On Computing Plans with Uniform Action Costs

Alberto Pozanco, Daniel Borrajo*, Manuela Veloso

J.P. Morgan AI Research
{alberto.pozancolancho,daniel.borrajo,manuela.veloso}@jpmorgan.com

Abstract

In many planning applications, agents might be interested in finding plans whose actions have costs that are as uniform as possible. Such plans provide agents with a sense of stability and predictability, which are key features when humans are the agents executing plans suggested by planning tools. This paper adapts three uniformity metrics to automated planning, and introduces planning-based compilations that allow us to lexicographically optimize sum of action costs and action costs uniformity. Experimental results in different benchmarks show that the reformulated tasks can be effectively solved in practice to generate uniform plans.

Introduction

Classical planning is the task of finding a plan, which is a sequence of deterministic actions such that, when applied in a given initial state, it results in a goal state (Ghallab, Nau, and Traverso 2004). Each action is associated with a non-negative cost, and the cost of a plan is defined as the sum of its actions' costs. Plans with minimal cost are called optimal, and how to efficiently compute them accounts for a large part of automated planning research.

However, the real-world is full of applications where the sum of action costs is only one of the objectives that define an optimal plan (Stewart and White III 1991; Geißer et al. 2022; Salzman et al. 2023). For example, in network routing one might be interested in finding widest shortest paths or shortest widest paths, in order to reduce bottlenecks and provide better service (Sobrinho 2001). Or, in navigation domains, one can represent travel time as cost and be interested in finding shortest quickest or quickest shortest routes (Golledge 1995).

In this paper we introduce a novel objective that might be relevant in many planning applications: that of finding plans whose actions have costs that are as uniform as possible. Consider the navigation scenario shown in Figure 1, where a car driver (located at position `1_0`) is interested in finding a plan to reach location G (located at `0_4`). To do so, the car can move in the four cardinal directions. The color of each cell depicts the cost (travel time) of traversing the cell: visiting green, yellow, orange and red cells have a cost of 1, 2, 3, and 4 respectively. In this case, there exist several cost-optimal plans, all having a cost of 9, but requiring to traverse a different number of congested areas, with each of them having an associated congestion level. Arguably, some drivers would prefer plans where they reduce the number or density of the traffic jams, thus being able to drive at the same pace. Other drivers might even prefer longer travel times sacrificing cost-optimality in benefit of longer routes where they completely avoid traffic jams. Moreover, not only drivers but also traffic authorities would prefer vehicles to keep a constant velocity, as this reduces congestions and CO_2 emissions (Barth and Boriboonsomsin 2009).

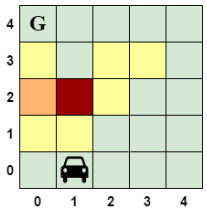


Figure 1: Navigation task where a driver wants to reach location G. The color of each cell depicts the cost (travel time) of traversing the cell.

Financial planning (Pozanco et al. 2023) is yet another example where computing plans with uniform actions is relevant. In financial planning, a person wants to save some amount of money (e.g. $\$1,000$) in a given horizon (e.g. 4 months). The available actions are to save different quantities at each month, with increasing costs associated with higher savings levels due to the difficulty in saving more money. People could prefer plans where they can save the same money every month (e.g. $\$250$ each month) rather than plans where they should save very different amounts. This preference can be justified by several reasons. Firstly, a consistent savings plan provides individuals with a sense of stability and predictability. Knowing exactly how much they need to save each month allows them to budget and plan their expenses more effectively. This stability can alleviate financial stress and provide a sense of control over one's financial situation. Secondly, a fixed savings plan promotes discipline and consistency in saving habits. By com-

*On leave from Universidad Carlos III de Madrid

Copyright © 2024, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

mitting to saving a specific amount each month, individuals are more likely to develop a routine and stick to their financial goals (Hanna and Lindamood 2010).

We are interested in lexicographically optimizing both objectives (in any order): sum of action costs and action costs uniformity. There exist three main approaches to solve such multi-objective planning problems in the literature. The first one is using cost-algebraic A* (Edelkamp, Jabbar, and Lluch-Lafuente 2005), which proves that A* returns an optimal solution in any *cost algebra* (or their prioritized Cartesian product), not just in the traditional shortest path setting. A cost measure is a cost algebra iff it is *strictly isotone*, i.e., a solution that optimizes the cost measure is composed of optimal solutions to its subproblems. Unfortunately, most uniformity measures do not satisfy this property, so we cannot use standard A*. The second approach is to use multi-objective search algorithms such as NAMOA* (Mandow and De La Cruz 2008) or BOA* (Ulloa et al. 2020; Hernández et al. 2023). These algorithms extend A* to compute the set of Pareto-optimal paths. However, these algorithms are slower and consume more memory than standard A*, since they (i) need to explore the whole state space to return all the solutions in the Pareto front; and (ii) need to perform extra checks to prune dominated paths. The third approach consists of reformulating the original planning task so that plans that optimally solve the new task are plans that lexicographically optimize the two objectives. This approach allows us to use any planner to solve such problems without the need of crafting domain-dependent heuristics for each objective. For example, Katz, Röger, and Helmert (2022) transform the action costs of the original planning tasks in order to generate shortest cost-optimal plans. Here, we will reformulate the whole planning task, and not only its action costs, to generate uniform plans.

The main contributions of this paper are:

1. Introduction of a novel bi-objective planning task with many real-world applications: finding cost-optimal plans whose actions have costs that are as uniform as possible.
2. Adaptation of three dispersion metrics to automated planning.
3. Three different compilations to produce plans that lexicographically optimize sum of action costs and the given dispersion metric.
4. New benchmarks for multi-objective planning (Salzman et al. 2023).

