The Australian National University
2600 ACT | Canberra | Australia

Australian National University

School of Computing

College of Engineering and Computer Science (CECS)

MOLAO*: heuristic search for multi-objective stochastic shortest path problems

— 6 pt research project (S1 2022)

By:
Dillon Chen

Supervisors:
Felipe Trevizan
Syvlie Thiebaux

May 2022

Declaration:

I declare that this work:

- upholds the principles of academic integrity, as defined in the University Academic Misconduct Rules;
- is original, except where collaboration (for example group work) has been authorised in writing by the course convener in the class summary and/or Wattle site;
- is produced for the purposes of this assessment task and has not been submitted for assessment in any other context, except where authorised in writing by the course convener;
- gives appropriate acknowledgement of the ideas, scholarship and intellectual property of others insofar as these have been used;
- in no part involves copying, cheating, collusion, fabrication, plagiarism or recycling.

July, Dillon Chen

Abstract

Planning problems are hard, being PSPACE-complete in its simplest form. Nevertheless, there exists various methods for reasoning and solving planning problems. One of these methods is heuristic search, a powerful solving technique which originated with the A* algorithm and has evolved to integrate deep learning methods to solve problems efficiently. We extend these heuristic search algorithms to substantially harder planning settings where both stochasiticity and multiple objectives are involved to better model real world problems. To the best of our knowledge our (i)MOLAO* algorithms are the first heuristic search algorithms for solving multi-objective probabilistic planning problems efficiently.

Table of Contents

Chapter 1

Introduction

Although planning problems in their simplest form are PSPACE-complete, there exist various methods to solve them efficiently, one of which is heuristic search for example the A* algorithm. Various powerful domain-independent heuristics [13, 5, 14] have been developed over the several decades to aid search and more recently methods which integrate deep learning models such as convolutional neural networks [26] and graph neural networks [25].

Heuristic search algorithms have also been extended to deal with probabilistic actions such as LAO* [12] and LRTDP [6], and multiple objectives with NAMOA* [17]. However, there are no heuristic search algorithms which combine *both* probabilistic actions and multiple objectives. One may argue that there is no need for considering both probabilistic actions and multiple objectives this given that one could construct an additional objective from risk associated with probabilities. However, the issue with deterministic multi-objective problems is that solutions for these problems consists of only linear plans and may not be able to deal with the contingencies which probabilistic solutions are able to.

There do however exist other algorithms for solving multi-objective stochastic shortest path problems (MOSSPs) with origins starting from multi-objective value iteration by White in 1982 [30] and various other value iteration methods [31, 1, 24], but these have trouble scaling up to MOSSPs with large state spaces, such as those modelled by planning problems. Our main contribution in this work consists of the (i)MOLAO* algorithms which leverage heuristic search techniques to more efficiently solve MOSSPs. We both theoretically and empirically verify its completeness, correctness and efficiency.

Chapter 2

Background

In this section we provide all the necessary preliminaries and background required to understand multi-objective stochastic shortest path problems and our proposed algorithm for solving such problems. The section begins with a formal set of definitions for MOSSPs followed by some well known theoretical results, an interlude into propositional planning and how we can use it to encode MOSSPs compactly, and ending with fundamental algorithms for solving SSPs and MOSSPs. Most of the content in this section is discussed in more detail by Mausam and Kolobov [18] and Roijers and Whiteson [23].

2.1 Definitions

We begin by providing a formal set of definitions for a multi-objective stochastic shortest path problem and what solutions for such problems look like. For ease of readability, we first begin with the single-objective case and inject multi-objectiveness where required to get the more general version.

2.1.1 Single-objective case

To begin, we have a stochastic shortest path problem which may be seen as a case of Markov decision processes (MDPs) with rewards replaced by costs and a specified stopping criteria via goal states.

Definition 2.1.1. A *stochastic shortest path problem* (SSP) consists of a tuple $P = (S, A, s_I, G)$ where

- S is a set of states.
- A is a set over probabilistic actions where a probabilistic action maps states to probability distributions of states. More formally this means that each $a \in A$ can be described as a function $a : S|_a \to \mathbb{P}(S)$ where $\mathbb{P}$ denotes a probability

distribution of states, and $S|_a \subseteq S$ denotes a subset of states which a is applicable to. This means that we have $a : S \times S \to \mathbb{R}_+$ where $a(s, s')$ represents the probability of progressing s to s' with action a, and a satisfies $\sum_{s' \in S} a(s, s') = 1$ for all $s \in S$. Alternatively we can define a transition function $P : S \times A \times S \to \mathbb{R}_+$ where $P(s, a, s')$ denotes the probability of action a progressing s to state s'. Furthermore, we have a cost function $C : S \times A \times S \to \mathbb{R}$ where $C(s, a, s')$ describes the cost of applying action a to state s with successor s'.

- $s_I \in S$ is an initial state.
- $G \subseteq S$ is a set of goal states.

When we define a problem, we would like to define what a solution is. This is easy for standard shortest path problems where a solution consists of a sequence of applicable actions that takes us from the initial state to the goal and is optimal with respect to the sum of the action costs. For SSPs we need to account for stochastic actions and its multiple effects which may result in cycles in our solution graph. So instead of a sequence of actions we need instead use a policy.

Definition 2.1.2. A *(stochastic) policy* for an SSP maps states to probability distributions over actions. A *deterministic policy* maps states to actions. So a stochastic policy has the form $\pi : S \to \mathbb{P}(A)$ whereas a deterministic policy has the form $\pi : S \to A$. For a stochastic policy we can equivalently write $\pi : S \times A \to \mathbb{R}_+$ where π satisfies $\sum_{a \in A} \pi(s, a) = 1$ for all $s \in S$.

For simplicitly, we also require that one is able to eventually reach the goal by following a policy from any state, i.e. there are no *dead ends*. There exist policies in other settings where we may only sometimes reach the goal due to the existence of dead ends but for the sake of this study here we do not need to consider them.

One may see that a deterministic policy is a special case of a general stochastic policy but wonder if we really need the idea of applying actions to states stochastically. The answer is no for the single-objective case but this answer is a bit more involved for the multi-objective case. In order to see what this means, we first how to define a method to measure the cost of a policy and hence define a solution for a SSP.

Definition 2.1.3. A *value function* $V^\pi : S \to \mathbb{R}_+$ associated with a policy π specifies the expected cost of following the policy π from state s to any goal $g \in G$, so

$$V^\pi(s) = \begin{cases} 0, & \text{if } s \in G \\ \mathbb{E}\left[\sum_{t=0}^{\infty} C(s_t, a_t, s_{t+1}) \mid \pi, s\right], & \text{otherwise.} \end{cases} \tag{2.1}$$

This definition may appear quite uninformative as one may question how we could calculate or express V^π in a nicer form. We will see that this is possible to approximate computationally and it has an alternative form given by the *Bellman equation* for all

$s \notin G$:

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P(s,a,s')[C(s,a,s') + V^{\pi}(s')]. \tag{2.2}$$

For deterministic policies one may simplify this equation by removing the outer sum and setting $a = \pi(s)$. Given a value function, we can measure the cost of a policy by simply looking at the expected cost from the initial state. Thus, we can define a solution an SSP as follows.

Definition 2.1.4. A *solution* for an SSP is an optimal policy π^* such that for all valid policies π we have $V^{\pi^*}(s_0) \leq V^{\pi}(s_0)$.

We conclude this introduction on single-objective SSPs by a theorem clearing up the question we had earlier concerning stochastic and deterministic policies.

Theorem 2.1.5 (The optimality principle [21])**.** *Given any SSP, an optimal policy exists which is deterministic. Furthermore, any optimal policy π^* satisfies*

$$\pi^*(s) = \arg\max_{a \in A} \left[\sum_{s'} P(s,a,s')[C(s,a,s') + V^*(s')] \right]. \tag{2.3}$$

where $V^*(s) = V^{\pi^*}(s)$ is the optimal value function.

The main takeaway here is that for SSPs we only need to focus on deterministic policies, policies which maps each state to a single (unique) action. Furthermore, the Bellman equation has a unique solution and thus all optimal policies share the same value function. Lastly we make clear that optimal policies may not necessarily be unique, for example by considering zero cost actions which do not do anything.

2.1.2 Multi-objective case

To generalise to the multi-objective case, we need to inject multidimensional cost vectors to above definitions where possible. Thus, defining MOSSPs is easy.

