

Planning with Minimal Disruption

Alberto Pozanco, Marianela Morales, Daniel Borrajo, Manuela Veloso

J.P. Morgan AI Research

{alberto.pozanco,marianela.morales}@jpmorgan.com, {name.surname}@jpmorgan.com

Abstract

In many planning applications, we might be interested in finding plans that minimally modify the initial state to achieve the goals. We refer to this concept as plan disruption. In this paper, we formally introduce it, and define various planning-based compilations that aim to jointly optimize both the sum of action costs and plan disruption. Experimental results in different benchmarks show that the reformulated task can be effectively solved in practice to generate plans that balance both objectives.

Introduction

Classical planning is the task of finding a plan, which is a sequence of deterministic actions that, when executed from a given initial state, lead to a desired goal state (Ghallab, Nau, and Traverso 2004). Each action is associated with a non-negative cost, and the total cost of a plan is defined as the sum of the costs of its actions. 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 the quality of a plan (Sobrinho 2001; Geißer et al. 2022; Salzman et al. 2023).

In this paper, we present an objective that could be significant in various planning applications: the number of modifications required to transform the initial state into the goal state. We refer to this concept as *plan disruption*, and minimizing it leads to plans that retain as much of the initial state as possible while still achieving the goals. This property is often desired when computing solutions in related areas such as scheduling and optimization (Sakkout and Wallace 2000; Mütze 2014). Concrete real-world examples include employee scheduling (Clark and Walker 2011; Medard and Sawhney 2007) or project management (Zhu, Bard, and Yu 2005). This concept is closely related to the idea of *avoiding side effects* (Amodei et al. 2016), as it assumes that changes in the state which are not specified in the goal are undesirable and result from underspecification of objectives.

(Klassen et al. 2022) formalized the concept of avoiding side effects in planning and introduced a compilation to

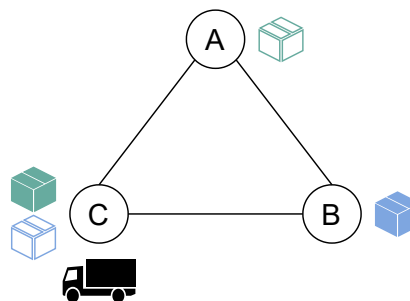


Figure 1: Logistics task where a truck must deliver two packages by moving them from their current (filled) locations to their goal (empty) destinations.

solve problems by minimizing the number of changes or side effects required to reach the goal state, without considering the quality of the plan used to achieve it. Also in the field of planning, some studies have explored the related problem of *plan stability*, which focuses on making minimal changes to an existing plan during replanning (Van Der Krogt and De Weerd 2005; Fox et al. 2006). We argue that in certain situations, such as when iteratively solving successive planning tasks within the same environment, minimizing plan disruption could be highly beneficial. In such scenarios, we may not have an existing plan to adhere to, but instead, we might be interested in computing a plan from scratch while making minimal changes to the initial state, as it includes certain elements that we wish to preserve.

Let us consider the logistics task depicted in Figure 1, where a truck must deliver two packages by moving them from their current (filled) locations to their goal (empty) destinations. Among the various plans that solve this task, there are two cost-optimal plans (with the minimum number of actions) that we would like to highlight. The first plan involves the truck traveling to B, loading the blue package, unloading it at C, then loading the green package, and finally unloading it at A. This plan successfully delivers all the packages but leaves the truck at A. Conversely, there is another cost-optimal plan where the truck first loads the green package, delivers it to A, picks up the blue package at B, and

finally delivers it to C. This plan also completes the delivery of all packages but leaves the truck at its original location, C, which may be advantageous if C serves as a depot or a strategic point for the truck’s positioning for future planning tasks (Pozanco, Torralba, and Borrajo 2024).

In simple examples like this, such constraints can sometimes be encoded as part of the goal state. However, this approach is not generally applicable, especially for larger tasks. In these cases, it is impossible to determine in advance which aspects of the initial state (such as which trucks) can retain their initial truth values without making the planning task unsolvable. A better strategy is to allow the planner to minimize the number of changes performed, ensuring that solutions will always exist.

We are interested in jointly optimizing both objectives: sum of action costs and plan disruption. This contrasts with (Klassen et al. 2022), which focuses exclusively on the plan disruption (side effects), disregarding the inherent trade-offs between these two objectives. There exist three main approaches to solve such multi-objective planning problems in the literature: cost-algebraic A* (Edelkamp, Jabbar, and Lluch-Lafuente 2005); multi-objective search algorithms such as NAMOA* (Mandow and De La Cruz 2008) or BOA* (Ulloa et al. 2020; Hernández et al. 2023); or reformulate the original planning task so that plans that solve the new task are plans that optimize the two objectives (Katz, Röger, and Helmert 2022; Pozanco, Borrajo, and Veloso 2024). While the first two approaches require developing new heuristics for each metric, reformulating the task allows us to leverage all the power of domain-independent heuristics and planners. We will reformulate the original task to generate plans that jointly optimize sum of action costs and plan disruption.

The main contributions of this paper are: (1) introduction of a novel bi-objective planning task with many real-world applications: jointly optimizing sum of action costs and plan disruption; and (2) definition of different compilations to produce plans that jointly optimize both objectives.