Background

We formally define a planning task as follows:

Definition 1. *A **planning task** can be defined as a tuple $\Pi = \langle F, A, I, G\rangle$, where F is a set of propositions, A is a set of instantiated actions, $I \subseteq F$ is an initial state, and $G \subseteq F$ is a goal state.*

For each proposition $p \in F$ we introduce the term $\neg p$ representing the negation of p. Propositions and their negations are called literals L. A (partial) state s is a consistent set of literals $l \subseteq L$, i.e., s does not contain a proposition and its negation. A complete state is a state which contains each proposition or its negation. Each action $a \in A$ is described by a set of preconditions (pre(a)), which represent literals that must be true in a state to execute an action, and a set of effects (eff(a)), which are the literals that are added (add(a) effects) or removed (del(a) effects) from the state after the action execution. The definition of each action might also include a cost $c(a) \in \mathbb{N}^0$ (the default cost is one). We denote by $c(A)$, $\min(c(A))$ and $\max(c(A))$ the set of action costs, minimum action cost, and maximum action cost in the planning task, respectively. The execution of an action a in a state s is defined by a function γ such that $\gamma(s, a) = (s \setminus \text{del}(a)) \cup \text{add}(a)$ iff $\text{pre}(a) \subseteq s$. The output of a planning task is a sequence of actions, called a plan, $\pi = (a_1, \ldots, a_n)$. The execution of a plan π in a state s can be defined as:

$$\Gamma(s, \pi) = \begin{cases} \Gamma(\gamma(s, a_1), (a_2, \ldots, a_n)) & \text{if } \pi \neq \emptyset \\ s & \text{if } \pi = \emptyset \end{cases}$$

A state s is reachable iff there exists a sequence of operators π that when applied from I reach s, i.e., $s \subseteq \Gamma(I, \pi)$. With S we refer to the set of all reachable states of the planning task. A plan π is valid if $G \subseteq \Gamma(I, \pi)$. The plan cost is commonly defined as $c(\pi) = \sum_{a_i \in \pi} c(a_i)$, with $\boldsymbol{c}_\pi = (c(a_i), \ldots, c(a_n))$ denoting the action cost vector of a plan. A plan with minimal cost is called optimal. In the rest of the paper we assume plans are simple, i.e., they do not have loops visiting the same state more than once. This is the case for any cost-optimal plan.

Uniformest Cost-Optimal Planning Problem

While classical planning focuses on finding cost-optimal plans, we seek plans that also optimize the uniformity of the plans' action costs. Like Katz, Röger, and Helmert (2022), we are not interested in any form of weighting both objectives, but in lexicographically optimizing them.

We seek to minimize two objectives , rather than minimizing plan's cost and maximizing its uniformity. Therefore, we will minimize action costs dispersion in order to find uniform plans. The *dispersion of a plan*, denoted by $d(\pi)$, is defined as the variability of its action costs. In other words, a plan only composed by actions of the same cost will have a lower dispersion, therefore being more uniform, than a plan composed by actions with different costs. We denote by $\leq_d$ the partial order of plans defined by their action costs dispersion, i.e., $\pi \leq_d \pi'$ iff $d(\pi) \leq d(\pi')$. Likewise, we denote by $\leq_c$ the partial order of plans defined by their cost, i.e., $\pi \leq_c \pi'$ iff $c(\pi) \leq c(\pi')$. We will use $\preceq_{c,d}$ when we want to optimize cost, and break ties in favor of more uniform (less disperse) plans. Formally, $\pi \preceq_{c,d} \pi'$ if $c(\pi) < c(\pi')$ or if $c(\pi) = c(\pi')$ and $d(\pi) \leq d(\pi')$. The same notation and definition applies to the opposite case $\preceq_{d,c}$, where we want to optimize uniformity and break ties by cost.

Summarizing, for a given task Π in standard classical planning we must identify a plan that is minimal among all plans with respect to $\leq_c$, or detect that the task is unsolvable. On the other hand, in the uniformest cost-optimal planning problem, we must find a plan for Π that is minimal among

all plans with respect to $\preceq_{c,d}$ (or $\preceq_{d,c}$), or detect that the planning task is unsolvable.

Dispersion Metrics

Dispersion, also called variability, is a statistical term that refers to the spread between numbers in a data set. In this case, we are interested in measuring the dispersion of vectors of plans action costs. There exist many dispersion metrics: standard deviation, variance, range, or entropy, to name a few. In this paper we will focus on the following three measures of statistical dispersion:

Definition 2. ***Number of different action costs*** *of a plan π: the cardinality of the set of plan's action costs.*

$$\#(\pi) = |\{\boldsymbol{c}_\pi\}| \tag{1}$$

Definition 3. ***Delta*** *of a plan π: the largest difference between the cost of any two adjacent actions in the plan.*

$$\Delta(\pi) = \max\Big(\bigcup_{i=1}^{|\pi|-1} |c(a_i) - c(a_{i+1})|\Big) \tag{2}$$

Definition 4. ***Range*** *of a plan π: the difference between the largest and smallest values in the costs of its actions.*

$$R(\pi) = \max(\boldsymbol{c}_\pi) - \min(\boldsymbol{c}_\pi) \tag{3}$$

The reason why we select the above dispersion metrics over others is two-folded. First, we seek diverse set of dispersion metrics. Well-known statistical measures such as standard deviation or variance provide the same information, while the three selected metrics have different semantics. Range and Delta measure the difference between action costs in the plan, but while the former focuses on the entire plan, the latter only considers changes in two consecutive actions. On the other hand, just measuring the number of different action costs offers less information, but might be easier to compute while still a good dispersion proxy. We further investigate this in our evaluation.

Second, the above dispersion metrics have a nice property that other measures do not have: their potential values are bounded a priori by the available actions in the planning task. While we can know all the possible values for Range, Delta and the number of different action costs by just inspecting the planning task, this is not possible for metrics such as standard deviation or entropy because the number of actions needed to achieve a goal is unbounded. As we will discuss later, this is an important property that allows us to generate tractable reformulations in practice.

Let us exemplify these dispersion metrics and the plans we are interested in finding by using the navigation scenario introduced in Figure 1. Table 1 shows four possible plans to reach the goal in this scenario. Each plan is depicted by a sequence of squares (from left to right) that show the path of the driver from its initial state to the goal. Columns in this table show the three dispersion metrics, i.e., Range, Delta, and Number of different action costs, as well as the cost of the plan. The plans in the first three rows are cost-optimal and have an associated cost (travel time) of 9. However, these plans are very different in terms of action cost dispersion.