Definition 2.1.6. A *multi-objective stochastic shortest path problem* (MOSSP) consists of a tuple $P = (S, A, s_I, G)$ where

- S is a set of state.
- A is a set of probabilistic actions where a probabilistic action maps states to probability distributions of states. Furthermore, we have a cost function $\mathbf{C} : S \times A \times S \to \mathbb{R}^n_+$ where $\mathbf{C}(s,a,s')$ describes the cost of applying action a to state s with successor s'.
- $s_I \in S$ is an initial state.
- $G \subseteq S$ is a set of goal states.

The definition of a policy for an MOSSP is no different than that for an SSP given that costs are not involved here.

Definition 2.1.7. A *policy* for an MOSSP maps states to probability distributions of actions. A *deterministic policy* maps states to actions.

The difficulty with MOSSPs lie in generalising a value function and notion of solution, given that multidimensional costs or vectors have a different notion of ordering. Before we define the value function for a multi-objective policy, we require some notation for multi-dimensional costs. First, we need a method to compare vectors and values (sets of vectors). Contrary to scalars, not all vectors and values are comparable. We are instead limited to a partial order $\preceq$.

Definition 2.1.8. A vector u *dominates* another vector v denoted by $u \preceq v$ if $u_i \leq v_i$ for $i = 1, \ldots, n$. We have that u *strictly dominates* v denoted by $u \prec v$ if $u \preceq v$ and $u \neq v$.

To help us better compare values, we quickly define a pruning operator on values in the form of a coverage set .

Definition 2.1.9. A *coverage set* for a value $\mathbf{V}$ denoted $\mathrm{CS}(\mathbf{V})$ is any set satisfying

$$\forall v \in \mathrm{CS}(\mathbf{V}), \nexists u \in \mathrm{CS}(\mathbf{V}), u \prec v. \tag{2.4}$$

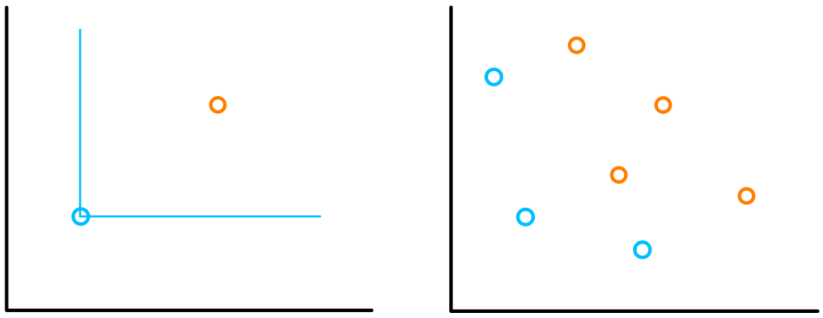
We note that there exists various methods for defining CS. One example is the *Pareto coverage set* PCS, where $\mathrm{PCS}(\mathbf{V})$ is the largest subset of $\mathbf{V}$ satisfying Eq. (2.4). Another example is the convex coverage set which will be discussed in more detail later in Sec. 2.3.3.

We are now able to define a partial order on values with a fixed choice for a coverage set operator.

Definition 2.1.10. A value $\mathbf{U}$ *dominates* another value $\mathbf{V}$ denoted by $\mathbf{U} \preceq \mathbf{V}$ if for all $v \in \mathbf{V}$, there exists $u \in \mathbf{V}$ such that $u \preceq v$. We have that $\mathbf{U}$ *strictly dominates (w.r.t.* CS*)* $\mathbf{V}$ denoted by $\mathbf{U} \prec \mathbf{V}$ if $\mathbf{U} \preceq \mathbf{V}$ and $\mathrm{CS}(\mathbf{U}) \neq \mathrm{CS}(\mathbf{V})$.

Fig. 2.1a illustrates an example of a dominating vector. Further note that any vector lying in the quadrant defined by the blue lines is dominated by the blue vector. One can equivalently define vector domination with generalised inequalities with respect to the cone $\mathbb{R}_+$. Fig. 2.1b illustrates a dominating value. The CS operator can be seen as an equivalence relation to quotient out pairs of values which may have the same coverage set but different number of interior vectors as we only care about the optimal vectors of a value.

Now we have all the tools to define a value function and a solution for multi-objective problems where again we assume we fix a choice of a coverage set operator.



(a) The blue vector dominates the orange vector.

(b) The blue value dominates the orange value.

Figure 2.1: Dominating vectors and values.

Definition 2.1.11. A *multi-objective value function* $\mathbf{V}^\pi : S \to \mathcal{P}(\mathbb{R}^n)$ associated with a policy specifies the expected cost of following the policy π from state s, where $\mathcal{P}$ denotes the power set. In other words, values of states are now sets of vectors.

Similarly to Eq. (2.2) we have an expression for $\mathbf{V}$ with the multi-objective Bellman equation:

$$\mathbf{V}^\pi(s) = \mathrm{CS}\left(\bigoplus_a \pi(s,a) \bigoplus_{s'} P(s,a,s')[\mathbf{C}(s,a,s') \oplus \mathbf{V}^\pi(s')]\right) \tag{2.5}$$

where $\oplus$ denotes elementwise addition between sets of vectors[1], $\bigoplus$ a generalised version of $\oplus$ for several elements and CS stands for coverage set and is an operator for pruning away dominated vectors of an input set of vectors.

For example if $\mathbf{V} = \{[0,1],[1,0],[1,1]\}$ and $\mathbf{U} = \{[-1,0],[0,-1]\}$, then $\mathbf{V} \oplus \mathbf{U} = \{[-1,1],[0,0],[1,-1],[0,1],[1,0]\}$ and $\mathrm{CS}(\mathbf{V} \oplus \mathbf{U}) = \{[-1,1],[0,0],[1,-1]\}$ for any choice of CS.

Now that we have defined a measure of cost of policies with a generalised value function and a partial order on values, we can easily proceed to define a solution for an MOSSP.

Definition 2.1.12. A *solution* to an MOSSP is a set of policies $\Pi = \{\pi : S \to \mathbb{P}(A)\}$ such that for all policies $\pi \in \Pi$, there does not exist any other valid policy π' such that $\mathbf{V}^{\pi'}(s_0) \prec \mathbf{V}^\pi(s_0)$.

2.2 Propositional planning

Before we start solving MOSSPs, we enter a small interlude into propositional planning for compactly encoding MOSSPs. There are many reasons why we should consider planning:

[1] We implicitly replace the vector $\mathbf{C}(s,a,s')$ with the singleton set containing itself.

- Planning has powerful formalisms for encoding search problems as it is able to implicitly store exponentially many states which may be infeasible to pass in as input naively to a solver.
- Planning languages allow us to compute domain-independant heuristics for all sorts of MOSSPs. This means that we do not need to handcraft heuristics for solving any sort of problems, although this may still be done if desired.
- There exists a unified framework for encoding search problems as planning problems (with multi-objective and stochastic support) with the Planning Domain Definition Language (PDDL).

One can rewrite an MOSSP as a propositional planning problem.

Definition 2.2.1. A (multi-objective stochastic) *propositional planning problem* consists of a tuple $P = (F, A, s_I, g)$ where

- F is a set of propositions or facts. A state is defined as a subset of propositions.
- A is a set of probabilistic actions. Actions take the form $a = (\text{pre}(a), \text{eff}(a))$ with precondition $\text{pre}(a) \subseteq F$ and probabilistic effects $\text{eff}(a) = \{(\text{add}_1(a), \text{del}_1(a), p_1), \ldots, (\text{add}_m(a), \text{del}_m(a), p_m)\}$ where $\text{add}_i(a), \text{del}_i(a) \subseteq F$ denote add and delete effects, and p_i denote probabilities of applying the ith effect $(\text{add}_i(a), \text{del}_i(a))$ and satisfies $p_i > 0, \sum_i p_i = 1$. As defined with MOSSPs earlier, each action has an associated cost function.
- $s_I \subseteq F$ is an initial state.
- $g \subseteq F$ is a goal condition. One says a state s is a goal state if $s \supseteq g$.

An action is applicable at a state $s \subseteq F$ if $\text{pre}(a) \subseteq s$. In this case, its successor states are given by $s_i = (s \setminus \text{del}_i(a)) \cup \text{add}_i(a)$ with probability p_i.

At this point, one should hopefully be able to see the connection between the general MOSSP definition and a planning problem. To go from a propositional planning problem to an MOSSP, we have that states are given by $S = 2^F$, the powerset of all propositions, and goal states by $G = \{s \in S \mid s \supseteq g\}$. Actions are compiled by using the rules defined above on actions being applicable at a given state and its probabilistic successors.