Background

We formally define a planning task as follows:

Definition 1 (Planning task). *A Planning task is a tuple $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$, where \mathcal{F} is a set of fluents, \mathcal{A} is a set of actions, $\mathcal{I} \subseteq \mathcal{F}$ is an initial state, and $\mathcal{G} \subseteq \mathcal{F}$ is a goal specification.*

A state $s \subseteq \mathcal{F}$ is a set of fluents that are true at a given time. A state $s \subseteq \mathcal{F}$ is a goal state if and only if $\mathcal{G} \subseteq s$. Each action $a \in \mathcal{A}$ is described by its name $\text{NAME}(a)$, which is a string; a set of positive and negative preconditions $\text{PRE}^+(a)$ and $\text{PRE}^-(a)$, which are set of fluents that need to be true (or false) for the action to be applied; add and delete effects $\text{ADD}(a)$ and $\text{DEL}(a)$, which are set of fluents that are added (or deleted) once the action is applied; and cost $c(a) \in \mathbb{R}$. An action a is applicable in a state s if and only if $\text{PRE}^+(a) \subseteq s$ and $\text{PRE}^-(a) \cap s = \emptyset$. We define the result of applying an action in a state as $\gamma(s, a) = (s \setminus \text{DEL}(a)) \cup \text{ADD}(a)$. We assume $\text{DEL}(a) \cap \text{ADD}(a) = \emptyset$. A sequence of actions $\pi = (a_1, \dots, a_n)$ is applicable in a state s_0 if there are states (s_1, \dots, s_n) such that a_i is applicable in

	π_1	π_2
\mathcal{I}	at(green C), at(blue B), at(truck C)	
$\Gamma(\mathcal{I}, \pi)$	at(green A), at(blue C), at(truck A)	at(green A), at(blue C), at(truck C)
$\mathcal{D}(\pi)$	6	4

Table 1: Plans π_1 and π_2 to reach \mathcal{G} in the LOGISTICS task shown in Figure 1. Rows in the table define the initial state \mathcal{I} , the goal state reached after executing each plan $\Gamma(\mathcal{I}, \pi)$, and the plan disruption $\mathcal{D}(\pi)$ of each plan.

s_{i-1} and $s_i = \gamma(s_{i-1}, a_i)$. The resulting state after applying a sequence of actions is $\Gamma(s, \pi) = s_n$, and $c(\pi) = \sum_i^n c(a_i)$ denotes the cost of π . A state s is reachable from state s' if and only if there exists an applicable action sequence π such that $s \subseteq \Gamma(s', \pi)$. A state s is a dead-end state if and only if it is not a goal state and no goal state is reachable from s . The solution to a planning task \mathcal{P} is a plan, i.e., a sequence of actions π such that $\mathcal{G} \subseteq \Gamma(\mathcal{I}, \pi)$. A plan with minimal cost is optimal.

Plan Disruption

In classical planning, a plan π solves a planning task if and only if all the goals are true in the final state $\mathcal{G} \subseteq \Gamma(\mathcal{I}, \pi)$. This definition overlooks the number of modifications to transform \mathcal{I} into \mathcal{G} . In this section, we introduce plan disruption, a metric that counts the number of propositions that differ between \mathcal{I} and $\Gamma(\mathcal{I}, \pi)$. Specifically, the disruption of a plan is calculated as the cardinality of the symmetric difference of the initial and goal state sets. The symmetric difference Δ of two sets is an operation that returns a set that includes elements present in either of the two sets but absent in their intersection. This definition is similar to the concept of fluent side effect in (Klassen et al. 2022).

Definition 2 (Plan Disruption). *Given a planning task $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ and a plan π that solves it, the plan disruption of π is defined as:*

$$\mathcal{D}(\pi) = |\mathcal{I} \Delta \Gamma(\mathcal{I}, \pi)|$$

Let us illustrate Plan Disruption by using the LOGISTICS task depicted in Figure 1. Table 1 describes the initial state of the task \mathcal{I} , together with the two cost-optimal plans outlined in the Introduction. The first plan π_1 involves the truck traveling to B, loading the blue package, unloading it at C, then loading the green package, and finally unloading it at A. In the second plan π_2 , the truck first loads the green package, delivers it to A, picks up the blue package at B, and finally delivers it to C. The third row of Table 1 shows the goal state after executing each respective plan, while the final row presents the plan disruption metric for each plan. After executing π_1 , none of the fluents that were true in the initial state remain true in the goal state. Instead, three new fluents become true in the goal state that were not true initially, leading to a plan disruption of 6. In contrast, plan π_2 keeps the truck at its original location C, altering only the fluents related to the packages’ positions. This results in a plan disruption value of 4.

We can determine both a lower and an upper bound for the plan disruption of a planning task without actually computing a plan. The lower bound is given by the minimum changes required to transition from \mathcal{I} to \mathcal{G} . On the other hand, the upper bound corresponds to the maximum number of changes that could occur when going from \mathcal{I} to \mathcal{G} . This is given by the total number of fluents \mathcal{F} in the planning task, excluding those goal fluents that are already true in the initial state, as they do not need to change. We provide these bounds to offer theoretical insight into the possible values of plan disruption, and formalize them as follows:

Proposition 1. *Given a planning task $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$, the lower bound ℓ and the upper bound \mathcal{L} of the plan disruption $\mathcal{D}(\pi)$ of any plan π that solves \mathcal{P} are defined as:*

$$\ell(\mathcal{D}(\pi)) = |\mathcal{G} \setminus \mathcal{I}| \quad \mathcal{L}(\mathcal{D}(\pi)) = |\mathcal{F}| - |\mathcal{G} \cap \mathcal{I}|$$

Proof. To establish the lower bound $\ell(\mathcal{D}(\pi))$, consider the initial state \mathcal{I} and the goal state \mathcal{G} . The lower bound is determined by the fluents that are in the goal state but not in the initial state, i.e., $|\mathcal{G} \setminus \mathcal{I}|$. This represents the minimum number of changes required to achieve the goal state from the initial state, as these fluents must be made true by any plan that solves the task.