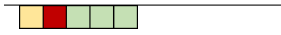
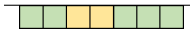
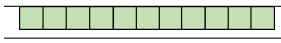
π	$\#(\pi)$	$\Delta(\pi)$	$R(\pi)$	$c(\pi)$
	3	3	3	9
	3	1	2	9
	2	1	1	9
	1	0	0	11

Table 1: Four alternative plans to reach G in the scenario shown in Figure 1. Plans are depicted by sequences of squares (from left to right) that show the driver's path from I to G. Columns show dispersion and cost measures and cells in the table indicate the metric obtained by each plan.

The first plan combines driving through extremely (red) and slightly (green) congested areas, yielding the higher (worse) values for all the dispersion metrics. This plan will not be to the liking of drivers who prefer to maintain a steady pace. The second plan has a lower Delta and Range, since the change in congestion levels occurs more progressively and extremely congested areas are not traversed. The third plan has the lowest values for all the dispersion metrics along the cost-optimal plans, and would be the one selected by drivers that prefer more uniform driving experience while not sacrificing cost (driving time). Finally, the fourth plan might be more appealing to drivers that might be willing to sacrifice cost-optimality in benefit of more uniform routes where they can completely avoid traffic jams. Among all the plans that minimize dispersion, this is the one with the lowest cost.

Computing Plans with Uniform Action Costs

We present three different compilations, one for each of the dispersion metrics previously described. Although some of the compilations could be represented more naturally using conditional effects or numerical planning, we opted for classical planning compilations, as the resulting tasks can be efficiently solved optimally by a wider variety of planners. In particular, unlike other compilations that take as input the lifted planning task (represented in PDDL (McDermott et al. 1998)) and return a new lifted planning task (Gragera et al. 2023), our reformulation operates at the grounded (STRIPS) level. We opted for this approach as in some planning tasks the cost of the actions is not known at the lifted level, but only after the task has been grounded.

The intuition behind all the compilations is to extend a standard planning task with new propositions that keep track of the action cost distribution. They also extend the original planning task with actions that either (i) update these propositions; or (ii) increase the total cost depending on the value of the dispersion metric induced by the plan. The cost of these actions will depend on the objective function that we try to lexicographically optimize. We will use $\omega_d \in \mathbb{N}$ to weigh the importance of getting uniform plans.

Next, we introduce the compilations in increasing order of complexity, according to the number of extra propositions and actions they require.

Compilation 1: Number of Different Action Costs

This is the most naive dispersion metric, as it only considers the cardinality of the plan's action cost set. It also requires the least complex compilation, as this metric is strictly isotone (Edelkamp, Jabbar, and Lluch-Lafuente 2005). Given a planning task Π, we extend F with a set of propositions $F_u = \bigcup_{i \in c(A)} \{\text{used}_i\}$, which keep track of the costs that have been used by a plan. We also update the set of original actions A to associate each action $a \in A$ a different cost depending on whether that action cost has already been used in the plan or not. We create two new actions for each original action $a \in A$, effectively dividing actions into two sets, A_u and $A_{\neg u}$.

- A_u, which represents the original actions that can be executed when their cost is already present in the plan. Each action $a_u \in A_u$ is defined as follows:
  - $\text{pre}(a_u) = \text{pre}(a) \cup \{\text{used}_{c(a)}\}$
  - $\text{eff}(a_u) = \text{eff}(a)$
  - $c(a_u) = c(a)$
- $A_{\neg u}$, represent the original actions that can be executed when their cost is not present in the plan yet. Each action $a_{\neg u} \in A_{\neg u}$ is defined as follows:
  - $\text{pre}(a_{\neg u}) = \text{pre}(a) \cup \{\neg\text{used}_{c(a)}\}$
  - $\text{eff}(a_{\neg u}) = \text{eff}(a) \cup \{\text{used}_{c(a)}\}$
  - $c(a_{\neg u}) = c(a) + \omega_d$

The difference between both sets of actions is that the ones in $A_{\neg u}$ set the unseen cost $c(a)$ to used, and increase the total cost with the weight ω_d, thus effectively penalizing the use of actions with different action costs to achieve the goal.

The new planning task is defined as follows:

Definition 5. *Given an original planning task Π, a **compiled task** $\Pi_\#$ that lexicographically **optimizes sum of action costs and number of different action costs** as dispersion metric is a tuple*

$$\Pi_\# = \langle F', A', I', G, \omega_d \rangle$$

where,

- $F' = F \cup F_u$
- $A' = A_u \cup A_{\neg u}$
- $I' = I \cup \bigcup_{i \in c(A)} \{\neg used_i\}$

This compilation grows polinomially with $|A|$ and generates $|A'| = 2 \times |A|$ actions, as we need to have two versions of each action, depending on whether its cost has already been used in the plan or not.

Proposition 1. *If Π is solvable, $\Pi_\#$ is also solvable.*

Proof Sketch. Let π be a solution for Π. It is trivial to see that there exists a plan π' that solves $\Pi_\#$ comprised by the same sequence of actions as π, only varying their version, i.e., if their cost has already been used in the plan (A_u) or not ($A_{\neg u}$). The applicability of these actions is equivalent to their A counterparts, with the extra precondition of the *used* proposition of that cost, which appear in I' and are updated by the actions in A'. □

Next we prove that $\preceq_{c,d}$ (or $\preceq_{d,c}$) is preserved in the reformulated task. The proof is similar to that of (Katz, Röger, and Helmert 2022), where the authors prove their reformulated task can generate shortest cost-optimal plans. The main idea is to set ω_d so that for any plan, the extra cost incurred by introducing new action costs does not surpass the sum of action costs (or the opposite in case $\preceq_{d,c}$).

Proposition 2. *Let π and π' be any two simple plans for Π, and $\pi_\#$ and $\pi'_\#$ their counterparts in $\Pi_\#$. There exists a value for ω_d such that $\pi \preceq_{c,d} \pi'$ iff $\pi_\# \leq_c \pi'_\#$.*

Proof Sketch. Since the plans are simple, the cost of any π that solves Π is bounded by $|S| \times \max(c(A))$. Let $C^- > 0$ be smaller or equal to the smallest non-zero cost difference $|c(\pi) - c(\pi')|$ between two plans that solve Π. Let D^+ be larger than the max different action costs difference $|\#(\pi) - \#(\pi')|$ between two plans that solve Π. We define $\omega_d := \frac{C^-}{D^+} > 0$.