We conclude this section on some facts on the complexity of single-objective planning. Due to being able to encode exponentially many states with respect to its input, planning naturally becomes a hard problem. Proofs of the following theorems arise from reductions from (alternating) Turing machines.

Theorem 2.2.2 (Complexity of deterministic planning [9])**.** *Deterministic propositional planning is* PSPACE*-complete.*

Theorem 2.2.3 (Complexity of stochastic planning [16, 22])**.** *Stochastic propositional planning is* EXPTIME*-complete.*

Completeness for multi-objective problems are not as clear. However, it is easy to see that multi-objective stochastic propositional planning is in EXPSPACE and EXPTIME-hard.

2.3 Fundamental algorithms

Contrary to deterministic path finding, simple graph search algorithms are insufficient for finding solutions for (MO)SSPs due to the existance of stochastic actions and loops in the solution graph of policies. However, we recall that solutions satisfy the Bellman equations in Eq. (2.2) and Eq. (2.5) for single and multi-objective problems respectively. Although an analytic solution for these equations do not exist, we may approximate it using iterative methods.

2.3.1 Value Iteration

Let us define a set of recursive or iterative equations for approximating Eq. (2.2), known as *Bellman backups*, for $s \notin G$ by

$$V_n(s) \leftarrow \min_{a \in A} Q_n(s, a) \tag{2.6}$$

$$Q_n(s, a) \leftarrow \sum_{s' \in S} P(s, a, s') \left[C(s, a, s') + V_{n-1}(s') \right] \tag{2.7}$$

where we may initialise $V_0(s)$ to be any value, for example zero. For $s \in G$, we instead set $\mathbf{V}_n(s) = 0$. This method was originally proposed by Bellman alongside the Bellman equations [2] and is was originally coined as dynamic programming. The main idea is to update the current value of a state by greedily choosing the action which minimises the corresponding Q-value which is an expectation of the cost for choosing action a at state s.

It is known that with this iterative method, we have convergence to the optimal value function described in the optimality principle from Thm. 2.1.5. This is described more formally in the following theorem.

Theorem 2.3.1. *Given any initialisation V_0, we have $\lim_{n\to\infty} V_n(s) = V^*(s)$ for all $s \in S$ for VI.*

This gives us our first algorithm for approximating value functions for SSPs. The main idea is to iteratively update an approximate value function on all states of the problem until changes in V become minimal. This algorithm is called Value Iteration (VI) and is given in Alg. 1.

We see in lines 3-7 that Bellman backups are run over all non-goal states. In line 8, we calculate the change in value since the previous backup was performed. This value is known as a *residual*. Line 9 updates the new value function and line 10 checks the convergence criteria: the maximum residual over all states is less than a specified ϵ.

Algorithm 1: VI

Data: SSP problem $P = (S, A, s_I, G)$, initial values V for each state (default to $V[s] = 0$, $\forall s \in S$), and consistency threshold ε.

Result: ε-consistent value function

```
converged ← false
while not converged do
    for s ∈ S do
        if s ∈ G then
            V_new[s] ← 0
        else
            V_new[s] ← min_{a∈A} Σ_{s'∈S} P(s,a,s') [C(s,a,s') + V(s')]
        residual(s) ← |V_new(s) − V(s)|
    V ← V_new
    converged ← (max_{s∈S} residual(s) < ε)
return V
```

To extract a policy from the output value function V and its corresponding Q-function, we define the greedy policy $\pi(s) = \arg\min_{a \in A} \sum_{s' \in S} Q(s, a)$.

We say that Value Iteration is run to ε-*consistency* if the stopping criteria is defined as above. At this point one may wonder how accurate is our value function approximation with VI to the true value function. We have some loose bounds for converged value functions [3].

Theorem 2.3.2. *Let Value Iteration be run to ε-consistency with output value function V. Let $N^*(s)$ and $N^\pi(s)$ denote the expected number of steps to reach a goal $g \in G$ from s by following the optimal policy and the corresponding greedy policy π respectively. Then V satisfies the following inequality: $\forall s \in S, |V(s) - V^*(s)| < \varepsilon \cdot \max\{N^*(s), N^\pi(s)\}$.*

Although we have a bound on optimality, the expected steps $N^*(s)$ and N^π are difficult to compute as this requires solving another SSP [4]. Nevertheless in practice most problems end up returning the optimal policy even with an approximate value function.

2.3.2 Multi-objective Value Iteration

One may wonder if value iteration can be generalised for multi-objective costs. As hinted by Eq. 2.5 and this subsection title, the answer is yes. Similarly to VI, one is able to approximate multi-objective value functions by the following set of iterative equations,

multi-objective Bellman backups, which were first introduced by White [30]:

$$\mathbf{V}_n(s) \leftarrow \text{CS}\left(\bigcup_{a \in A} \mathbf{Q}_n(s,a)\right) \tag{2.8}$$

$$\mathbf{Q}_n(s,a) \leftarrow \bigoplus_{s' \in S} P(s,a,s')\left[\mathbf{C}(s,a,s') \oplus \mathbf{V}_{n-1}(s')\right]. \tag{2.9}$$

As with the single-objective Bellman backups, if $s \in G$, we instead set $\mathbf{V}_n(s) = \{\mathbf{0}\}$.

The two main differences with the single-objective VI equations are (1) generalised sums on sets of vectors as we have seen before with $\oplus$ and (2) generalised $\max_{a \in A}$ by considering the union of all multi-objective $\mathbf{Q}$-values and taking its coverage set with a CS operator. Algorithms for explictly computing coverage sets are described later in Sec. 2.3.3.

Thus we extend Alg. 1 to get multi-objective Value Iteration (MOVI) in Alg. 2. Lines 3-7 operate similarly to VI by running multi-objective Bellman backups iteratively on all states. However, we also need to generalise the convergence criteria by defining a metric on values. Given $\mathbf{U}$ and $\mathbf{V}$, we define its distance $D(\mathbf{U},\mathbf{V}) = \max_{u \in \mathbf{U}} \min_{v \in \mathbf{V}} d(u,v)$ for some choice of metric d as used in line 8. This is known as the Hausdorrf distance between sets and one can easily check that this is indeed a metric. Similarly to VI, we indeed get convergence to the optimal coverage set at the limit [30].

Algorithm 2: MOVI

Data: MOSSP problem $P = (S, A, s_I, G)$, initial values $\mathbf{V}$ for each state (default to $\mathbf{V}[s] = \{\mathbf{0}\}$, $\forall s \in S$), and consistency criteria ε.
Result: ε-consistent value function

```
1  converged ← false
2  while not converged do
3      for s ∈ S do
4          if s ∈ G then
5              V_new[s] ← {0}
6          else
7              V_new[s] ← CS(⋃_{a∈A} ⊕_{s'∈S} P(s,a,s') [C(s,a,s') ⊕ V(s')])
8          residual(s) ← D(V_new[s], V[s])
9      V ← V_new
10     converged ← (max_{s∈S} residual(s) < ε)
11 return V
```

Given a value function $\mathbf{V}$ and its corresponding $\mathbf{Q}$-function we would also like to extract a solution: a set of policies. This is done by selecting a scalarising weight vector $\boldsymbol{w}$ and extracting an optimal deterministic policy for each weight by $\pi(s,\boldsymbol{w}) = \arg\min_{a \in A} \min_{\boldsymbol{q} \in \mathbf{Q}(s,a)} \boldsymbol{w} \cdot \boldsymbol{q}$.

From here we would like to comment that we can approach multi-objective value iteration in the opposite direction: by first fixing a scalarising vector $\boldsymbol{w}$ and solving the scalarsied single-objective value iteration problem where costs are compiled away with $c(s, a, s') = \mathbf{C}(s, a, s') \cdot \boldsymbol{w}$. Algorithms using this method of solving several scalarised MOSSPs are described as outer loop approaches by Roijers and Whiteson [23]. For the remainder of this report, will focus on inner loop approaches such as MOVI described above.

Algorithm 3: PPrune

Data: A set of cost vectors $\mathbf{V}$
Result: A Pareto coverage set of $\mathbf{V}$

```
1 V_ret ← ∅
2 while V ≠ ∅ do
3     v ← any element from V
4     for v' ∈ V do
5         if v' ≺ v then
6             v ← v'
7     V ← V \ {v' | v ⪯ v'}
8     V_ret ← V_ret ∪ {v}
9 return V_ret
```

2.3.3 Pruning operators

We end this section on pruning operators and their importance for MOSSPs. As mentioned above, we need a method to compute coverage sets of values, one of which is by computing the Pareto coverage of $\mathbf{V}$. Recalling the definition from earlier, this is the set of vectors $v \in \mathbf{V}$ such that there exists no other $u \in \mathbf{V}$ with $u \preceq v$. Alg. 3 provides a routine for computing Pareto coverage sets with complexity $O(|\mathbf{V}|\,|\text{PCS}|)$ where $|\text{PCS}|$ is the size of the Pareto coverage set.

