Choosing the Initial State for Online Replanning

Maximilian Fickert,[*1] **Ivan Gavran,**[*2] **Ivan Fedotov,**[2]
Jörg Hoffmann,[1] **Rupak Majumdar,**[2] **Wheeler Ruml**[3]
[1] Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
[2] Max Planck Institute for Software Systems, Kaiserslautern, Germany
[3] Department of Computer Science, University of New Hampshire, USA
{fickert, hoffmann}@cs.uni-saarland.de, {gavran, ivanan, rupak}@mpi-sws.org, ruml@cs.unh.edu

Abstract

The need to replan arises in many applications. However, in the context of planning as heuristic search, it raises an annoying problem: if the previous plan is still executing, what should the new plan search take as its initial state? If it were possible to accurately predict how long replanning would take, it would be easy to find the appropriate state at which control will transfer from the previous plan to the new one. But as planning problems can vary enormously in their difficulty, this prediction can be challenging. Many current systems merely use a manually chosen constant duration. In this paper, we show how such ad hoc solutions can be avoided by integrating the choice of the appropriate initial state into the search process itself. The search is initialized with multiple candidate initial states and a time-aware evaluation function is used to prefer plans whose total goal achievement time is minimal. Experimental results show that this approach yields better behavior than either guessing a constant or trying to predict replanning time in advance. By making replanning more effective and easier to implement, this work aids in creating planning systems that can better handle the inevitable exigencies of real-world execution.

Introduction

After plan execution has begun, the need to replan can arise for many reasons. Perhaps the agent's goal has changed (Molineaux, Klenk, and Aha 2010) or additional goals have become known (Benton, Do, and Ruml 2007). Perhaps the environment, or the agent's perception of it, has changed, possibly giving rise to better ways of achieving the agent's goals (Cashmore et al. 2016). Or perhaps the original plan was constructed in haste, and now that execution is underway, the agent has the leisure to try to find a better plan to switch to (Likhachev, Gordon, and Thrun 2003). Regardless of cause, if the agent is using a planning technique such as forward state-space search, replanning necessitates choosing an initial state for the search. This small detail, often glossed over in previous work on replanning, raises a vexing problem: with execution underway, this initial state must be one far enough along the current plan that the agent will not encounter it until the replanning process has finished, to allow transitioning to the new plan. But choosing an initial state that is too far along the old plan risks inefficiency: the agent's actions will not reflect the new information until the state is reached, possibly causing it to miss opportunities.

The most common solution in current systems appears to be to always choose a transition point that is a fixed time ahead in the future, either as part of system design (Gregory et al. 2002; McGann et al. 2007) or through an estimate for the replanning time (Likhachev, Gordon, and Thrun 2003; Ruml et al. 2011). But this is at odds with the purpose of automated planning, which is to enable an agent to handle a variety of problems and situations with a single algorithm. In this paper, we introduce a principled approach to the problem of choosing the initial state for online replanning, which we call the Multiple Initial State Technique (MIST). MIST integrates this choice into the search itself, as only the search has the proper information about the necessary trade-offs.

In MIST, the open list is initialized with multiple potential initial states, rather than just one. These states represent speculations on the state in which the agent might be once the search is done. Open nodes are prioritized by their estimated goal achievement time, taking into account both the makespan of the resulting plan and an estimate of when the planning process itself will finish. In this way, MIST reasons online during search about which initial state is most promising to explore.

In MIST, once the execution goes past a state, that state is no longer viable as an initial state. We also present a variant of MIST that allows the planner to consider initial states that may have already been passed but that are still reachable. For both variants, we prove that, under suitable conditions on the heuristic functions, the first solution MIST finds is better than any other one it might find by continuing its search.

To assess the effectiveness of MIST in a concrete yet domain-independent way, we implement it in the Fast Downward planner (Helmert 2006) and extend a set of benchmarks from the International Planning Competition (IPC) to our replanning setting. We find that MIST yields better agent behavior, in the sense of achieving its goals more quickly, than either using a fixed constant or trying to predict replanning time in advance.

Previous Work

One example of the need for replanning in an industrial context is on-line goal arrival in manufacturing. (Ruml et al.

[*] These authors contributed equally to this work.

2011) address printing systems, where additional pages to print arrive asynchronously. They use a hand-tuned constant to represent the maximum planning time and predict when the new plan can start; if this bound is exceeded, the planning process is interrupted and re-tried later when the plant is assumed to be less busy and easier to plan for. Similarly, but in the context of robot navigation, (Likhachev, Gordon, and Thrun 2003) use a predefined constant to predict the time when a new plan will be found during an anytime search, allowing them to select the state at which the new plan will take control from the currently executing plan. Systems designed for spacecraft control commonly incorporate replanning, often by fixing a latency within which the planner must react (Gregory et al. 2002; McGann et al. 2007), thereby fixing the initial state for the revised plan. In our work, we aim for a more principled and flexible approach that eschews hand-tuned constants.