We distinguish two cases. If $c(\pi) = c(\pi')$, then $\pi \preceq_{c,d} \pi'$ iff $\#(\pi) \leq \#(\pi')$. By our compilation we have that

$$c(\pi_\#) = c(\pi) + \omega_d \#(\pi) \leq c(\pi') + \omega_d \#(\pi') = c(\pi'_\#) \quad (4)$$

so when $c(\pi) = c(\pi')$ this can be simplified to $\#(\pi) \leq \#(\pi')$, iff $\pi_\# \leq_c \pi'_\#$.

On the other hand, if $\pi \preceq_{c,d} \pi'$ and $c(\pi) \neq c(\pi')$ then $c(\pi) < c(\pi')$. Since $c(\pi') - c(\pi)$ is non-zero, it must be $\geq C^-$. By the choice of D^+, $\#(\pi) - \#(\pi') < D^+$. We get that $c(\pi'_\#) - c(\pi_\#) = c(\pi') + \omega_d \#(\pi') - c(\pi) - \omega_d \#(\pi) = c(\pi') - c(\pi) - \omega_d(\#(\pi') - \#(\pi)) > c(\pi') - c(\pi) - \omega_d D^+ = c(\pi') - c(\pi) - C^- \geq 0$. Overall we get that $c(\pi'_\#) > c(\pi_\#)$, thus $\pi_\# \leq_c \pi'_\#$. In the opposite direction, if $\pi_\# \leq_c \pi'_\#$, then $c(\pi_\#) \leq c(\pi'_\#)$. By (4), $c(\pi) + \omega_d \#(\pi) \leq c(\pi') + \omega_d \#(\pi')$. Since $\#(\pi) - \#(\pi') < D^+$, it holds that $\omega_d(\#(\pi') - \#(\pi)) < \omega_d D^+ = C^-$. By the definition of C^-, $C^- > c(\pi) - c(\pi')$. This implies $c(\pi) < c(\pi')$ and thus $\pi \preceq_{c,d} \pi'$. □

As in the case of (Katz, Röger, and Helmert 2022), since we only focus on integer action costs ($C^- = 1$), it is sufficient for the proof if $|\#(\pi_\#) - \#(\pi'_\#)| < 1/\omega_d$. This guarantees that if we set $\omega_d < 1/D^+$, $\preceq_{c,d}$ is preserved.

Analogously it is easy to see that if we set $\omega_d > C^+$, where C^+ is the largest cost difference between two plans ($|S| \times \max(c(A))$), then $\pi \preceq_{d,c} \pi'$ iff $c(\pi_\#) \leq c(\pi'_\#)$.

Compilation 2: Delta

Delta measures the largest jump in action costs in two consecutive actions in a plan. Minimizing it can be useful for agents content with plans having disperse action costs, as long as the cost changes are gradual rather than sudden. This dispersion metric is not strictly isotone, so the compilation to optimize it becomes more complex.

In this compilation we will use $abs(c(A))$ (or just abs) to refer to the set of absolute values obtained after subtracting all the cost pairs in the domain, i.e., all the possible Delta values. These values are given by the following formula:

$$abs(c(A)) = \bigcup_{m,n \in c(A) \cup \{0\}} \{|m - n|\} \quad (5)$$

We cannot define a general formula based on the size of $c(A)$ to compute the size of abs, as it does not solely depend on the number of possible costs, but also on the arithmetic relationship of its elements. For example, $c(A) = \{0, 5, 10\}$ yields a set of possible Delta values $abs = \{0, 5, 10\}$, while $c(A) = \{1, 5, 10\}$ yields $abs = \{0, 1, 4, 5, 9, 10\}$. However, we can upper bound the size of this set by the binomial coefficient $\binom{|c(A)|+1}{2}$. In the worst case, all the pairs of the set will yield a different absolute value, plus 0, since that is the result of the subtraction of any pair (x, x).

Given a planning task Π, we extend F with the following sets of propositions:

- $F_{prev} = \bigcup_{i \in c(A)} \{\text{prev_cost}_i\}$, which is a set of propositions that keep track of the last executed action's cost.
- $F_\delta = \bigcup_{i \in abs} \{\text{delta}_i\}$, which is a set of propositions that keep track of the Delta between the current and the previous actions executed in the plan.
- $F_\Delta = \bigcup_{i \in abs} \{\text{max_delta}_i\}$, which is a set of propositions that track the largest Delta in the plan.
- $check$, which is a proposition used as a flag to indicate when the max_delta propositions should be checked and updated.
- end, which is a proposition that represents the fact that all goals have been achieved.

We also extend A with three types of actions. The first type A_δ updates the original actions to keep track of the prev_cost counter and the current value of Delta between the current and the previous action. Each action $a_\delta \in A_\delta$ is defined as follows:

- $\text{pre}(a_\delta) = \text{pre}(a) \cup \{\text{prev_cost}_i, \neg check\}$
- $\text{eff}(a_\delta) = \text{eff}(a) \cup \{\neg\text{prev_cost}_i, \text{prev_cost}_{c(a)}, \text{delta}_{|i-c(a)|}, check\}$
- $c(a_{prev}) = c(a)$

The second type of actions will appear interleaved in the plan between the updated original actions. For each pair of costs $i, j \in c(A)$, if $i > j$, we generate a new action in a new set A_Δ. Otherwise, we generate a new action in another new set $A_{\neg\Delta}$. Since we are operating over the grounded task, we can make the check when generating the compilation.