However, we can do further pruning. To motivate what we are about to see, consider the following simple MOSSP $P = (S, A, s_I, G)$ with

- $S = \{s_0, g_1, g_2\}$
- $A = \{a_1, a_2\}$ where
  - $C(s_0, a_1, s_0) = C(s_0, a_1, g_1) = [1, 0]$,
  - $C(s_0, a_2, s_0) = C(s_0, a_2, g_2) = [0, 1]$, and
  - $P(s_0, a_1, s_0) = P(s_0, a_1, g_1) = P(s_0, a_2, s_0) = P(s_0, a_2, g_2) = 0.5$
- $s_I = s_0$
- $G = \{g_1, g_2\}$.

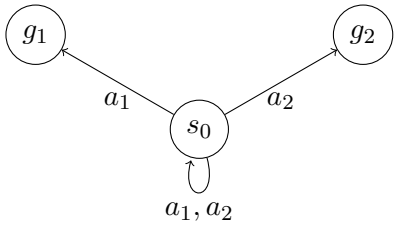


Figure 2.2: A simple MOSSP with a solution consisting of two deterministic policies.

Then it is easy to see that there exists a solution with only two deterministic policies $\pi_1(s_0) = a_1$ and $\pi_2(s_0) = a_2$ with corresponding expected costs $\{[2, 0]\}$ and $\{[0, 2]\}$. Thus a value function for this problem would be $\mathbf{V}(s_0) = \{[2, 0], [0, 2]\}$. However, let us look at how MOVI iteratively computes a value function if we choose PPrune as the coverage set operator. Denoting $\mathbf{V}_i$ to be the value function at s_I at the ith iteration, we have

$$\begin{aligned}
\mathbf{V}_0 &= \{[0, 0]\} \\
\mathbf{V}_1 &= \{[0.5, 0], [0, 0.5]\} \\
\mathbf{V}_2 &= \{[1.25, 0], [1, 0.25], [0.25, 1], [0, 1.25]\} \\
&\vdots \\
\lim_{n\to\infty} \mathbf{V}_n &= \{[2 - 2t, 2t] \mid t \in [0, 1]\}
\end{aligned}$$

We see that by using PPrune, the updated value functions begin to amass all points lying on the line segment between the two points corresponding to the expected costs of the expected policies. This becomes infeasible after several iterations of MOVI as the number of points increases exponentially due the nature of multi-objective Bellman backups implicitly computing stochastic policies.

In order to deal with this problem, we introduce a stronger coverage set operator with convex pruning to compute the *convex coverage set* CCS. In addition to computing the Pareto set, we only consider the vertices of the polytope drawn out by the Pareto set. Alg. 4 adapted from Feng and Zilberstein's pruning methods for POMDPs [10] describes this routine in more detail. Complexity is now given by $O(|\mathbf{V}|\,|\text{PCS}| + |\text{PCS}|\,P(|CCS|))$ where P is the (polynomial) complexity of solving a linear program in the size of the convex coverage set CCS.

Alg. 5 solves a linear program in order to determine in the dual space whether cost vectors are vertices in the convex polytope of PPrune($\mathbf{V}$). This is done by computing whether there exists a weight vector $\boldsymbol{w}$ such that the input vector $\boldsymbol{v}$ has a higher scalarised value $\boldsymbol{w} \cdot \boldsymbol{v}$ than any other vector in $\mathbf{V}$. The inequality constraint finds the best $\boldsymbol{w}$ with gap x, whereas the equality constraint is a normalising function using the assumption that cost functions are nonnegative.

Algorithm 4: CPrune

Data: A set of cost vectors $\mathbf{V}$
Result: A convex coverage set of $\mathbf{V}$

```
1  V ← PPrune(V)
2  V_ret ← ∅
3  while V ≠ ∅ do
4      v ← any element from V
5      w ← findWeight(v, V_ret)
6      if w is not null then
7          V ← V \ {v}
8      else
9          v ← arg min_{v'∈V} w · v'
10         V ← V \ {v}
11         V_ret ← V_ret ∪ {v}
12 return V_ret
```

Algorithm 5: findWeight

Data: A cost vector $\boldsymbol{v}$ and set of cost vectors $\mathbf{V}$
Result: A weight $\boldsymbol{w}$ where $\boldsymbol{v}$ has a higher scalarised value than any vector in $\mathbf{V}$

1 Solve the following LP

$$\begin{aligned} \max_{x,\boldsymbol{w}} \quad & x \\ \text{s.t.} \quad & \boldsymbol{w} \cdot (\boldsymbol{v} - \boldsymbol{v}') + x \leq 0, \quad \forall \boldsymbol{v}' \in \mathbf{V} \\ & \sum_{i=1}^{n} w_i = 1 \end{aligned}$$

2 **if** $x > 0$ **then return** $\boldsymbol{w}$;
3 **else return** *null* ;

Then returning to our previous example, denoting $\mathbf{V}_i^c$ to be the value function at s_I at the ith iteration where CS is computed using CPrune, we have

$$\begin{aligned}
\mathbf{V}_0^c &= \{[0,0]\} \\
\mathbf{V}_1^c &= \{[0.5,0],[0,0.5]\} \\
\mathbf{V}_2^c &= \{[1.25,0],[0,1.25]\} \\
&\vdots \\
\lim_{n\to\infty} \mathbf{V}_n^c &= \{[2,0],[0,2]\}
\end{aligned}$$

in comparison to simply using PPrune for CS where in the limit $\mathbf{V}_n$ would contain infinitely many elements as each iteration doubles the number of vectors in the value function.

Another viewpoint on why we only need to consider the polytope vertices of a value function is because such vertices correspond to deterministic policies and that we are able to implicitly infer the values for stochastic policies by walking along the facets of the polytope defined by such vertices.

Chapter 3

Heuristic search

In this chapter we will present the (i)MOLAO* algorithms, a generalisation of the (i)LAO* algorithms [12] which are used to solve SSPs. However, we will begin with an introduction to heuristics and multi-objective heuristics in order to present the algorithms.

3.1 Heuristics

Heuristics are used to guide search for solving search problems by providing solvers information on which states are worth looking at and which are not. Heuristic functions in the context of search problems and planning can be generally defined as a function estimating the expected cost from a given state to the goal. For definitions in this section, we provide both the single-objective and multi-objective variants.

Definition 3.1.1. An *heuristic* is a function which maps states to nonnegative values. In the context of SSPs, values are scalars, while in MOSSPs, values are sets of vectors. We usually denote heuristics for SSPs by $h(\cdot)$ and $\mathbf{H}(\cdot)$ for MOSSPs.

One may ask how we may use a heuristic for solving SSPs. Heuristics provide initial state values when any SSP solver encounters a state for the first time. For example, in MOVI we may use heuristics to kick start the algorithm where each state's value is initialised to be a heuristic $\mathbf{V}[s] = \mathbf{H}(s)$ instead of $\mathbf{V}[s] = \{\mathbf{0}\}$. We do note however, that $\mathbf{H}(s) = \{\mathbf{0}\}$ itself is a heuristic and is commonly referred to as the null or zero heuristic.

3.1.1 Admissible heuristics

However some heuristics are more useful than others. One class of heuristics to consider are admissible heuristics which are underestimations of the expected cost to the goal.

The definition for a heuristic for the single-objective case is as follows.

Definition 3.1.2. An *admissible heuristic* for an SSP is a function h satisfying $h(s) \leq V^*(s)$ for all $s \in S$ where $V^*(s)$ is the optimal expected cost of reaching a goal from state s.

We extend the definition of admissible multi-objective heuristics by Mandow and Pérez-de-la-Cruz [17] to deal with stochasticity.

Definition 3.1.3. An admissible heuristic for an MOSSP is a function H such that for every state s, every cost vector in $\mathbf{H}(s)$ does not . Using the language of value functions we can define this equivalently by

$$\forall s \in S, \quad \forall u \in \mathbf{V}^*(s), \quad \exists v \in \mathbf{H}(s) \quad v \preceq u \tag{3.1}$$

where $\mathbf{V}^*(s)$ is the optimal value at s. We implicitly assume that $\mathbf{H}(s)$ has already been pruned with some coverage set operator.