For the upper bound $\mathcal{L}(\mathcal{D}(\pi))$, any fluent in $\{\mathcal{G} \cap \mathcal{I}\}$ is already true in \mathcal{I} and must also be true in the final state $\Gamma(\mathcal{I}, \pi)$ of any plan π . Thus, such fluents cannot differ between \mathcal{I} and $\Gamma(\mathcal{I}, \pi)$ and never contribute to disruption. At most, every other fluent can differ once relative to \mathcal{I} , and therefore $\mathcal{L}(\mathcal{D}(\pi)) = |\mathcal{F}| - |\mathcal{G} \cap \mathcal{I}|$. This ensures that $\mathcal{L}(\mathcal{D}(\pi))$ is indeed an upper bound, as it represents the maximum number of changes possible from the initial state. \square

Computing Plans that Minimize Plan Disruption

We present two different compilations that optimize plan disruption. The compilations balance the trade-off between the accuracy of optimizing plan disruption and the complexity of the reformulated task. In all cases, we will use $\omega \in \mathbb{R}_0^+$ to represent the weight assigned to the importance of plan disruption in the quality (cost) of a plan. Next, we present these compilations, starting with the most accurate and complex, and progressing to the least accurate and simplest.

Compilation 1: Lazy Plan Disruption

The first compilation seeks to minimize Plan Disruption by examining, after the goals have been achieved, which fluents have changed their values from the initial to the goal state. This compilation follows a similar structure to the soft goals compilation by Keyder and Geffner (2009) to solve oversubscription planning tasks (Smith 2004), which also inspired the compilation in (Klassen et al. 2022) to avoid side effects. The plan is divided into two parts. The first part focuses on achieving the hard goals, i.e., the original goals of the problem \mathcal{G} . The second part addresses the soft goals by imposing penalties (increasing the cost of the plan) for not achieving them. In this case the soft goals involve maintaining the truth values of the fluents in the goal state as they were in the initial state.

Given a planning task \mathcal{P} , we extend the set of fluents \mathcal{F} as follows:

- $\mathcal{F}_I = \bigcup_{f \in \mathcal{I}} \{\text{init}_f\}$, a set of propositions that mark whether a fluent $f \in \mathcal{I}$ was true in the initial state.
- $\mathcal{F}_c = \bigcup_{f \in \mathcal{F}} \{\text{checked}_f\}$, a set of propositions that mark whether a fluent $f \in \mathcal{F}$ has been checked or not. As seen later, this check will test the truth value of f at the end of planning, increasing the total cost of the plan depending on its truth value in \mathcal{I} .
- ga , a proposition representing that all goals in \mathcal{G} have been achieved.
- end , a proposition that represents the end of the planning episode.

We update the actions in \mathcal{A} to force that the original actions are only executed before the goals have been achieved. For each action $a \in \mathcal{A}$, we generate a modified version a' that is stored in a new set \mathcal{A}' . Each action a' is defined as follows:

- $\text{NAME}(a') = \text{NAME}(a)$
- $\text{PRE}^+(a') = \text{PRE}^+(a)$
- $\text{PRE}^-(a') = \text{PRE}^-(a) \cup \{\text{ga}\}$
- $\text{ADD}(a') = \text{ADD}(a)$
- $\text{DEL}(a') = \text{DEL}(a)$
- $c(a') = c(a)$

Apart from updating the original actions, we also extend \mathcal{A} with new actions that can only be executed after the goals have been achieved. The first new action a^G sets ga to true once a goal state is reached:

- $\text{NAME}(a^G) = \text{goalstate}$
- $\text{PRE}^+(a^G) = \mathcal{G}$
- $\text{PRE}^-(a^G) = \emptyset$
- $\text{ADD}(a^G) = \{\text{ga}\}$
- $\text{DEL}(a^G) = \emptyset$
- $c(a^G) = 0$

The second set of new actions will increase the total cost of the plan depending on the number of changes between the initial and goal states. In particular, we generate two actions for each fluent $f \in \mathcal{F}$: a collect^f action that does not increase the total cost and can be executed if and only if f has the same truth value in both states; and a forgo^f action that increases the total cost by ω when the truth value of f has changed. We store these actions in a new set \mathcal{A}^c . Below we show the forgo and collect actions.

- $\text{NAME}(a^f) = \text{forgo}^f$
- $\text{PRE}^+(a^f) = \{\text{ga}\}$
- $\text{PRE}^-(a^f) = \{\text{checked}_f, \text{end}\}$
- $\text{ADD}(a^f) = \{\text{checked}_f\}$
- $\text{DEL}(a^f) = \emptyset$
- $c(a^f) = \omega$

We only show the collect action for the positive case, i.e., when init_f is true. The other collect case is defined analogously by adding f and init_f to PRE^- .

- $\text{NAME}(a^f) = \text{collect}^f$
- $\text{PRE}^+(a^f) = \{f, \text{init}_f, \text{ga}\}$
- $\text{PRE}^-(a^f) = \{\text{checked}_f, \text{end}\}$
- $\text{ADD}(a^f) = \{\text{checked}_f\}$
- $\text{DEL}(a^f) = \emptyset$
- $c(a^f) = 0$

Finally, we extend \mathcal{A} with a new action a^{end} that makes end true when all the goals have been achieved and the truth value of all the fluents in \mathcal{F} have been checked.

- $\text{NAME}(a^{\text{end}}) = \text{end}$
- $\text{PRE}^+(a^{\text{end}}) = \{\text{ga}\} \cup \bigcup_{f \in \mathcal{F}} \{\text{checked}_f\}$
- $\text{PRE}^-(a^{\text{end}}) = \emptyset$
- $\text{ADD}(a^{\text{end}}) = \{\text{end}\}$
- $\text{DEL}(a^{\text{end}}) = \emptyset$
- $c(a^{\text{end}}) = 0$

Definition 3 (Lazy Plan Disruption Planning Task). *Given a planning task $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$, a lazy plan disruption planning task \mathcal{P}_L^ω that optimizes sum of action costs and plan disruption is a tuple $\mathcal{P}_L^\omega = \langle \mathcal{F}_L, \mathcal{A}_L, \mathcal{I}_L, \mathcal{G}_L \rangle$ where:*

- $\mathcal{F}_L = \mathcal{F} \cup \mathcal{F}_I \cup \mathcal{F}_c \cup \{\text{ga}, \text{end}\}$
- $\mathcal{A}_L = \mathcal{A}' \cup \mathcal{A}^c \cup \{a^G, a^{\text{end}}\}$
- $\mathcal{I}_L = \mathcal{I} \cup \mathcal{F}_I$
- $\mathcal{G}_L = \{\text{end}\}$

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

$$|\mathcal{A}_L| = |\mathcal{A}'| + \overbrace{(2 \times |\mathcal{F}|)}^{\mathcal{A}^c} + \overbrace{2}^{a^G, a^{\text{end}}}$$

The completeness and soundness of the lazy plan disruption planning task \mathcal{P}_L^ω with respect to the original planning task \mathcal{P} are straightforward derived from Keyder's work (Keyder and Geffner 2009).