In some situations, it can be assumed that replanning time will be negligible (Knight et al. 2001; Ma et al. 2017, 2019). This again allows selection of the initial state in advance, in particular stopping the execution as early as possible when a new plan can be assumed to be available. We take that approach as one of our baselines.

The setting in which replanning is driven by new goals appearing online has been called *continual online planning (COP)* (Benton, Do, and Ruml 2007; Lemons et al. 2010; Burns et al. 2012). In that work, COP tasks are defined as Markov Decision Processes where additional goals arrive stochastically at each time step and so world states are extended with the current goal set. In the present paper, we leave aside any assumptions about goal arrival distributions and focus on the fundamental question of how the plan search for the combination of an old goal and a new goal can be formulated in the context of a currently executing plan.

In what is perhaps the closest work to ours, (Cashmore et al. 2019) consider replanning in the context of plan execution. They use a feature of PDDL2.2, called timed initial literals (TILs), to represent non-interruptible commitments of the currently executing plan and an ad hoc extension of the domain model, called a 'bail-out action generator,' to provide alternatives for interruptible actions. Their approach always discards the remainder of the original plan, thus avoiding the need to choose an initial state but assuming that the system can safely idle if necessary during replanning. Furthermore, their approach is inherently tied to PDDL planning, focusing on the intricacies of interrupting individual actions, rather than a generic solution for state-space search, as we provide in the present paper.

The concern of time passing during planning has been addressed by (Cashmore et al. 2018), although without the complexity of a concurrently executing plan. Their main focus lies on finding plans that respect external temporal constraints, such as a deadline for arriving at a bus stop. An associated line of more theoretical work (Shperberg et al. 2019; Coles et al. 2019) takes as its objective to maximize plan feasibility, leaving aside the notion of overall goal achievement time that we address here. There are also several time-aware heuristic search methods, such as Bugsy (Burns, Ruml, and Do 2013) and Deadline-Aware Search (Dionne, Thayer, and Ruml 2011). However, these methods do not consider concurrent plan execution and merely search from a given static initial state.

Problem Definition

Although MIST applies to any setting in which planning occurs in the context of ongoing execution, for concreteness we investigate MIST in classical planning with online goal arrival. The focus is on the moment when a new task arrives while the agent is already executing its *current plan*. This is equivalent to a series of arriving jobs assuming that each task arrives after the previous planning phase is done.

Background

We consider the *finite-domain representation (FDR)* (Bäckström and Nebel 1995; Helmert 2009) for classical planning tasks:

Definition 1 *A **planning task** is a tuple* (V, A, c, s_0, s_*)*:*

- V *is a finite set of* state variables*, each with a finite domain of possible values,*
- A *is a finite set of* actions*. Each action* a *is a pair* (pre_a, eff_a) *of partial variable assignments called* preconditions *and* effects*,*
- $c\colon A \to \mathbb{R}$ *is a function assigning a cost to every action,*
- s_0 *is the* initial state *(complete variable assignment),*
- s_* *is a partial variable assignment called the* goal*.*

We denote the set of all complete variable assignments, or *states*, by $\mathcal{S}$. A partial assignment p is said to be *compliant* with a state $s \in S$ (denoted by $p \subseteq s$) if there is no variable in the domain of p to which p and s assign different values. An action $a \in A$ can only be applied to a state $s \in S$ if $pre_a \subseteq s$. The outcome of that application is state $s[\![a]\!]$ that is the same as s, except that the variables in the domain of the partial assignment eff_a are changed accordingly.

A solution (*plan*) to a planning task is a sequence of actions $\pi = \langle a_1, a_2, \dots, a_n \rangle$ with the overall cost $C(\pi) = \sum_{i=1}^{n} c(a_i)$ leading from s_0 to a state compliant with s_*.

In what follows, the action costs c will be interpreted as the time required to execute them.

Continual Online Planning Tasks

We express the notion of continual online planning in the classical planning setting by extending tasks with a second goal, assumed to arrive during the execution of the plan for the original goal.