One reason for considering admissible heuristics is clear for deterministic settings such as single or multi-objective path finding: the A* algorithm for single-objective path finding and NAMOA* [17] for multi-objective path finding with an admissible heuristic returns optimal solutions. One may ask how this transfers over to stochastic problems and the answer is that admissible heuristics are also required for returning optimal solutions in heuristic search algorithms for (MO)SSPs as in Thm. 3.2.1. Note however, that admissibility of heuristics is *not* required for returning optimal solutions in (MO)VI as seen in Thm. 2.1.5.

A method of constructing admissible heuristic for SSPs, is to take any admissible heuristic for the *determinised* problem of an SSP. To convert an SSP into a deterministic problem, one takes the same set of states, initial state and goal state, and creates n new deterministic actions for each stochastic action in the original SSP, one corresponding for each stochastic effect. Note however that generally solutions are not preserved during this translation. This method of constructing heuristics indeed yields admissible heuristics given that we are solving a simpler problem and hence an admissible heuristic for the easier problem is an admissible heuristic for the original problem. For more detailed examples of heuristics for deterministic problems, or more specifically for deterministic planning problems one may refer to [5]. More informative heuristics that better take account into interactions between probabilistic actions exist [28, 29].

One method of constructing admissible heuristics for MOSSPs is to consider single-objective admissible heuristics on each objective dimension and concatenate them to get a multi-objective heuristic containing a single vector. This natural method of constructing multi-objective heuristics from single-objective heuristics is known the *ideal point* heuristic. However, this method treats each objective dimension orthogonally and ignores any interactions between objectives. A recent work by Geisser et al. [11] constructs more informative heuristics for (deterministic) multi-objective problems beyond the ideal point heuristic.

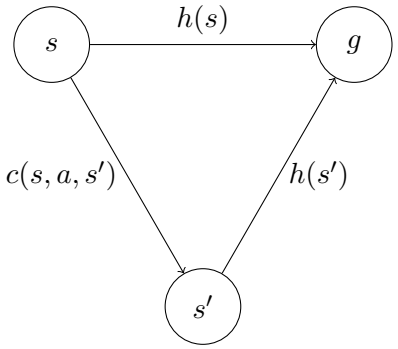


Figure 3.1: A consistent heuristic satisfies the triangle inequality.

3.1.2 Consistent heuristics

Another useful property of heuristics we may want to consider is consistency. Informally, consistency of heuristics may be likened to the triangle inequality.

Definition 3.1.4. A heuristic h for an SSP is *consistent* [12] if for all $s \in S, a \in A$, we have

$$h(s) \leq \sum_{s' \in S} P(s, a, s')[C(s, a, s') + h(s')]. \tag{3.2}$$

A heuristic H for a MOSSP is *consistent* if for all $s \in S, a \in S$, we have

$$\mathbf{H}(s) \preceq \bigoplus_{s' \in S} P(s, a, s')[\mathbf{C}(s, a, s') \oplus \mathbf{H}(s')]. \tag{3.3}$$

The definition of consistent heuristic is derived and generalised from the definition of consistent heuristics for deterministic path finding: $h(s) \leq c(s, a, s') + h(s')$. The main idea is that assuming nonnegative costs, state costs increase monotonically and which results in no reexpansions of nodes in A* search. Generalising in the multi-objective deterministic direction, we have consistency defined by $\mathbf{H}(s) \preceq \mathbf{C}(s, a, s') \oplus \mathbf{H}(s')$ [17]. As seen above, to generalise to the stochastic setting, one takes expectations over probabilistic action effects.

Note further that consistency implies admissibility. Another advantage of consistency in the MOSSP setting is that Bellman backups are ensured to increase monotonically. On the other hand, it may be the case that inconsistent heuristics result in nonmonotonic increase of value functions, resulting in unnecessary Bellman backups. An example of a (deterministic) multi-objective heuristic that is not consistent is the *anti-dominance maximum* [11] of two admissible heuristics H_1, H_2 denoted by $\text{admax}(H_1, H_2)$. It is one natural generalisation of the maximum operator for vectors defined by a union followed by pruning: $\text{admax}(H_1, H_2) = \text{CS}(H_1 \cup H_2)$.

However, the following example illustrates its nonconsistency (taken from an earlier version of [11]). Consider a problem a pair of states s, t and a deterministic transition between them with cost $[0, 1]$. Further consider the heuristic values in Table 3.1. We have that H_1, H_2 are consistent. However, $[0, 1] \oplus \text{admax}(H_1(t), H_2(t))$ is not, since $[3, 7]$ is in it but the vector is not dominated by the only vector in $[1, 10] \in \text{admax}(H_1(s), H_2(s))$. Thus, $\mathbf{H}(s) \not\preceq [0, 1] \oplus \mathbf{H}(t)$ where H is the admax heuristic.

Table 3.1: The admax heuristic is inconsistent.

	$\mathbf{H}(s)$	$\mathbf{H}(t)$	$[0, 1] \oplus \mathbf{H}(t)$
H_1	$\{[1, 5]\}$	$\{[3, 6]\}$	$\{[3, 7]\}$
H_2	$\{[1, 10]\}$	$\{[2, 11]\}$	$\{[2, 12]\}$
$\text{admax}(H_1, H_2)$	$\{[1, 10]\}$	$\{[2, 11], [3, 6]\}$	$\{[2, 12], [3, 7]\}$

To deal with inconsistent heuristics, we can make modifications to the Bellman backup operator to ensure monotonically increasing value functions again with a *pathmax* operator. This was first introduced [19] for the A* algorithm and extended for LAO*'s Bellman backups [12] by

$$V(s) \leftarrow \max\left(V(s), \min_{a \in A}\left(\sum_{s' \in S} P(s, a, s')\Big[C(s, a, s') + V(s')\Big]\right)\right). \tag{3.4}$$

We can generalise pathmax to the multi-objective setting using generalised max again with

$$\mathbf{V}(s) \leftarrow \text{CS}\left(\mathbf{V}(s) \cup \text{CS}\left(\bigoplus_{s' \in S} P(s, a, s')\Big[\mathbf{C}(s, a, s') \oplus \mathbf{V}(s')\Big]\right)\right). \tag{3.5}$$

Note that we may discard the inner pruning operation CS and get the same result, but this is used for optimisation purposes.

Thus, we see that the optimal heuristic is the one that returns the true value function and the weakest (admissible) heuristic we can have is the null heuristic. Intuitively, the closer our heuristic functions are to the true value function, the less time is required for Bellman backups to converge and more promising states are expanded during search.

3.1.3 Heuristic accuracy

We also require a method for measuring the usefulness of a heuristic. We know that admissible heuristics guarantee finding optimal solutions and consistency guarantees unnecessary reexpansions or Bellman backups. But the null heuristic satisfies both these attributes and one may wonder how can we show that we can do better and how can we measure if we can do better. One method of measuring heuristic quality is by measuring how close a heuristic is to approximating the true value function. We have the following theorem from Nilsson [20] about (admissible) heuristics for A* search.

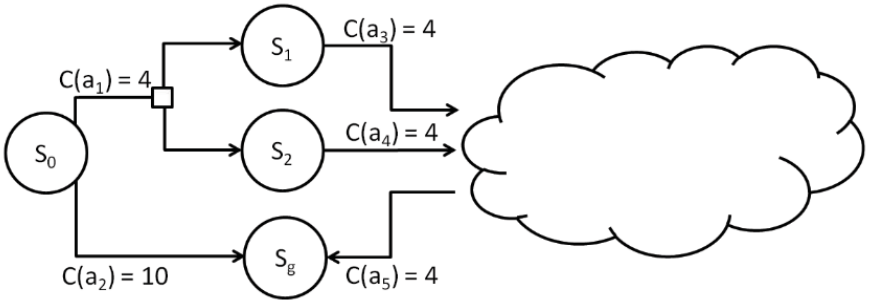


Figure 3.2: An SSP for which VI performs poorly [18]. The cloud abstractly denotes any number of states.

Theorem 3.1.5. *Suppose two heuristic functions h_1 and h_2 satisfy $h_1(s) \leq h_2(s) \leq V^*(s)$ for all states s. Then the set of states expanded by A^* with heuristic h_2 is a subset of the set of states expanded by A^* with heuristic h_2.*

This generalises for stochastic single-objective heuristics [12] where LAO* runs each VI step to exact convergence.