Proposition 2. *Given a planning task \mathcal{P} and a plan π that solves it, there exists a plan π' mapped from π that solves \mathcal{P}_L^1 , where the plan disruption of π , $\mathcal{D}(\pi)$, is equal to the plan disruption obtained by the lazy compilation, $\mathcal{D}_L(\pi)$, and is given by:*

$$\mathcal{D}(\pi) = c(\pi') - c(\pi) = \mathcal{D}_L(\pi)$$

Proof. Observe that π' is mapped from π , i.e., π' is obtained by appending to π the a^G action followed by some permutation of the actions in \mathcal{A}^c and finally by the end action a^{end} . Since the cost of the appended actions is zero except for the forgo^f actions in \mathcal{A}^c , then we have that $c(\pi') = c(\pi) + \sum_{\text{forgo}^f \in \pi'} c(\text{forgo}^f)$. We now have to check that $\sum_{\text{forgo}^f \in \pi'} c(\text{forgo}^f) = \mathcal{D}(\pi)$.

A forgo^f action is executed if a fluent that was true in the initial state is no longer true in the goal state, and viceversa, increasing the total cost by $\omega = 1$. This behavior mirrors the plan disruption metric, which is defined as the symmetric difference between the initial and goal states.

Therefore, the total cost of the forgo^f actions with $\omega = 1$, $\sum_{\text{forgo}^f \in \mathcal{A}^c} c(\text{forgo}^f)$, is equal to the plan disruption $\mathcal{D}(\pi)$, as it effectively counts the number of fluents that differ between the initial and goal states. \square

Compilation 2: Eager Plan Disruption

The previous compilation is capable of accurately replicating the Plan Disruption metric. However, it has a notable limitation: the quality of a plan, in terms of plan disruption, can only be assessed at the conclusion of the planning process, with no intermediate indicators. This limitation may lead search algorithms to backtrack multiple times in their quest to identify optimal plans. To address this issue, we propose a new compilation approach that prioritizes search efficiency over the precise accuracy of plan disruption. This is achieved by continuously monitoring the number of changes made to the initial state throughout the planning process. However, this monitoring will only compare the current state with \mathcal{I} , which might introduce inaccuracies in the plan disruption metric. By doing so, we aim to enhance the search algorithm's efficiency, allowing it to more effectively navigate towards optimal plans.

Given a planning task \mathcal{P} , we update each original action $a \in \mathcal{A}$ by modifying its cost depending on (1) the number of add and delete effects of a ; and (2) the value of the added/deleted fluents in the initial state \mathcal{I} . We refer to the set of updated actions as \mathcal{A}' , and formally define each action $a' \in \mathcal{A}'$ as follows:

- $\text{NAME}(a') = \text{NAME}(a)$
- $\text{PRE}^+(a') = \text{PRE}^+(a)$
- $\text{PRE}^-(a') = \text{PRE}^-(a)$
- $\text{ADD}(a') = \text{ADD}(a)$
- $\text{DEL}(a') = \text{DEL}(a)$
- $c(a') = c(a) + \omega(|(\text{ADD}(a) \setminus \mathcal{I}) \cup (\text{DEL}(a) \cap \mathcal{I})|)$

As we can see, the only difference between a and a' lies in the cost, which now incorporates an extra term weighted by ω . This extra term counts the number of propositions that are added by a and were not present in \mathcal{I} , plus the number of propositions that were present in \mathcal{I} and are removed by a . Since this check is only done with respect to \mathcal{I} , it might not accurately measure plan disruption, as seen later.

Definition 4 (Eager Plan Disruption Planning Task). *Given a planning task $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$, an eager plan disruption planning task \mathcal{P}_E^ω that optimizes sum of action costs and plan disruption is a tuple $\mathcal{P}_E^\omega = \langle \mathcal{F}, \mathcal{A}', \mathcal{I}, \mathcal{G} \rangle$.*

Remark 1. *Observe that the plan disruption of π that solves \mathcal{P} approximated by the eager compilation \mathcal{P}_E^ω can be obtained as in Proposition 2: there exists π'' mapped from π that solves \mathcal{P}_E^1 and the plan disruption of π obtained by the eager compilation is given by:*

$$\mathcal{D}_E(\pi) = c(\pi'') - c(\pi) \geq \mathcal{D}(\pi)$$

Proposition 3. *Given a planning task \mathcal{P} and a plan π that solves it, the plan disruption obtained by the lazy compilation $\mathcal{D}_L(\pi)$, is always lower than or equal to the plan disruption obtained by the eager compilation $\mathcal{D}_E(\pi)$.*

Proof. By Proposition 2 and Remark 1, we have that there exist π' and π'' that solve \mathcal{P}_L^1 and \mathcal{P}_E^1 respectively, and the plan disruption obtained by \mathcal{P}_L^1 is $\mathcal{D}_L(\pi) = c(\pi') - c(\pi)$ and the one obtained by \mathcal{P}_E^1 is $\mathcal{D}_E(\pi) = c(\pi'') - c(\pi)$.