Definition 2 *A **continual online planning (COP) task** is a tuple* $(V, A, c, s_*^{OLD}, s_*^{NEW}, s_0, \pi_{s_0, s_*^{OLD}})$*:*

- V*, A, and c are as before,*
- s_*^{OLD} *is a partial variable assignment called the* old goal*,*
- s_*^{NEW} *is a partial variable assignment called the* new goal*,*
- s_0 *is the agent's state at the time when* s_*^{NEW} *appeared,*
- $\pi_{s_0, s_*^{OLD}} = \langle a_1 a_2 \dots a_n \rangle$ *is a sequence of actions, taking the agent from the state* s_0 *to a state compliant with the old goal* s_*^{OLD} *(the agent's* current plan*).*

Note that, while s_0 is not necessarily the first state of the agent's original plan for s_*^{OLD}, we disregard all states before the arrival of s_*^{NEW} and focus on the suffix of the original plan that can still be changed. We assume the actions to be non-interruptible: if s_*^{NEW} appeared during the execution of an action, s_0 is the state at the end of the action.

We assume for simplicity that s_*^{OLD} and s_*^{NEW} are not in direct conflict, i.e., if both are defined on the same variable then the assigned value is identical. A solution to a COP task is a plan π consisting of two parts: a prefix of $\pi_{s_0,s_*^{\text{OLD}}}$ and the newly planned extension. There must exist $1 \leq j \leq n$ such that $\pi = \langle a_1 a_2 \dots a_j b_1 \dots b_m \rangle$. If the extension, denoted by $b_1 \dots b_m$, is not empty, we call the state in which b_1 will be applied the *deviation state*. The state to which the plan π takes the agent must be compliant with both s_*^{OLD} and s_*^{NEW}. A solution is said to be optimal if it minimizes the total planning and execution time, i.e., the time from the arrival of the new job s_*^{NEW} to the end of the execution of π.

In order to achieve s_*^{NEW}, it may be useful to deviate from $\pi_{s_0,s_*^{\text{OLD}}}$ early. For example, such a situation occurs if a warehouse robot is moving back to its home base to deliver a good for s_*^{OLD}, while s_*^{NEW} requires picking up another good close to where the robot was when s_*^{NEW} arrived.

Observe that such indirect conflicts can be characterized by the degree to which $\pi_{s_0,s_*^{\text{OLD}}}$ is useful for the combined goal $s_*^{\text{OLD}} \cup s_*^{\text{NEW}}$. At one extreme end, $\pi_{s_0,s_*^{\text{OLD}}}$ is a prefix of an optimal plan for $s_*^{\text{OLD}} \cup s_*^{\text{NEW}}$. In this case, it is best to complete the execution of $\pi_{s_0,s_*^{\text{OLD}}}$. At the other extreme end, $\pi_{s_0,s_*^{\text{OLD}}}$ is not useful at all, i.e., no optimal plan for $s_*^{\text{OLD}} \cup s_*^{\text{NEW}}$ shares a non-empty prefix with $\pi_{s_0,s_*^{\text{OLD}}}$. In that case, it is best to stop $\pi_{s_0,s_*^{\text{OLD}}}$ immediately. However, it is not known upfront where between the two extremes to position the planner. Our purpose is to address this trade-off automatically and in full generality.

The Multiple Initial State Technique

Algorithm 1 shows the pseudocode of the MIST algorithm. Its structure closely resembles that of A*, with a few important modifications. In contrast to A*, MIST uses a set of potential initial states that we call *reference states*. The reference states are sampled from the plan $\pi_{s_0,s_*^{\text{OLD}}}$ towards the original goal, and are passed to the algorithm as a parameter R. These starting states are different "guesses" on where the agent will be when planning finishes, and they are the potential deviation states for the overall plan.

The open list (*open*) is initialized using the reference states (line 3). Each element of *open* is a pair of the search node and its corresponding reference state (a candidate for the deviation state). Each newly created search node inherits the reference state of its parent (line 13).

As in A*, nodes in the open list are expanded in a best-first order according to f, and put into the closed list afterwards. When a node is expanded, its successors are inserted into the open list if they are new or replaced if they are reached with a lower g-value than before (line 14). As time passes and execution of the current plan progresses, search nodes whose corresponding plan deviates from the executed actions become invalid and are discarded (line 7). Line 8 checks the termination condition, reflecting Definition 2.