- A_Δ, which are new actions that update the counter when delta is larger than max_delta. Each action $a_\Delta \in A_\Delta$ is defined as follows:
  - $\text{pre}(a_\Delta) = \{check, \text{delta}_i, \text{max_delta}_j\}$
  - $\text{eff}(a_\Delta) = \{\neg check, \neg\text{delta}_i, \neg\text{max_delta}_j, \text{max_delta}_i\}$
  - $c(a_\Delta) = 0$
- $A_{\neg\Delta}$, which are new actions that do not update the counter when delta is lower or equal than max_delta, just turn off the $check$ flag, so A_δ actions can be executed again. Each action $a_{\neg\Delta} \in A_{\neg\Delta}$ is defined as follows:
  - $\text{pre}(a_{\neg\Delta}) = \{check, \text{delta}_i, \text{max_delta}_j\}$
  - $\text{eff}(a_{\neg\Delta}) = \{\neg check, \neg\text{delta}_i\}$
  - $c(a_{\neg\Delta}) = 0$

Finally, the third type of actions can only be executed at the end of planning, i.e., once a goal state has been reached. They increase the total cost of the plan depending on its Delta (Definition 3), i.e., the value of the max_delta proposition. We refer to this set of actions as A_{end}, and each action $a_{end} \in A_{end}$ is defined as follows:

- $\text{pre}(a_{end}) = G \cup \{\neg check, \text{max_delta}_i\}$
- $\text{eff}(a_{end}) = \{end\}$
- $c(a_{end}) = i \times \omega_d$

The new planning task is defined as follows:

Definition 6. *Given an original planning task Π, a **compiled task** Π_Δ that lexicographically **optimizes sum of action costs and Delta as dispersion metric** is a tuple*

$$\Pi_\Delta = \langle F', A', I', G', \omega_d \rangle$$

where,

- $F' = F \cup F_{prev} \cup F_\delta \cup F_\Delta \cup \{check, end\}$
- $A' = A_\delta \cup A_\Delta \cup A_{\neg\Delta} \cup A_{end}$
- $I' = I \cup \{\textit{prev_cost}_{\min(c(A))}, \textit{max_delta}_{\min(c(A))}\}$
- $G' = \{end\}$

This compilation generates a polinomial number of actions that is given by the following formula:

$$|A'| = \Big(\overbrace{|A| \times |c(A)|}^{A_\delta}\Big) + \overbrace{|abs|^2}^{A_\Delta, A_{\neg\Delta}} + \overbrace{|abs|}^{A_{end}}$$

The first part of the formula refers to the different possible values of prev_cost appearing in the precondition of the A_δ actions. The second part refers to the actions that update the max_delta propositions, i.e., one action for each combination of $\text{delta}_i, \text{max_delta}_j, \forall_{i,j \in abs}$. The third part refers to the potential values of max_delta at the end of planning. Following a similar reasoning as in Propositions 1 and 2, it is easy to see that this compilation is also complete, sound, and can preserve $\preceq_{c,d}$ or $\preceq_{d,c}$ if we set ω_d as a relation between the max/min Delta and cost differences between two plans.

Compilation 3: Range

Range is one of the most widely employed dispersion metrics, since it is easy to compute and is known to correctly approximate standard deviation σ under different distributions by applying the formula $\sigma \approx \frac{\text{Range}}{4}$ (Shiffler and Harsha 1980). This dispersion metric is again not strictly isotone.

Given a planning task Π, we extend F as follows:

- $F_{\min} = \bigcup_{i \in c(A)} \{\text{min_cost}_i\}$, which is a set of propositions that keep track of the cost of the least costly action executed in a plan.
- $F_{\max} = \bigcup_{i \in c(A)} \{\text{max_cost}_i\}$, which is a set of propositions that keep track of the cost of the most costly action executed in a plan.
- end, which is a proposition that represents the fact that all goals have been achieved.

We also extend the original set of actions with two types of actions. The first type updates the original actions to keep track of the min and max counters. We divide them into the following subsets of actions, which cover all possible ways in which the counters can be updated. Like in the Delta compilation, we can do this because we know all the action costs combinations beforehand.

- A_{both}, which are the original actions that update both counters when the cost of the action is (i) lower than the minimum, and (ii) greater than the maximum costs employed in the plan. Each action $a_{both} \in A_{both}$ is defined as follows:
  - $\text{pre}(a_{both}) = \text{pre}(a) \cup \{\text{min_cost}_i, \text{max_cost}_j\}$
  - $\text{eff}(a_{both}) = \text{eff}(a) \cup$
  $$\{\neg\text{min_cost}_i, \text{min_cost}_{c(a)}, \neg\text{max_cost}_j, \text{max_cost}_{c(a)}\}$$
  - $c(a_{both}) = c(a)$
- $A_{\min}$, which are the original actions that update the min_cost counter when the cost of the action is (i) lower than the minimum cost, and (ii) lower or equal than the maximum cost employed in the plan. Each action $a_{\min} \in A_{\min}$ is defined as follows:
  - $\text{pre}(a_{\min}) = \text{pre}(a) \cup \{\text{min_cost}_i, \text{max_cost}_j\}$
  - $\text{eff}(a_{\min}) = \text{eff}(a) \cup \{\neg\text{min_cost}_i, \text{min_cost}_{c(a)}\}$
  - $c(a_{\min}) = c(a)$
- $A_{\max}$, which are the original actions that update the max_cost counter when the cost of the action is (i) greater or equal than the minimum cost, and (ii) greater than the maximum cost employed in the plan. Each action $a_{\max} \in A_{\max}$ is defined as follows:
  - $\text{pre}(a_{\max}) = \text{pre}(a) \cup \{\text{min_cost}_i, \text{max_cost}_j\}$
  - $\text{eff}(a_{\max}) = \text{eff}(a) \cup \{\neg\text{max_cost}_j, \text{max_cost}_{c(a)}\}$
  - $c(a_{\max}) = c(a)$
- A_{none}, which are the original actions that do not update any counter when the cost of the action is (i) greater or equal than the minimum cost, and (ii) lower or equal than the maximum cost employed in the plan. Each action $a_{none} \in A_{none}$ is defined as follows:
  - $\text{pre}(a_{none}) = \text{pre}(a) \cup \{\text{min_cost}_i, \text{max_cost}_j\}$
  - $\text{eff}(a_{none}) = \text{eff}(a)$
  - $c(a_{none}) = c(a)$

The second type of actions can only be executed at the end of planning, i.e., once a goal state has been reached, and increase the total cost of the plan depending on its Range. We create one action for each possible value of abs (Equation 4). We refer to this set of actions as A_{end}, and each action $a_{end} \in A_{end}$ is defined as follows:

- $\text{pre}(a_{end}) = \text{G} \cup \{\text{min_cost}_i, \text{max_cost}_j\}$
- $\text{eff}(a_{end}) = \{end\}$
- $c(a_{end}) = (j - i) \times \omega_d$

The new planning task is defined as follows:

Definition 7. *Given an original planning task* Π*, a* ***compiled task*** Π_R *that lexicographically* ***optimizes sum of action costs and Range as dispersion metric*** *is a tuple*

$$\Pi_R = \langle F', A', I', G', \omega_d \rangle$$

where,

- $F' = F \cup F_{\min} \cup F_{\max} \cup \{end\}$
- $A' = A_{both} \cup A_{\min} \cup A_{\max} \cup A_{none} \cup A_{end}$
- $I' = I \cup \{\textit{min_cost}_{\max(c(A))}, \textit{max_cost}_{\min(c(A))}\}$
- $G' = \{end\}$

This compilation generates a polinomial number of actions that is given by the following formula:

$$|A'| = \overbrace{(|A| \times |c(A)|^2)}^{A_{both}, A_{\min}, A_{\max}, A_{none}} + \overbrace{|c(A)|^2}^{A_{end}}$$

The first part of the formula refers to the different variations of the original actions that we create for each possible value of the min_cost and max_cost counters. The second part refers to the actions that achieve the end goal proposition. The number of these actions is given by all the combinations of the min and max counters values. Following a similar reasoning as in Propositions 1 and 2, it is easy to see that this compilation is also complete, sound, and can preserve $\preceq_{c,d}$ or $\preceq_{d,c}$ if we set ω_d as a relation between the max/min Range and cost differences between two plans.

Evaluation

Experimental Setting

Benchmark. We analyzed the tasks in the Planning Domains repository[1], and picked those domains without conditional effects that have problems that, after grounding, satisfy $|c(A)| \geq 2$. In case some domains were duplicated (either from different years or tracks), we selected the problems of the latest optimal track. Our benchmark consists of 387 problems divided into 19 domains. 17 domains and 347 problems come from the Planning Domains repository. In this set, most of the problems only have 2 different action costs, with $> 50\%$ of the problems having below 6 different action costs. In order to have more challenging tasks for uniformest cost-optimal planning, we introduce FINANCE and NAVIGATION, the two novel domains we discussed in the introduction. They have problems with up to 41 and 101 different action costs, respectively.

Approaches. We evaluate the three compilations, namely $\Pi_{\#}, \Pi_{\Delta}$ and Π_R, on the above benchmark. Since most planners only accept integer costs, we perform a similar cost transformation as in (Katz, Röger, and Helmert 2022) and set the weights as follows. When solving $\preceq_{c,d}$, we set $c(a) = c(a) \times 10^6$ and $\omega_d = 1$. Since 10^6 is a loose upper bound on the maximum dispersion a plan can have in our benchmark, we keep the theoretical properties. When solving $\preceq_{d,c}$, we set $\omega_d = 2 \times 10^6$, which is again a loose upper bound on the cost of the simple plans in our benchmark. Given a compilation C, we will refer to these configurations as $\Pi_C^{c,d}$ or $\Pi_C^{d,c}$, respectively. We compare these $3 \times 2 = 6$ compilations against two *baselines*. The first one uses a standard planner solving task Π. We use this to understand the overhead incurred by the compilations compared to solving the simpler task where only cost is optimized. The second one computes all the optimal plans of the standard planning task, which we refer to as $\mathcal{P}(\Pi)$. We use this to understand if solving the compiled planning tasks is faster than computing $\mathcal{P}(\Pi)$ and then post-processing that set to find the most uniform plan. These two baselines can only be used to compare against

[1] https://github.com/AI-Planning/classical-domains

	Π	$\Pi^{d,c}_{\#}$	$\Pi^{c,d}_{\#}$	$\Pi^{d,c}_{\Delta}$	$\Pi^{c,d}_{\Delta}$	$\Pi^{d,c}_{R}$	$\Pi^{c,d}_{R}$	$\mathcal{P}(\Pi)$
Coverage	225	167	204	100	115	105	125	115

Table 2: Number of problems solved by each compilation.

the $\Pi^{c,d}$ compilations, since the $\Pi^{d,c}$ compilations are solving a different task. However, we put them together so we can understand (i) which task is easier; and (ii) how far solving $\Pi^{c,d}$ is from the optimal dispersion achieved by solving $\Pi^{d,c}$.

Reproducibility. We solve all the planning tasks ($\Pi, \Pi_{\#}, \Pi_{\Delta}$ and Π_R) and their different variations using the SEQ-OPT-LMCUT configuration of Fast Downward (Helmert 2006), which runs A^* with the admissible LMCUT heuristic to compute an optimal plan. For computing the set of optimal plans $\mathcal{P}(\Pi)$, we use FORBIDITERATIVE (Katz et al. 2018) with a limit of $100,000$ plans to avoid disk overflows. We chose this planner over other alternatives such as SYM-K (Speck, Mattmüller, and Nebel 2020) because its search process is more similar to that of SEQ-OPT-LMCUT, and therefore comparing their results is more fair. Experiments were run on an Intel Xeon E5-2666 v3 CPU @ 2.90GHz x 8 processors with a 8GB memory bound and a time limit of 1800s. Code and benchmarks are available upon request.

Results

Coverage and Execution Time Overhead. First, we want to understand how hard solving our compiled tasks is. Table 2 shows the coverage of each compilation and the two baselines across all domains and problems. As expected, solving the standard planning task where only cost is optimized is easier, and the planner can find a cost-optimal plan for 225 instances. Optimizing for cost and breaking ties in favor of less disperse plans ($\Pi^{c,d}$ compilations) tends to be easier than doing the opposite, regardless of the dispersion metric. Taking a look to the specific metrics, minimizing the number of different action costs ($\Pi_{\#}$ compilations) is somewhat comparable to standard planning, and definitely more feasible than computing all the cost-optimal plans. This is not the case when jointly optimizing cost and Delta or Range, where we get slightly lower coverages than when solving $\mathcal{P}(\Pi)$. This difference in coverage comes mainly from domains such as FINANCE or NAVIGATION, which are difficult for uniformest cost-optimal planning due to their large number of different action costs, but easier for top-quality planning, as they contain few cost-optimal plans. However, by post-processing all the cost-optimal plans we would only get those that optimize these metrics as a second objective, i.e., plans generated by the $\Pi^{c,d}$ compilations, and not those plans that prioritize uniformity ($\Pi^{d,c}$ compilations).