$\mathcal{F} = \{A, B, C, D\}, \mathcal{I} = \{A, B\}$	$\mathcal{G} = \{D\}$
$\text{PRE}^+(a_1) = \{A\}$	$\text{PRE}^+(a_2) = \{C\}$
$\text{DEL}(a_1) = \{A, B\}$	$\text{DEL}(a_2) = \{A\}$
$\text{ADD}(a_1) = \{C\}$	$\text{ADD}(a_2) = \{D, B\}$
$c(a_1) = 10$	$c(a_2) = 10$
$\pi = (a_1, a_2) \Rightarrow (s_0 = \{A, B\}, s_1 = \{C\}, s_2 = \{C, D, B\})$	

Table 2: Planning task where the first row defines the fluents, as well as the initial and goal states; the second row defines the available actions \mathcal{A} ; and the third row shows a plan π that solves the task along with the states it traverses.

Then we have to show that $c(\pi') \leq c(\pi'')$. In particular, $c(\pi') = c(\pi) + \sum_{\text{forgo}^f \in \pi'} c(\text{forgo}^f)$, and $c(\pi'') = c(\pi) + \sum_{a \in \pi} |(\text{ADD}(a) \setminus \mathcal{I}) \cup (\text{DEL}(a) \cap \mathcal{I})|$. We then have to show that

$$\sum_{\text{forgo}^f \in \pi'} c(\text{forgo}^f) \leq \sum_{a \in \pi} |(\text{ADD}(a) \setminus \mathcal{I}) \cup (\text{DEL}(a) \cap \mathcal{I})|$$

Consider a fluent f that was not true in \mathcal{I} and is added to the state by an action. If f is later removed by another action, the eager task \mathcal{P}_E^1 still counts the addition of f even if it is subsequently removed, since \mathcal{P}_E^1 does not account for net changes. The lazy task \mathcal{P}_L^1 , however, does not count such transient changes if they do not result in a net effect on the goal state. This results in a lower or equal cost compared to the eager task, leading to the conclusion that the plan disruption for \mathcal{P}_L^1 is always less than or equal to that for \mathcal{P}_E^1 . \square

Example

Let us show how the two compilations approximate Plan Disruption by using the simple planning task depicted in Table 2.

The Plan Disruption of π is $|\{A, B\} \Delta \{C, D, B\}| = 3$, since A is present in the initial but not in the goal state, and C and D are true in the goal but not in the initial state. Let us see how both compilations \mathcal{P}_L^1 and \mathcal{P}_E^1 approximate Plan Disruption.

Consider the following plan, which optimally solves \mathcal{P}_L^1 :

$$\pi_L = (a_1, a_2, \text{goalstate}, \text{forgo}^A, \text{collect}^B, \text{forgo}^C, \text{forgo}^D, \text{end})$$

The first two actions, a_1 and a_2 , contribute 20 to the total cost (10 + 10). From that moment, the remainder of the plan increases the total cost depending on the plan disruption metric induced by the execution of these two actions. The next action, goalstate has a cost of 0, the same as the final end action. The rest of the actions in the plan check the truth values of all the fluents \mathcal{F} in the reached goal state. forgo^A increases the total cost by 1, as A was removed. collect^B does not increase the total cost, as B remains true in the goal state. And both forgo^C and forgo^D increase the total cost by 1 since C and D appear in the goal but not in the initial state. Therefore, following Proposition 2, the part of the total cost that belongs to the plan disruption metric is 3, which equals the actual value of the metric.

On the other hand consider the following plan, which optimally solves \mathcal{P}_E^1 :

$$\pi_E = (a_1, a_2)$$

The cost of a_1 is 10 (the cost of the original action), plus 3: 2 fluents removed from \mathcal{I} (A and B), and 1 fluent added (C). The cost of a_2 is 10 (the cost of the original action), plus 2: 1 fluent removed from \mathcal{I} (A), and 1 fluent added (D), since B was already true in \mathcal{I} . Therefore, according to Remark 1, the part of the total cost that belongs to the plan disruption metric is 5, which is 2 units higher than the actual plan disruption metric. This difference comes from the fact that A and B are wrongly counted twice. In the case of A both actions delete it, but only the first time should be considered. In the case of B , it is first deleted by a_1 and then added by a_2 , so it should not be counted as it is true both in the initial and goal state.

In summary, while \mathcal{P}_L^ω can accurately assess the disruption of a plan, it requires a polynomial increase in the number of fluents and actions. On the other hand, \mathcal{P}_E^ω may not be as accurate in certain situations, but it offers the advantage of not needing to introduce new actions, only adjusting the cost of the original ones.

Evaluation

Experimental Setting

Benchmark. We selected all the STRIPS tasks from the optimal suite of the Fast Downward (Helmert 2006) benchmark collection¹. This gives us 1847 original tasks \mathcal{P} divided across 66 domains.

Approaches. We evaluate the two compilations, namely Lazy Plan Disruption \mathcal{P}_L^ω and Eager Plan Disruption \mathcal{P}_E^ω on the above benchmark.

We experiment with $\omega = \{10^{-3}, 1, 10^3\}$ to weight differently the importance of plan disruption wrt. sum of action costs. With $\omega = 10^{-3}$ we can expect plan disruption to just serve as a tie breaker, with sum of action costs being the main driver for solution's quality. With $\omega = 1$ we can expect plan disruption to have a similar weight than sum of action costs in the quality of a solution. Finally, with $\omega = 10^3$ we can expect plan disruption to drive the optimization, with the sum of action costs only serving as tie breaker. We will compare the compilations to a baseline, which consists on solving the original task \mathcal{P} , where optimal plans are only defined by the sum of action costs.

Reproducibility. We solve all the planning tasks (\mathcal{P}_L^ω , \mathcal{P}_E^ω and \mathcal{P}) using the SEQ-OPT-LMCUT configuration of Fast Downward, which runs A^* with the admissible LMCUT heuristic to compute an optimal plan. 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.