Theorem 3.1.6. *Suppose two heuristic functions h_1 and h_2 satisfy $h_1(s) \leq h_2(s) \leq V^*(s)$ for all states s. Then the worst-case set of states expanded by LAO* with heuristic h_2 is a subset of the worst-case set of states expanded by LAO* with heuristic h_2.*

In other words, the worse case of LAO* with a more accurate heuristic is more efficient. In the general case, this is not necessarily true and this is further complicated by the fact that LAO* generally runs VI to ε-consistency instead of absolute convergence. Nevertheless in practice, accurate heuristics on average perform more efficiently. One may generalise Thm. 3.1.6 for multi-objective heuristics by replacing $\leq$ with $\preceq$ for MOLAO*.

3.2 MOLAO*

In this section we will present the MOLAO* algorithm, a heuristic search algorithm for solving MOSSPs. To motivate this section, let us examine the major flaw of (MO)VI: it may run many unnecessary Bellman backups on states that do not contribute to a solution. Fig. 3.2 illustrates this idea more explicitly. We see that no matter what number of states exist in the cloud, a solution for the SSP simply applied a_2 at the initial state s_0 for a total expected cost of 10. If we choose a_1, then the expected cost is at least 12. However, VI would still run Bellman backups on every state of the problem despite every other state in the cloud having no impact on the optimal solution. This is where the power of heuristic search comes in.

Search algorithms generally build up a partial solution during its execution, gradually extending the partial solution until the goal state is reached by expanding unseen nodes in a frontier. Heuristics speed up search by intelligently choosing promising states to

expand and build upon for our solution. One version of heuristic search for MOSSPs is given in Alg. 6 which generalises the LAO* algorithm [12] for solving SSPs. The main idea is that instead of running Bellman backups on all possible states in the state space, we run VI to convergence on a subset of the state space corresponding to partial solutions. Each time VI is run to convergence, we check whether the goal condition is reached and if not, we extend and reconstruct our partial solution by choosing new states to add based on our heuristic function.

Algorithm 6: MOLAO*

Data: MOSSP problem $P = (S, A, s_I, G)$, heuristic $\mathbf{H}$, and consistency criteria ε
Result: ε-consistent value function on states contributing to the solution graph

1 $\mathbf{V} \leftarrow \mathbf{H}$ // lazy initialisation
2 $\Pi \leftarrow \emptyset$ // partial solution
3 $F \leftarrow \{s_I\}$ // frontier states
4 $I \leftarrow \emptyset$ // interior states
5 $N \leftarrow \{s_I\}$ // solution graph states
6 **while** $(F \cap N) \setminus G \neq \emptyset$ **do**
7 $\quad s \leftarrow$ any element from $(F \cap N) \setminus G$
8 $\quad F \leftarrow F \setminus \{s\}$
9 $\quad I \leftarrow I \cup \{s\}$
10 $\quad F \leftarrow F \cup (\text{successors}(s) \setminus I)$
11 $\quad Z \leftarrow \text{ancestorStates}(s, \Pi)$
12 $\quad \mathbf{V}|_Z \leftarrow \text{MOVI}(P|_Z, \mathbf{V}|_Z, \varepsilon)$
13 $\quad$ **for** $s \in Z$ **do**
14 $\quad\quad \Pi[s] \leftarrow \text{getActions}(s, \mathbf{V}[s])$
15 $\quad N \leftarrow \text{solutionGraph}(s_I, \Pi)$
16 **return V**

The algorithm begins by lazily assigning an initial value function $\mathbf{V}$ to each state with the heuristic function in Line 1, as opposed to explicitly intiailising all initial values at once which is unnecessary and infeasible if we have a large state space as in planning problems. Line 2 initialises a dictionary Π which maps states to sets of optimal actions corresponding to their current value function. This is our partial solution from which we can extract policies with the choice of a scalarising weight $\boldsymbol{w}$. Lines 3 and 4 initialises the frontier and set interior of states for search, and line 5 intialises the set of nodes N corresponding to our partial solution. The remainder of the algorithm consists of a loop with the condition in Line 6 stating that we have achieved a set of *closed policies* that is the policies that reach the goal when there are no nongoal states in the frontier.

Line 7 chooses a nongoal state from the frontier representing a state that is on the boundary of our partial solution graph but not a goal. We remove it from the frontier and add it to the interior state (lines 8 and 9). From here in line 10, we add the set of unobserved successors of s to the frontier, i.e. any state s' such that $P(s, a, s') > 0$

for some $a \in A$. Next, we extract all states that can reach s by following the partial solution Π with any simple graph search algorithm and call it Z in line 11. From here in line 12, we run MOVI to ε-consistency on the MOSSP problem restricted to the set of states Z and update the value functions for the corresponding states. Lines 13 and 14 update the partial solution by extracting the set of actions corresponding to the value function by

$$\text{getActions}(s, \mathbf{V}[s]) = \begin{cases} \emptyset, & \text{if } s \in G \\ \{a \in A \mid \mathbf{Q}(s,a) \cap \mathbf{V}[s] \neq \emptyset\}, & \text{otherwise.} \end{cases} \tag{3.6}$$

In other words, getActions returns a set of actions for which the Q-value of the state action pair contributes to the value function. This generalises the arg min function in the single-objective case, where an action is selected based on whether it has the lowest Q-value. Line 15 extracts the nodes corresponding to the solution graph denoted N. The solution graph consists of all states reachable from s_I by the partial solution Π and again can be computed by a simple graph search.

We extend the proof by Hansen and Zilberstein [12] for LAO* of completeness and correctness of the algorithm to MOLAO*.

Theorem 3.2.1. *If an admissible heuristic is used for MOLAO*, then*

1. $\mathbf{V}[s] \preceq \mathbf{V}^*[s]$ *for every state s at any time during execution of the algorithm, and*
2. $\mathbf{V}[s]$ *converges to within ε of $\mathbf{V}^*[s]$ with respect to the Hausdorrf distance for every state in the solution graph in a finite number of iterations.*

Proof. 1. This is proven by induction. By admissibility we have $\mathbf{V}[s] = \mathbf{H}(s) \preceq \mathbf{V}^*[s]$ as the initial value function at each state. Now assuming the inductive hypothesis that $\mathbf{V}[s] \preceq \mathbf{V}^*[s]$ for every state, a Bellman backup from Eq. (2.8) and (2.9) is bounded above by

$$\begin{aligned} \mathbf{V}_{new}[s] &= \text{CS}\left(\bigcup_a \bigoplus_{s'} P(s,a,s')\Big[\mathbf{C}(s,a,s') \oplus \mathbf{V}[s]\Big]\right) \\ &\preceq \text{CS}\left(\bigcup_a \bigoplus_{s'} P(s,a,s')\Big[\mathbf{C}(s,a,s') \oplus \mathbf{V}^*[s]\Big]\right) \\ &= \mathbf{V}^*[s] \end{aligned}$$

with the second equality holding from the Bellman equation.

2. The state space has finitely many states so eventually MOLAO* will find a solution graph with no nonterminal tip states in finitely many iterations. MOVI is performed on the solution graph to ε-consistency, completing the proof. □

3.3 iMOLAO*

One potential issue with MOLAO* is that we may waste a lot of Bellman backups while running MOVI to convergence several times on partial solution graphs which do not end up contributing to the final solution. This can be seen as an issue of overexploitation in the exploitation vs exploration viewpoint of decision making problems. The original authors of LAO* thus proposed an alternate algorithm iLAO* which better balances exploration and exploitation. We similarly provide an alternative algorithm iMOLAO* in the same vein. The main idea with the 'i' version of the algorithms is that we only run a set of Bellman backups every time we (re)expand a state instead of running VI to convergence which may take a lot more Bellman backups. This is described in more detail in Alg. 7.