Algorithm 1 MIST

```
 1: procedure MIST($s_0$, $s_*^{\text{OLD}}$, $h$, $s_*^{\text{NEW}}$, $\pi_{s_0,s_*^{\text{OLD}}}$, $R$)
 2:     $\gamma \leftarrow 0$; closed $\leftarrow \emptyset$
 3:     open $\leftarrow \{(r, r) \mid r \in R\}$
 4:     while open $\neq \emptyset$ do
 5:         $(s, ref_s) \leftarrow \arg\min_{(t, ref_t) \in \text{open}} f(t, ref_t, \gamma)$
 6:         if $(s, ref_s)$ is not consistent with
                the state of the execution then
 7:             discard $(s, ref_s)$
 8:         if $(s_*^{\text{OLD}} \cup s_*^{\text{NEW}}) \subseteq s$ then
 9:             return path to s
10:         closed $\leftarrow$ closed $\cup \{(s, ref_s)\}$
11:         $\gamma \leftarrow \gamma + 1$
12:         for $t \in$ successors(s) do
13:             $ref_t \leftarrow ref_s$
14:             if $((t, ref_t) \notin$ (open $\cup$ closed) or
                    $g_{ref_t}(t) < g^{old}_{ref_t}(t)$ then
15:                 open $\leftarrow$ open $\cup \{(t, ref_t)\}$
16:     return fail
```

Finally, the most important difference is the open list ordering function f. Given a state s, its reference state ref_s, and the number of expansions made by the algorithm so far γ (s_0 is treated as a default parameter), MIST uses $f(s, ref_s, \gamma) = C(\pi_{s_0,ref_s}) + g_{ref_s}(s) + h(s) + os(s, ref_s, \gamma)$. The first part, $C(\pi_{s_0,ref_s})$, represents the time required to move from the initial state s_0 to the reference state that was used to reach s along the execution path. The second part, $g_{ref_s}(s) + h(s)$, is the same as in A*: following our interpretation of action costs as their durations, it combines the time needed to get from the reference state ref_s to the state under consideration s with the estimated time to reach the goal from s. The third part, the overshot function os, models a state becoming irrelevant ($f = \infty$) as the execution passes through it before the estimated end of planning. It is defined as:

$$os(s, ref_s, \gamma) = \begin{cases} 0 & \text{if } (\gamma + \eta(s)) \cdot t_{exp} \leq C(\pi_{s_0,ref_s}) \\ \infty & \text{otherwise} \end{cases}$$

where $\eta(s)$ estimates the remaining number of expansions until a plan is found, t_{exp} is the time per expansion to translate expansions to execution time, and $C(\pi_{s_0,ref_s})$ is the time to reach ref_s along $\pi_{s_0,s_*^{\text{OLD}}}$. We will discuss below how to obtain such estimates by adapting prior work (Thayer and Ruml 2009; Burns, Ruml, and Do 2013).

Having γ as an argument for f has an interesting consequence: it now matters when the function f is evaluated for the relative order of the nodes in *open*. In practice, we do not re-evaluate f on all the nodes in the open list each time the best element is retrieved (line 5). Instead, we approximate the value of the f-function by keeping the search nodes sorted only by $g + h$, but separately for each reference state. Subsequently, we do the full evaluation only to select the next reference state for which a node should be expanded using the nodes with minimal $g + h$ for each reference state.

This approximation is justified by the fact that a changed value of γ affects all the nodes corresponding to the same reference state equally. The loss of precision comes from disregarding differences in η. In preliminary experiments, we also tried a recomputation strategy in exponentially increasing intervals as used in Bugsy (Burns, Ruml, and Do 2013) and found the difference in solution quality to be negligible compared to this approximation strategy.

Another practical consideration is that we can safely prune a search node (s, ref_s) if s is a reference state itself that is reached by $\pi_{s_0, s_*^{\mathrm{OLD}}}$ after passing through ref_s and $C(\pi_{s_0,\mathit{ref}_s}) + g_{\mathit{ref}_s}(s) \geq C(\pi_{s_0,s})$ holds.

Theoretical Properties

A* is guaranteed to find an optimal solution, provided that the heuristic function is admissible (and nodes can be reopened). A similar guarantee can not be given for MIST. The essential difference between the two settings (and thus necessarily between the two algorithms) is that in an offline setting, the exploration of the state space during the planning phase comes at no cost. On the other hand, in an online setting, exploring a part of the search space that is not going to be used in the solution can decrease the quality of the final plan, since that time was not used effectively.

Consider a situation where the only optimal plan deviates at the reference state r, and expanding all the nodes on that plan takes exactly the time that the agent needs to reach r. Expanding any other nodes will make MIST miss this path. Hence, unless the heuristic functions h and η were perfect, there is no guarantee that MIST will find an optimal solution.

With optimality out of reach, we can prove a simpler property: the stopping criterion of MIST is a correct one. MIST stops the search as soon as the first state compliant with both of its goals is found, which raises the question of whether there is some trade-off between continuing the search and the quality of the solution. We show that continuing the search can not result in a better plan, assuming the heuristic functions h and η are admissible.