Results

Coverage and Execution Time Overhead. First, we aim to understand the difficulty of solving our compiled tasks

¹<https://github.com/aibasel/downward-benchmarks>

Domain (# Problems)	\mathcal{P}	$\mathcal{P}_E^{10^{-3}}$	\mathcal{P}_E^1	$\mathcal{P}_E^{10^3}$	$\mathcal{P}_L^{10^{-3}}$	\mathcal{P}_L^1	$\mathcal{P}_L^{10^3}$
agricola-opt18 (20)	0	0	0	0	0	0	0
airport (50)	28	23	24	23	0	0	0
barman-opt11 (20)	4	4	4	4	0	0	0
barman-opt14 (14)	0	0	0	0	0	0	0
blocks (35)	28	28	28	30	3	3	3
childsack-opt14 (20)	0	0	0	0	0	0	0
data-net-opt18 (20)	12	12	13	19	0	0	0
depot (22)	7	7	7	7	0	0	0
driverlog (20)	13	14	14	13	1	1	1
elevators-opt08 (30)	22	18	19	18	0	0	0
elevators-opt11 (20)	18	15	16	15	0	0	0
floortile-opt11 (20)	7	6	6	5	0	0	0
floortile-opt14 (20)	6	5	5	2	0	0	0
freecell (80)	15	15	15	11	0	0	0
ged-opt14 (20)	15	13	20	20	0	0	0
grid (5)	2	2	2	2	0	0	0
gripper (20)	7	7	7	7	5	4	4
hiking-opt14 (20)	9	9	9	10	3	0	0
logistics00 (28)	20	20	20	20	0	0	0
logistics98 (35)	6	6	6	6	0	0	0
miconic (150)	141	141	141	141	36	35	35
movie (30)	30	30	30	30	30	30	30
mprime (35)	22	22	24	22	0	0	0
mystery (30)	17	16	17	16	0	0	0
nomystery-opt11 (20)	14	14	14	14	0	0	0
openstacks-opt08 (30)	21	14	8	7	9	6	6
openstacks-opt11 (20)	16	9	3	2	4	1	1
openstacks-opt14 (20)	3	1	0	0	0	0	0
openstacks (30)	7	7	7	7	5	5	5
org-syn-opt18 (20)	7	7	7	7	0	0	0
org-syn-split-opt18 (20)	15	14	15	15	0	0	0
parcprinter-08 (30)	18	21	19	19	3	3	3
parcprinter-opt11 (20)	13	16	14	14	0	0	0
parking-opt11 (20)	2	1	1	1	0	0	0
parking-opt14 (20)	3	1	0	0	0	0	0
pathways (30)	5	5	5	5	0	0	0
pegsol-08 (30)	28	27	26	26	0	0	0
pegsol-opt11 (20)	18	17	16	16	0	0	0
petri-net-opt18 (20)	9	8	11	11	0	0	0
pipes-notank (50)	17	14	17	13	0	0	0
pipes-tank (50)	12	8	9	8	0	0	0
psr-small (50)	49	49	49	49	14	13	13
quantum-opt23 (20)	11	11	11	11	0	0	0
rovers (40)	7	8	9	10	2	1	1
satellite (36)	7	8	11	11	1	1	1
scanalyzer-08 (30)	15	9	9	7	3	3	3
scanalyzer-opt11 (20)	12	6	6	4	1	1	1
snake-opt18 (20)	6	4	6	4	0	0	0
sokoban-opt08 (30)	29	27	25	21	0	0	0
sokoban-opt11 (20)	20	20	19	17	0	0	0
spider-opt18 (20)	11	7	6	6	0	0	0
storage (30)	15	15	15	15	1	1	1
termes-opt18 (20)	5	4	5	5	0	0	0
tetris-opt14 (17)	6	3	4	3	0	0	0
tidybot-opt11 (20)	14	13	12	10	0	0	0
tidybot-opt14 (20)	8	7	7	5	0	0	0
tpp (30)	6	6	6	6	3	3	3
transport-opt08 (30)	11	11	11	11	3	3	3
transport-opt11 (20)	6	6	6	6	0	0	0
transport-opt14 (20)	6	6	6	6	0	0	0
trucks (30)	10	10	10	10	0	0	0
visitall-opt11 (20)	10	11	12	10	7	6	6
visitall-opt14 (20)	5	6	6	5	1	0	0
woodworking-opt08 (30)	17	17	18	20	1	1	1
woodworking-opt11 (20)	12	11	12	14	0	0	0
zenotravel (20)	13	12	13	11	2	2	2
Total (1847)	948	884	893	863	138	123	123

Table 3: Number of problems solved by each compilation.

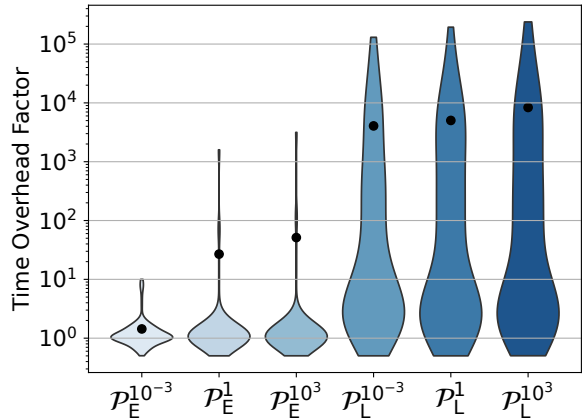


Figure 2: Distribution of the execution time overhead factor $\frac{T(X)}{T(\mathcal{P})}$ for each compilation X . Black dots represent the average.

compared to the original task. Table 3 presents the coverage of SEQ-OPT-LMCUT when solving both the compiled and original tasks across all domains and problems. As expected, solving the standard planning task, where only cost optimization is considered, is easier, allowing the planner to find cost-optimal plans for 948 instances. The eager compilation \mathcal{P}_E^ω , which merely updates the cost of the original actions, solves nearly the same number of tasks, with 893 instances when $\omega = 1$. In contrast, the lazy compilation \mathcal{P}_L^ω , which requires introducing additional actions and transforming the original problem into an oversubscription planning task, solves only 138 tasks. This clearly indicates that the lazy compilation results in tasks that are more complex than both the eager compilation and the original tasks.