In order to further understand the overhead induced by our compilations, we compare the execution time T needed by the planner to solve the new tasks X versus the time needed to solve the standard planning task. We refer to this as the *time overhead factor*, and formally define it as $\frac{T(X)}{T(\Pi)}$. This execution time includes both the time needed to translate (and ground) the task and the solving time. To make the comparison fair, we only consider the 59 problems that are commonly solved by all the approaches. Figure 2 shows this analysis as a set of violinplots, which represent the distribution of these factors in log scale for each approach. As we can see, minimizing the number of different action costs hardly introduces any overhead compared to solving the standard planning task, resulting into easier tasks in few problems. On the other hand, the rest of the compilations are much closer in execution time to computing all the cost-optimal plans.

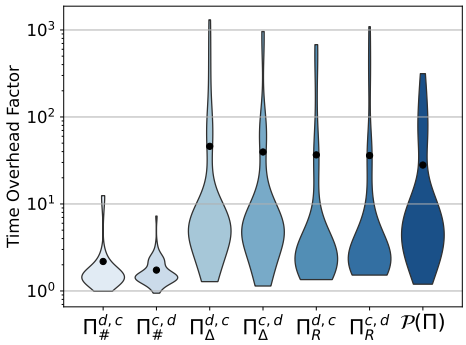


Figure 2: Distribution of the execution time overhead factor $\frac{T(X)}{T(\Pi)}$ for each approach X. Black dots represent the average.

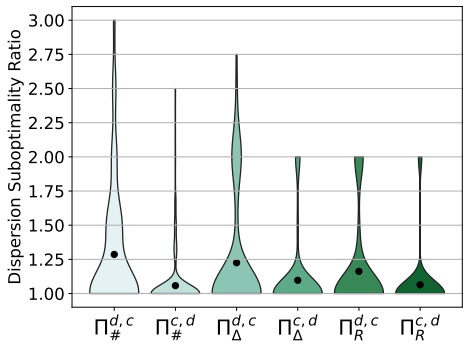


Figure 3: Distribution of the dispersion suboptimality ratio $\frac{d(\pi)}{d(\pi_d)}$ for each dispersion metric d.

Is Solving Π Good Enough? With the coverage and execution time results at hand, one could arguably ask whether solving the standard planning task Π is a good enough uniformity proxy. In order to refute this hypothesis, we compare the dispersion metric d induced by the plan obtained by the compilation that optimizes it, π_d, against the dispersion metric induced by the plan obtained by solving the standard planning task, π. We refer to this as the *dispersion suboptimality ratio*, and formally define it as $\frac{d(\pi)}{d(\pi_d)}$. Figure 3 shows this analysis as a set of violinplots, which represent the distribution of these ratios for each compilation. Here we consider problems that were commonly solved by the given compilation and the standard planning task. If we only focus

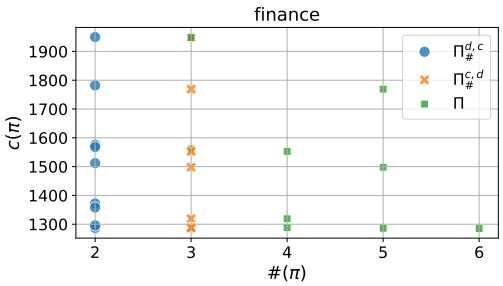


Figure 4: Cost (y) and number of different action costs (x) of the plans that solve Π, $\Pi_{\#}^{c,d}$, and $\Pi_{\#}^{d,c}$ tasks in FINANCE.

on cost-optimal plans, i.e., only in the $\Pi^{c,d}$ compilations, the solution obtained by solving the standard planning task, π, also minimizes the given dispersion metric in most of the tasks. On the other hand, if we focus on uniformest plans, i.e., only in the $\Pi^{d,c}$ compilations, we observe a larger suboptimality gap, meaning that π is not as uniform as it could.

Although these results might initially suggest that solving the standard planning task provides uniform enough plans, that would be a conclusion drawn from a myopic analysis. This is because most planning tasks have few cost-optimal plans involving very few different action costs. If we perform this analysis in tasks with plans that have different action costs, we can clearly observe that cost-optimal plans obtain dispersion metrics that are up to 3 times suboptimal. Figure 4 shows the cost and number of different action costs of the plans that solve Π, $\Pi_{\#}^{c,d}$, and $\Pi_{\#}^{d,c}$ tasks in FINANCE. As we can see, the cost-optimal plans (green squares) are the cheapest, but often involve more action costs. On the other side of the spectrum we have $\Pi_{\#}^{d,c}$ (blue points), which computes uniformest plans, breaking ties in favor of less costly plans. Solving the compilation allows us to get plans that, although not cost-optimal in some cases, only contain two different action costs. In this domain, this means the plan is suggesting to always save the same amount of money every month (one action cost), plus the last zero-cost action that checks if the financial goal has been achieved. $\Pi_{\#}^{c,d}$ (orange crosses) represent a good compromise between both approaches, as the plans generated are cost-optimal and require to use only three different action costs.

How Some Dispersion Metrics Approximate Others. Finally, we want to understand whether optimizing for some dispersion metrics allow us to get good values in other metrics. We perform this analysis with two objectives: (i) test if more scalable compilations such as $\Pi_{\#}$ also approximate Range or Delta; and (ii) test if some of these compilations also approximate well-established dispersion metrics for which we could not generate compilations such as standard deviation σ. Figure 5 shows the result of this analysis as a heatmap, where we have the compilations that generate uniformest plans in the rows, and the three different dispersion metrics plus standard deviation in the columns. The values in the cells of the heatmap indicate the average dispersion suboptimality ratio for each combination of compilation and dispersion metric in the 90 instances that were commonly solved by the three compilations. For example, a value of 1.00 in the $(\#(\pi), \Pi_{\#}^{d,c})$ cell indicates that, as expected, compilation $\Pi_{\#}^{d,c}$ was able to always find plans with optimal $\#(\pi)$ values. On the other hand, the 1.12 in the next column of the same row indicates that compilation $\Pi_{\#}^{d,c}$ returned plans with an average suboptimality of 1.12 with respect to the Delta dispersion metric. Since we cannot get the optimal standard deviation value for all the problems, we compute its dispersion suboptimality ratio by using the best σ value found by any compilation in the denominator.