Algorithm 7: iMOLAO*

Data: MOSSP problem $P = (S, A, s_I, G)$, heuristic H, and consistency criteria ε
Result: ε-consistent value function on states contributing to the solution graph

1 $\mathbf{V} \leftarrow \mathbf{H}$ // lazy initialisation
2 $\Pi \leftarrow \emptyset$ // partial solution
3 *converged* $\leftarrow$ *false*
4 **while** *not converged* **do**
5 $N \leftarrow \text{postorderTraversalDFS}(s_I, \Pi)$
6 **for** $s \in N$ *in the computed order* **do**
7 $\mathbf{V}_{new}[s] \leftarrow \text{BellmanBackup}(s)$ // Eq. (2.8)
8 $\Pi[s] = \text{getActions}(s, \mathbf{V}_{new}[s])$ // Eq. (3.6)
9 $\mathit{residual}(s) \leftarrow D(\mathbf{V}_{new}[s], \mathbf{V}[s])$
10 $\mathbf{V}[s] \leftarrow \mathbf{V}_{new}[s]$
11 *converged* $\leftarrow (N \cap G \neq \emptyset) \wedge (\max_{s \in N} \mathit{residual}(s) < \varepsilon)$
12 **return V**

Lines 1-2 initialises the same data structures as MOLAO* but without the need for a frontier and interior set. Lines 4 to 11 contains the main loop. In line 5, we collect the set of nodes corresponding to the current partial solution graph using DFS postorder traversal. Then in lines 6-10, we run Bellman backups in the postorder traversal order, with the idea that we want to run Bellman backups backwards from the border of the solution graph. We note in Line 10 that value functions are not updated synchronously, in other words we do not wait until all Bellman backups are run before the value functions are updated. This more dynamic method of updating values combined with the postorder traversal order results in slightly faster converging times. Line 11 describes our convergence criteria: (1) the solution graph envelops the goal, meaning that all states in the solution graph is able to reach the goal with the partial solution Π, and (2) the change in value function is minimal.

One may introduce an additional parameter k to provide further choice of exploration

vs exploitation tradeoff by determining how many times we want to run Bellman backups in the inner loop 6-10. Thus iMOLAO* presented about would have the default parameter $k = 1$. However, this may introduce an additional optimisation problem on what parameter k we would like to choose and would likely be domain and problem dependent.

Chapter 4

Experiments

Here we present experiments showcasing the power of heuristic search for MOSSPs. Given that, to the best of our knowledge, there does not exist any MOSSP solvers which scale up to planning problems with large state spaces we will provide the first baseline for MOSSP solvers. We omit experimental results for MOVI methods given that these require enumerating the whole state space of search problems and do not fit in memory when we consider propositional planning problems.

4.1 Setup

We describe the benchmarks we use and the algorithm configurations used for experiments here. Each of the benchmarks are MOSSP extensions of existing planning benchmarks from previous studies and the International Planning Competitions.

4.1.1 Benchmarks

k-d Exploding Blocksworld

Exploding Blocksworld was first introduced by Younes and Littman [32] as part of PPDDL, an extension of PDDL to include probabilistic effects and later slightly modified for the IPPC'08 [7]. The domain extends the original Blocksworld where a robot has to pick up and stack blocks on top of each other to get a target tower configuration. Exploding Blocksworld extends this by introducing a probability of blocks detonating and destroying blocks underneath it or the table. We consider the version which removes unavoidable deadends by introducing an action to repair the table [27]. Additional problems were later constructed by Geisser et al. [11].

We extend Exploding Blocksworld one more step to contain multi-objective costs. One variant we consider is 2-d exploding Blocksworld in which there are two objectives:

to minimise the number of actions required to stack the blocks, and to minimise the number of times we need to repair the table. We also consider another variant with three objectives known as 3-d exploding Blocksworld. Here we introduce an action to repair blocks and an additional objective to minimise the number of times we need to repair blocks. Here, we want the planner to consider tradeoffs between repairing blocks, tables and plan length. This is an attempt to model real life problems where we have to consider risk and differing costs of repairing tables and blocks.

One may also generalise even further to k-d exploding Blocksworld where we introduce an additional cost and action for each individual block with the idea that some blocks may be more expensive to repair than other blocks and we may not know this a priori to solving the problem. However, we do not consider this problem here as this may be too difficult to solve.

MO Triangle Tireworld

Triangle tireworld was introduced by Little and Thiebaux [15] consisting of a triangular grid of locations. The goal is to travel from an initial location to the goal where each location has a probability of getting a flat tire. However, some locations contain a spare tire with which you can replace your tire. The problem has dead ends when you get a flat tire in a location without a spare.

Similarly to k-d Exploding Blocksworld, we introduce an additional action where you can order a spare tire to your current location if you get a flat tire. In turn we also introduce an additional cost dimension with the number of tires ordered. There exists a simple analytical solution to this problem but it is neverthless a challenging problem for SSP solvers.

MO Search and Rescue

Search and Rescue was first introduced by Trevizan et al. [27] as a constrained SSP. The goal is to find, board and escort to a safe location any one survivor in an $n \times n$ grid as quickly as possible and constrained to a fuel limit. Probabilities are introduced by modelling fuel consumption and partial observability of whether a survivor exists in a given location. However, the location of one survivor is known for certain.

We extend the problem to involve multiple objectives by considering fuel consumption as an additional objective instead of a constraint. Thus, a solution for the Search and Rescue MOSSP is a set of policies with different tradeoffs between fuel and time.

4.1.2 Solver configurations

We consider a simple combination of solvers and heuristics. We test the two MOSSP solvers MOLAO* and iMOLAO* with three admissible multi-objective heuristics on the determinised problems as discussed in Section 3.1: the null heuristic h_{null}, the ideal point

Table 4.1: Number of problems solved. Timeout of 30 minutes applied to all problems.

Algorithm	Heuristic	Tri. Tire.	SAR-4	SAR-5	exbw-2d	exbw-3d
MOLAO*	null	2	275	224	2	2
	max	3	275	184	25	20
	admax	3	**332**	246	5	5
iMOLAO*	null	3	163	160	5	5
	max	**4**	302	**289**	**29**	**23**
	admax	**4**	317	269	9	8

max heuristic h_{max}, and the admax heuristic h_{admax}[1].

Each solver configuration is given a timeout of 30 minutes for each problem. The consistency criteria ε is set to 10^{-3}. Furthermore, we modify the inequality constraint in Alg. 5 to

$$\boldsymbol{w} \cdot (\boldsymbol{v} - \boldsymbol{v}') + x \leq -\lambda, \quad \forall \boldsymbol{v}' \in \mathbf{V} \tag{4.1}$$

with $\lambda = 10^{-2}$. This is to deal with inevitable numerical precision errors when solving the LP in findWeight. If the additional λ term is not added, the function may fail to prune some points away in the convex coverage set, resulting in unncessary points in value functions and slower convergence.

4.2 Results

We collate the results for number of problems solved with different solver configurations in Table 4.1. We note that almost always both solvers perform better with heuristics than without heuristics, being able to solve more problems. We also note that generally iMOLAO* is able to solve more problems by better balancing exploration vs exploitation. This comes with an exception with the Search and Rescue problems with no heuristic which may be attributed to the fact that exploitation is more significant for blind search in such problems.

We further provide statistics on the mean and standard deviation of number of Bellman backups performed and states expanded on all solver configurations in Tables 4.2 and 4.3. In order to provide a fair comparison, we only collect such statistics on problems which all solvers were able to solve. Generally we expect a lower number of backups performed and states expanded for solver configurations which were able to solve more problems. For example, with the SAR problem, we notice that iMOLAO* with no heuristic expands significantly more nodes than MOLAO* with no heuristic.

[1] Geisser et al. [11] who came up with this heuristic also introduced an additional comax heuristic in the final version of the publication. However, this was only released very close to the deadline for this report, hence we have not considered it.

Table 4.2: Mean and standard deviation of number of Bellman backups performed on problems which all configurations solved.

Algorithm	Heur.	Tri. Tire.	SAR-4	SAR-5	exbw-2d	exbw-3d
MOLAO*	null	19662.5±19218.5	2512.0±4874.9	2260.2±2665.3	870.0±2.0	870.0±2.0
	max	2941.0±2820.0	2447.3±8647.7	3854.4±18125.1	17.0±0.0	17.0±0.0
	admax	2941.0±2820.0	933.4±2253.7	1164.3±2723.2	36.0±0.0	36.0±0.0
iMOLAO*	null	8552.0±8094.0	3731.0±4488.8	4253.4±3988.0	873.0±0.0	874.5±0.5
	max	2937.5±2742.5	378.4±908.3	283.0±420.6	23.0±0.0	23.0±0.0
	admax	2937.5±2742.5	264.5±713.2	183.0±240.2	42.0±0.0	42.0±0.0

Table 4.3: Mean and standard deviation of number of states expanded on problems which all configurations solved.