In order to further understand the overhead introduced 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 \mathcal{P} . We refer to this as the *time overhead factor*, and formally define it as $\frac{T(X)}{T(\mathcal{P})}$. 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 123 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 compilation. As we can see, the eager compilations hardly introduce any overhead compared to solving the original task. Most of the tasks can be solved in the same amount of time, with around 50 tasks that are faster to solve under the new cost's setting. On the other hand, solving the lazy tasks can require execution times that are one to five orders of magnitude longer.

Is Solving \mathcal{P} Good Enough? Given the coverage and execution time results, one might question whether solving the standard planning task \mathcal{P} already produces plans with minimal disruption. To challenge this hypothesis, we compare the disruption metrics of the plans obtained from solving

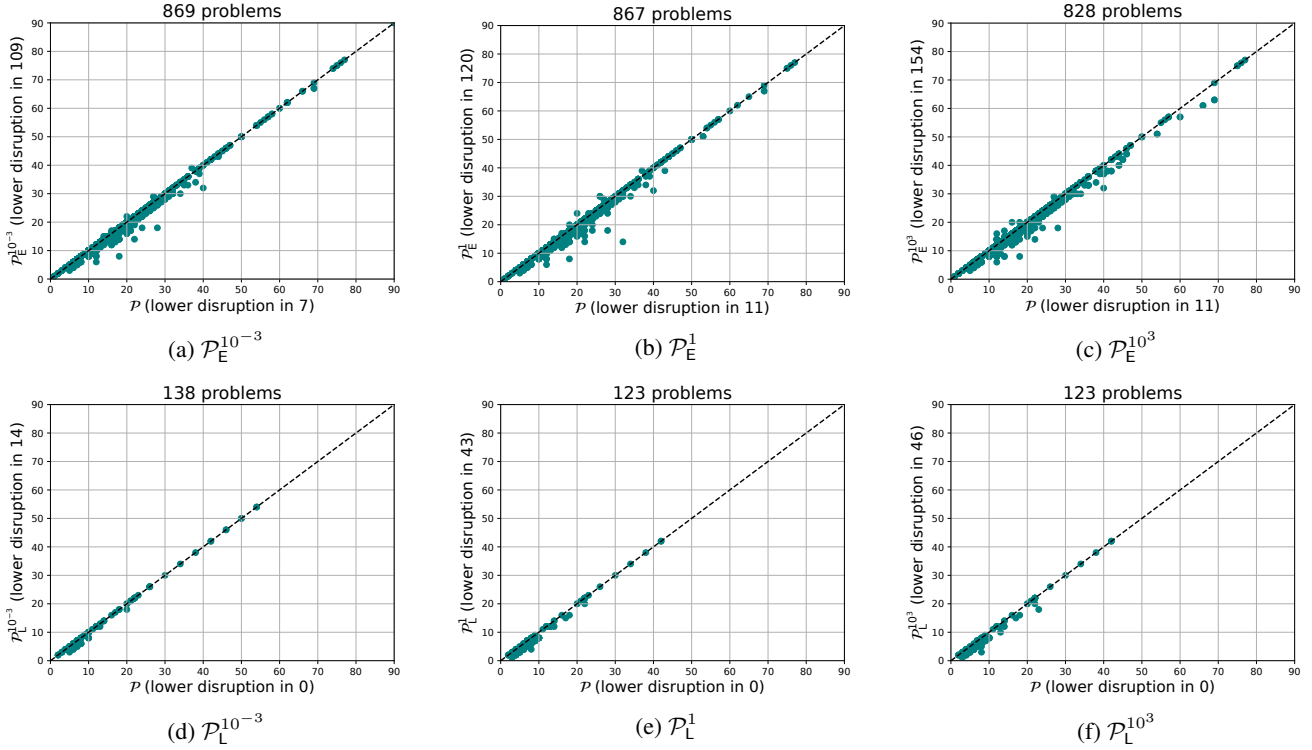


Figure 3: Plan disruption for optimally solved compiled tasks (y -axis) versus original tasks (x -axis). Figure titles show the number of problems plotted, i.e., those solved by both the compilation and the original task.

each compilation with those of the plans derived from solving the original task. These results are shown in Figure 3. The title of each subfigure indicates the number of problems represented in the plot, i.e., those commonly solved by the given compilation (y -axis) and the original task (x -axis). Points below the diagonal indicate that the plan returned after solving the compilation has a lower (better) plan disruption value.

As we can see, most points lie along the diagonal regardless of the compilation, indicating that the cost-optimal plan has the same disruption value as the compilation that explicitly optimizes this metric. In both the eager and lazy compilations, higher ω values result in more plans with better disruption values compared to those produced by the original task. In the eager compilation \mathcal{P}_E^ω , this difference increases from $109/869 = 12.5\%$ of the plans with $\omega = 10^{-3}$ to $154/828 = 18.5\%$ with $\omega = 10^3$. A similar trend is seen in the lazy compilation \mathcal{P}_L^ω , where the percentage rises from 10% to 37%. As discussed in Section 5, the eager compilation, although serving as a proxy for plan disruption, may introduce noise in its computation. In some cases, this can lead to plans that inaccurately assess the correct disruption metric, resulting in the cost-optimal plan having lower disruption than the one derived from solving the eager compilation. Conversely, the lazy compilation, which accurately computes the plan disruption, never returns a plan with higher disruption than when solving the original task.