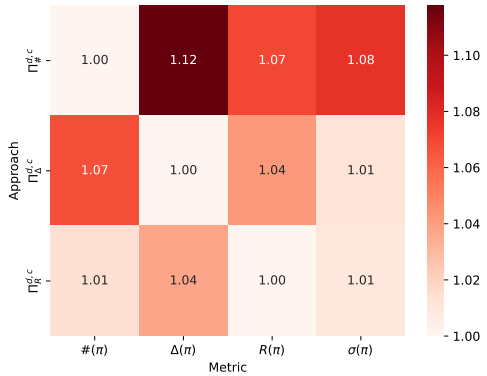


Figure 5: Dispersion suboptimality ratio for each combination of compilation (rows) and dispersion metric (columns). Lighter cells indicate better performance.

Cells with a lighter background indicate better (closer to the optimal) performance. If we analyze each approach (full row), we can observe that all the compilations approximate quite well all the dispersion metrics, including standard deviation. This makes sense, as all the compilations are in one way or another restricting the variety of the action costs used. In particular, $\Pi_{\Delta}^{d,c}$ and $\Pi_{R}^{d,c}$, the most complex compilations, obtain the best results across all dispersion metrics. On the other hand, the more scalable compilation $\Pi_{\#}^{d,c}$ shows only a slightly worse behavior, which makes it a good candidate as an affordable way of getting uniform plans.

Conclusions and Future Work

In this paper we have introduced a novel objective that might be relevant in many planning applications: that of finding plans whose actions have costs that are as uniform as possible. We have adapted three different dispersion metrics to automated planning, and presented six different compilations to produce plans that lexicographically optimize sum of action costs and the given dispersion metric. Experimental results across a large number of existing and novel planning tasks show that our methods are able to generate plans that optimally balance these two objectives. While some of the compilations generate tasks that are hard to solve in practice, our compilations to generate plans with a low number

of different action costs are able to scale on par with the standard planning task. The plans generated by solving these more amenable compilations are also able to implicitly optimize more complex dispersion metrics, making them good candidates to compute uniform plans in practice.

In this work we solely focused on computing optimal solutions for uniformest cost-optimal planning problems. In future work we would like to solve the compiled tasks using satisficing planners to study the trade-off between scalability and suboptimality. Finally, we focused on actions cost uniformity, as cost usually encodes the effort of executing an action. Extending our compilations to other definitions of uniformity such as preferring to use the same grounded (lifted) action is straightforward, and we would like to evaluate the resulting plans as part of future work.

Disclaimer

This paper was prepared for informational purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co. and its affiliates ("JP Morgan"), and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful.

References

Barth, M.; and Boriboonsomsin, K. 2009. Traffic congestion and greenhouse gases. *Access Magazine*, 1(35): 2–9.

Edelkamp, S.; Jabbar, S.; and Lluch-Lafuente, A. 2005. Cost-algebraic heuristic search. In *AAAI*, volume 5, 1362–1367.

Geißer, F.; Haslum, P.; Thiébaux, S.; and Trevizan, F. 2022. Admissible Heuristics for Multi-Objective Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, 100–109.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated planning - theory and practice*. Elsevier. ISBN 978-1-55860-856-6.

Golledge, R. G. 1995. Path selection and route preference in human navigation: A progress report. In *International conference on spatial information theory*, 207–222. Springer.

Gragera, A.; Fuentetaja, R.; Olaya, Á. G.; and Fernández, F. 2023. A Planning Approach to Repair Domains with Incomplete Action Effects. In Koenig, S.; Stern, R.; and Vallati, M., eds., *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling, July 8-13, 2023, Prague, Czech Republic*, 153–161. AAAI Press.

Hanna, S. D.; and Lindamood, S. 2010. Quantifying the economic benefits of personal financial planning. *Financial Services Review*, 19(2).

Helmert, M. 2006. The Fast Downward Planning System. *J. Artif. Intell. Res.*, 26: 191–246.

Hernández, C.; Yeoh, W.; Baier, J. A.; Zhang, H.; Suazo, L.; Koenig, S.; and Salzman, O. 2023. Simple and efficient bi-objective search algorithms via fast dominance checks. *Artificial Intelligence*, 314: 103807.

Katz, M.; Röger, G.; and Helmert, M. 2022. On Producing Shortest Cost-Optimal Plans. In *Proceedings of the International Symposium on Combinatorial Search*, volume 15, 100–108.

Katz, M.; Sohrabi, S.; Udrea, O.; and Winterer, D. 2018. A novel iterative approach to top-k planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, 132–140.

Mandow, L.; and De La Cruz, J. L. P. 2008. Multiobjective A* Search with Consistent Heuristics. *J. ACM*, 57(5).

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL - The Planning Domain Definition Language.

Pozanco, A.; Papasotiriou, K.; Borrajo, D.; and Veloso, M. 2023. Combining Heuristic Search and Linear Programming to Compute Realistic Financial Plans. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 33, 527–531.

Salzman, O.; Felner, A.; Hernandez, C.; Zhang, H.; Chan, S. H.; and Koenig, S. 2023. Heuristic-Search Approaches for the Multi-Objective Shortest-Path Problem: Progress and Research Opportunities. In *32nd International Joint Conference on Artificial Intelligence, IJCAI 2023*, 6759–6768. International Joint Conferences on Artificial Intelligence.

Shiffler, R. E.; and Harsha, P. D. 1980. Upper and lower bounds for the sample standard deviation. *Teaching Statistics*, 2(3): 84–86.

Sobrinho, J. L. 2001. Algebra and algorithms for QoS path computation and hop-by-hop routing in the Internet. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, volume 2, 727–735. IEEE.

Speck, D.; Mattmüller, R.; and Nebel, B. 2020. Symbolic top-k planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 9967–9974.

Stewart, B. S.; and White III, C. C. 1991. Multiobjective a. *Journal of the ACM (JACM)*, 38(4): 775–814.

Ulloa, C. H.; Yeoh, W.; Baier, J. A.; Zhang, H.; Suazo, L.; and Koenig, S. 2020. A simple and fast bi-objective search algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 143–151.