In *Proceedings of ECAI’04*, pages 1033–1034, 2004.
- [9] N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic least-commitment planning. In *Proceedings of AAAI’94*, pages 1073–1078, 1994.
- [10] Jian-Yi Lin, Chun-Tian Cheng, and Kwok-Wing Chau. Using support vector machines for long-term discharge prediction. *Hydrological Sciences Journal*, 51(4):599–612, AUG 2006.
- [11] S.M. Majercik and M.L. Littman. MAXPLAN: A new approach to probabilistic planning. In *Proceedings of AIPS’98*, pages 86–93, 1998.
- [12] L. Mandow and J.L. Perez de la Cruz. A new approach to multiobjective A* search. In *Proceedings of IJCAI’05*, pages 218–223, 2005.
- [13] Nitin Muttil and Kwok wing Chau. Neural network and genetic programming for modelling coastal algal blooms. *International Journal of Environment and Pollution*, pages 223–238, 2006.
- [14] Ioannis Refanidis and Ioannis Vlahavas. The MO-GRT system: Heuristic planning with multiple criteria. In *Proceedings of the Workshop on Planning and Scheduling with Multiple Criteria*, 2002.
- [15] J. Rintanen. Expressive equivalence of formalisms for planning with sensing. In *Proceedings of ICAPS’03*, pages 185–194, 2003.
- [16] B. Srivastava, T.A. Nguyen, A. Gerevini, S. Kambhampati, M. B. Do, and I. Serina. Domain independent approaches for finding diverse plans. In *Proceedings of IJCAI’07*, pages 2016–2022, 2007.
- [17] Bradley S. Stewart and Chelsea C. White, III. Multiobjective A*. *Journal of the ACM*, 38(4):775–814, 1991.
- [18] M. Van den Briel, R.S. Nigenda, M. B. Do, and S. Kambhampati. Effective approaches for partial satisfaction (over-subscription) planning. In *Proceedings of AAAI’04*, pages 562–569, 2004.
- [19] C. L. Wu, K. W. Chau, and Y. S. Li. Predicting monthly streamflow using data-driven models coupled with data-preprocessing techniques. *Water Resources Research*, 450:W08432, August 2009.
- [20] J. Wu and S. Azram. Metrics for quality assessment of a multiobjective design optimization solution set. *Journal of Mechanical Design*, 123:18–25, 2001.
- [21] Jing-Xin Xie, Chun-Tian Cheng, Kwok-Wing Chau, and Yong-Zhen Pei. A hybrid adaptive time-delay neural network model for multi-step-ahead prediction of sunspot activity. *International Journal of Environment and Pollution*, pages 364–381, 2006.
- [22] H.L.S. Younes and M.L. Littman. PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical report, CMU-CS-04-167, Carnegie Mellon University, 2004.
- [23] Jun Zhang and Kwok-Wing Chau. Multilayer ensemble pruning via novel multi-sub-swarm particle swarm optimization. *Journal of Universal Computer Science*, 15(4):840–858, feb 2009.