Trading-off Plan Disruption and Cost. The previous results suggest that there is not much variability in the plan disruption of the plans that solve the planning tasks in the benchmark. Although this is the general trend, there are some tasks where we can observe a trade-off between optimizing plan cost and disruption. This is exemplified in the SATELLITE task depicted in Figure 4, which shows the cost (y -axis) and disruption (x -axis) of the plans returned by each compilation. As we can see, the plan we get after solving the original task \mathcal{P} (blue dot) is cost-optimal (9) but has the highest disruption value, making 8 changes to the initial state in order to achieve the goal. All the compilations manage to get plans with lower disruption, with some of them achieving the same cost as to when solving \mathcal{P} . On the other extreme of the spectrum we have the lazy compilation with $\omega = 10^3$, which is able to return a plan with a plan disruption of 3 at the expense of increasing the cost of the plan from 9 to 12. This result clearly demonstrates that when the original task offers sufficient diversity in the plans that solve it, our proposed compilations can effectively balance plan cost and disruption, yielding plans that prioritize each objective differently.

Conclusions and Future Work

In this paper, we introduce an objective that may be relevant to many planning applications: finding plans that minimally alter the initial state to achieve the goals. We term this concept *plan disruption* and propose two compilations

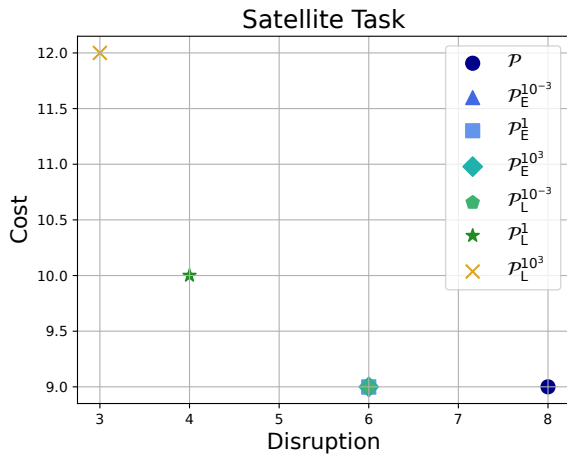


Figure 4: Plan Cost (y -axis) vs Plan Disruption (x -axis) of the plans that solve a SATELLITE task under the different compilations.

that jointly optimize plan cost and disruption. Experimental results from a comprehensive benchmark indicate that, although most planning tasks exhibit limited variability in plan disruption, our compilations effectively balance both objectives in tasks where there is potential for improving plan disruption. The eager compilation scales similarly to the standard planning task and effectively minimizes plan disruption. However, because it minimizes a proxy for plan disruption rather than the actual metric, it can occasionally produce plans with higher disruption values than those obtained from solving the original task. Conversely, the lazy compilation generally requires several orders of magnitude more execution time, making it unsuitable for larger planning tasks. Despite this, the additional time investment results in plans with the lowest possible plan dispersion, making it the preferred compilation for smaller planning tasks.

In this work we solely focused on computing optimal solutions for all the tasks, revealing that some of our compilations face scalability challenges. In future work we would like to solve the reformulated tasks using satisficing planners to study the trade-off between scalability and suboptimality. In this paper, we treated all the atoms in the initial state equally. We would also like to allow users to specify the importance of keeping the truth value of each atom, either by explicitly assigning weights or implicitly by providing input plans that reflect their preferences (Morales et al. 2024).

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 recommen-

dation, 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

- Amodei, D.; Olah, C.; Steinhardt, J.; Christiano, P.; Schulman, J.; and Mané, D. 2016. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*.
- Clark, A. R.; and Walker, H. 2011. Nurse rescheduling with shift preferences and minimal disruption. *Journal of Applied Operational Research*, 3(3): 148–162.
- Edelkamp, S.; Jabbar, S.; and Lluch-Lafuente, A. 2005. Cost-algebraic heuristic search. In *AAAI*, volume 5, 1362–1367.
- Fox, M.; Gerevini, A.; Long, D.; Serina, I.; et al. 2006. Plan Stability: Replanning versus Plan Repair. In *ICAPS*, volume 6, 212–221.
- 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.
- Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research*, 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.
- Keyder, E.; and Geffner, H. 2009. Soft goals can be compiled away. *Journal of Artificial Intelligence Research*, 36: 547–556.
- Klassen, T. Q.; McIlraith, S. A.; Muise, C.; and Xu, J. 2022. Planning to avoid side effects. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 9830–9839.
- Madow, L.; and De La Cruz, J. L. P. 2008. Multiobjective A* Search with Consistent Heuristics. *J. ACM*, 57(5).
- Medard, C. P.; and Sawhney, N. 2007. Airline crew scheduling from planning to operations. *European Journal of Operational Research*, 183(3): 1013–1027.
- Morales, M.; Pozanco, A.; Canonaco, G.; Gopalakrishnan, S.; Borrajo, D.; and Veloso, M. 2024. On learning action costs from input plans. *arXiv preprint arXiv:2408.10889*.
- Mütze, T. 2014. Scheduling with few changes. *European Journal of Operational Research*, 236(1): 37–50.

- Pozanco, A.; Borrajo, D.; and Veloso, M. 2024. On Computing Plans with Uniform Action Costs. In *Workshop on Human-Aware Explainable Planning at ICAPS*.
- Pozanco, A.; Torralba, Á.; and Borrajo, D. 2024. Computing Planning Centroids and Minimum Covering States Using Symbolic Bidirectional Search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 455–463.
- Sakkout, H. E.; and Wallace, M. 2000. Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints*, 5: 359–388.
- 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.
- Smith, D. E. 2004. Choosing Objectives in Over-Subscription Planning. In *ICAPS*, volume 4, 393.
- 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.
- 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.
- Van Der Krogt, R.; and De Weerd, M. 2005. Plan Repair as an Extension of Planning. In *ICAPS*, volume 5, 161–170.
- Zhu, G.; Bard, J. F.; and Yu, G. 2005. Disruption management for resource-constrained project scheduling. *Journal of the Operational Research Society*, 56(4): 365–381.