Algorithm	Heur.	Tri. Tire.	SAR-4	SAR-5	exbw-2d	exbw-3d
MOLAO*	null	261.0±235.0	57.2±48.4	64.9±41.2	37.0±0.0	37.0±0.0
	max	244.0±218.0	38.7±87.7	50.7±140.0	6.0±0.0	6.0±0.0
	admax	244.0±218.0	24.5±32.2	30.2±41.5	9.0±0.0	9.0±0.0
iMOLAO*	null	494.0±452.0	698.3±667.2	906.7±769.5	462.0±0.0	462.0±0.0
	max	413.0±371.0	58.2±111.7	50.7±55.1	10.0±0.0	10.0±0.0
	admax	413.0±371.0	41.2±95.4	31.9±29.7	17.0±0.0	17.0±0.0

Overall, we can conclude that heuristics indeed help speed up search by reducing the number of states expanded and Bellman backups performed. Generally iMOLAO* is more efficient than MOLAO* when heuristics are at play, but when no heuristics are considered the tradeoff between exploration vs exploitation becomes more significant depending some domains, as seen in results for SAR and Exploding Blocksworld with the null heuristic.

However, the work on experiments are still incomplete here. One could still leverage the multitude of existing admissible heuristics for special cases of MOSSP (SSP and deterministic MO search) from various works [28, 29, 11] and analyse their effectiveness on MOSSPs. Furthermore, we could perform additional ablation studies on the consistency criteria ε and the tolerance constant λ for the LP value function pruner.

Chapter 5

Conclusion

We have developed a novel heuristic search algorithm for solving multi-objective stochastic shortest path problems (MOSSPs) known as (i)MOLAO*. We prove that, similarly to its predecessors A* and LAO*, with admissible heuristics the algorithm terminates with optimal solutions. We further verify the effectiveness of heuristic search both theoretically and empirically, being able to solve planning problems with an exponential state space.

However, there is still some tasks left over for future work. There a whole range of other MOSSP heuristics we can define by either relaxing the problem and levering existing heuristics for MO planning or SSPs, or to create new heuristics which are better able to take into account stochasticity and multi-objectiveness. We might want to run additional ablation studies on some of the parameters of (i)MOLAO* such as on the effect of the consistency term ε and numerical stability tolerance λ. Furthermore, we may also consider a multi-objective variant of LRTDP, a heuristic search algorithm for SSPs which is generally incomparable with LAO* in terms of efficiency.

Bibliography

[1] L. Barrett and S. Narayanan. Learning all optimal policies with multiple criteria. In *ICML*, volume 307, pages 41–47. ACM, 2008. [Cited on page 1.]

[2] R. Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966. [Cited on page 9.]

[3] D. Bertsekas. *Dynamic programming and optimal control: Volume I*, volume 1. Athena scientific, 1995. [Cited on page 10.]

[4] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming*, volume 3 of *Optimization and neural computation series*. Athena scientific, 1996. [Cited on page 10.]

[5] B. Bonet and H. Geffner. Planning as heuristic search. *Artif. Intell.*, 129(1-2):5–33, 2001. [Cited on pages 1 and 18.]

[6] B. Bonet and H. Geffner. Labeled RTDP: improving the convergence of real-time dynamic programming. In *ICAPS*, pages 12–21, 2003. [Cited on page 1.]

[7] D. Bryce and O. Buffet. 6th Int. Planning Competition: Uncertainty Track. *3rd Int. Probabilistic Planning Competition*, 2008. [Cited on page 27.]

[8] D. Bryce, W. Cushing, and S. Kambhampati. Probabilistic planning is multi-objective. *ASU CSE TR*, 2007. [Cited on page 37.]

[9] T. Bylander. The computational complexity of propositional STRIPS planning. *Artif. Intell.*, 69(1-2):165–204, 1994. [Cited on page 8.]

[10] Z. Feng and S. Zilberstein. Region-based incremental pruning for POMDPs. In D. M. Chickering and J. Y. Halpern, editors, *Proceedings of the 20th Conference in Uncertainty in Artificial Intelligence UAI*, pages 146–153, 2004. [Cited on page 13.]

[11] F. Geisser, P. Haslum, S. Thiébaux, and F. Trevizan. Admissible Heuristics for Multi-Objective Planning. In *ICAPS*, 2022. [Cited on pages 18, 19, 20, 27, 29, and 30.]

[12] E. A. Hansen and S. Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artif. Intell.*, 129(1-2):35–62, 2001. [Cited on pages 1, 17, 19, 20, 21, 22, and 23.]

[13] P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In *AAAI*, pages 140–149, 2000. [Cited on page 1.]

[14] M. Helmert, P. Haslum, and J. Hoffmann. Flexible abstraction heuristics for optimal sequential planning. In *ICAPS*, pages 176–183, 2007. [Cited on page 1.]

[15] I. Little and S. Thiébaux. Probabilistic Planning vs Replanning. In *ICAPS Workshop International Planning Competition: Past, Present and Future*, 2007. [Cited on page 28.]

[16] M. L. Littman, J. Goldsmith, and M. Mundhenk. The computational complexity of probabilistic planning. *J. Artif. Intell. Res.*, 9:1–36, 1998. [Cited on page 8.]

[17] L. Mandow and J. Pérez-de-la-Cruz. Multiobjective a* search with consistent heuristics. *J. ACM*, 57(5):27:1–27:25, 2010. [Cited on pages 1, 18, and 19.]

[18] Mausam and A. Kolobov. *Planning with Markov Decision Processes: An AI Perspective*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012. [Cited on pages 3 and 21.]

[19] L. Mérõ. A heuristic search algorithm with modifiable estimate. *Artif. Intell.*, 23(1):13–27, 1984. [Cited on page 20.]

[20] N. J. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing, 1980. [Cited on page 20.]

[21] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 1994. [Cited on page 5.]

[22] J. Rintanen. Complexity of planning with partial observability. In S. Zilberstein, J. Koehler, and S. Koenig, editors, *ICAPS*, pages 345–354, 2004. [Cited on page 8.]

[23] D. M. Roijers and S. Whiteson. *Multi-Objective Decision Making*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2017. [Cited on pages 3 and 12.]

[24] E. Sedova, L. Mandow, and J. Pérez-de-la-Cruz. Asynchronous vector iteration in multi-objective markov decision processes. In *19th Conference of the Spanish Association for Artificial Intelligence*, volume 12882, pages 129–138, 2021. [Cited on page 1.]

[25] W. Shen, F. W. Trevizan, and S. Thiébaux. Learning domain-independent planning heuristics with hypergraph networks. In *ICAPS*, pages 574–584, 2020. [Cited on page 1.]

[26] S. Toyer, S. Thiébaux, F. W. Trevizan, and L. Xie. Asnets: Deep learning for generalised planning. *J. Artif. Intell. Res.*, 68:1–68, 2020. [Cited on page 1.]

[27] F. Trevizan, S. Thiébaux, P. Santana, and B. Williams. Heuristic Search in Dual Space for Constrained Stochastic Shortest Path Problems. In *ICAPS*, 2016. [Cited on pages 27 and 28.]

[28] F. W. Trevizan, S. Thiébaux, and P. Haslum. Occupation measure heuristics for probabilistic planning. In *ICAPS*, pages 306–315, 2017. [Cited on pages 18 and 30.]

[29] F. W. Trevizan, S. Thiébaux, and P. Haslum. Operator counting heuristics for probabilistic planning. In *IJCAI*, pages 5384–5388, 2018. [Cited on pages 18 and 30.]

[30] D. White. Multi-objective infinite-horizon discounted markov decision processes. *Journal of Mathematical Analysis and Applications*, 89(2):639–647, 1982. [Cited on pages 1 and 11.]

[31] M. A. Wiering and E. D. de Jong. Computing optimal stationary policies for multi-objective markov decision processes. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 158–165, 2007. [Cited on page 1.]

[32] H. L. Younes and M. L. Littman. PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. *Technical Report CMU-CS-04-162*, 2004. [Cited on page 27.]

Appendix A

Additional note on related work

This work was done under the assumption that (i)MOLAO* was the first heuristic search algorithm for solving MOSSPs. However, we very recently found out a day before the deadline for the report that a previous work exists by Bryce et al. [8] which also created MOLAO*. The work was only submitted as a technical report and was never published, maybe due to the controversy surrounding probabilistic planning around that time period. We decided not to mention it in the main body of this report in this final day for the following reasons:

- The work is incomplete and missing a lot of details for MOLAO*: a lot of equations, especially how we may want to consider multi-objective Bellman backups, and formalisms are omitted or defined vaguely.
- The original MOLAO* is on the whole similar to our version of MOLAO* but with many subtle differences, some due to missing formalisation and vague interpretations of definitions.
- Alongside missing definitions, there are also missing theorems and proofs concerning the algorithm such as convergence properties, completeness and correctness.