We will use $h^*(s)$ to denote the true value of the cost to reach the goal from s, and $\eta^*(s)$ to denote the number of expansions from s to the end of planning. Following the same notation style, $f^*(s, \mathit{ref}_s, \gamma_s)$ and $os^*(s, \mathit{ref}_s, \gamma_s)$ denote the functions f respectively os calculated using $h^*(s)$ and $\eta^*(s)$ instead of the heuristics h and η. We are using the notation $\gamma = \gamma_s$ to indicate that the third argument of the f-function is the value of γ when the node s was explored.

Theorem 1 *Let h be admissible with respect to planned execution time and η admissible with respect to the number of expansions. Let $\sigma_1 = s_0, s_1, \ldots, s_i, p_1, p_2 \ldots p_m$ be the sequence of states corresponding to the first solution π_1 found by MIST (with the deviation state s_i). Assume the algorithm continued the search and found another solution with the state sequence $\sigma_2 = s_0, s_1, \ldots, s_j, q_1, q_2, \ldots, q_n$ (with the deviation state s_j). Then $f^*(p_m, s_i, \gamma_{p_m}) \leq f^*(q_n, s_j, \gamma_{q_n})$.*

Proof: Our proof follows that of a similar property of Bugsy (Burns, Ruml, and Do 2013).

$$\begin{aligned}
& f^*(p_m, s_i, \gamma_{p_m}) \\
&= C(\pi_{s_0,s_i}) + g_{s_i}(p_m) + os^*(p_m, s_i, \gamma_{p_m}) \\
&= f(p_m, s_i, \gamma_{p_m}) && (1) \\
&\leq f(q_l, s_j, \gamma_{p_m}) && (2) \\
&= C(\pi_{s_0,s_j}) + g_{s_j}(q_l) + h(q_l) + os(q_l, s_j, \gamma_{p_m}) \\
&\leq C(\pi_{s_0,s_j}) + g_{s_j}(q_l) + h^*(q_l) + os^*(q_l, s_j, \gamma_{p_m}) && (3) \\
&\leq C(\pi_{s_0,s_j}) + g_{s_j}(q_l) + h^*(q_l) + os^*(q_l, s_j, \gamma_{q_l}) && (4) \\
&\leq C(\pi_{s_0,s_j}) + g_{s_j}(q_n) + os^*(q_n, s_j, \gamma_{q_n}) && (5) \\
&= f^*(q_n, s_j, \gamma_{q_n})
\end{aligned}$$

The true cost of the solution π_1 is $f^*(p_m, s_i, \gamma_{p_m}) = C(\pi_{s_0,s_i}) + g_{s_i}(p_m) + os^*(p_m, s_i, \gamma_{p_m})$. Following the search structure of MIST, at some point we chose to expand p_m. Since p_m is the last state on the path and our heuristic functions are admissible, the true cost f^* is equal to the cost function f (equality 1). Inequality 2 comes from our choice of the state p_m over some state q_l from σ_2.

From the admissibility of h and os (which follows from the admissibility of η) with respect to h^* and os^*, we get inequality 3. The value of os^* for some later point γ_{q_l} when exploring q_l, would be greater or equal to the value for the same state q_l and the reference state s_j at time point γ_{p_m} (inequality 4). This follows from the definition of os^*, which is monotonic when observed as a function of its third argument (time passed so far).

Finally, inequality 5 comes from (a) $\gamma_{q_n} + \eta^*(q_n) = \gamma_{q_l} + \eta^*(q_l)$ and (b) $g_{s_j}(q_l) + h^*(q_l) \leq g_{s_j}(q_n)$. Both sides of equality (a) are equal to the overall planning time. Inequality (b) comes from the fact that $h^*(q_l)$ is the smallest cost from q_l to a goal state. Since that goal state is not necessarily q_n, we get the inequality. □

MIST for Recoverable Tasks

Depending on how the reference states are selected among the initially computed plan, situations may arise in which MIST's replanning strategy incurs a large cost. Consider the example from Figure 1. The truck is heading to location L when a new goal to pick up and deliver a package to L appears. It may happen that the planner is not able to find a plan before the truck has already passed the first reference state R_1, which is closest to the package. MIST requires the truck to continue following the current plan until the next reference state R_2 and only then turn back to pick up the package. Depending on the distance between R_1 and R_2, this can cause an arbitrarily large cost penalty compared to turning around at some intermediate location. Even if reference states are frequent, MIST can still incur a large penalty if it repeatedly misses its predicted planning time.

An engineering solution addressing these issues could be to pause the execution for some time if planning is expected to finish soon. Here, we instead suggest a variant of MIST,

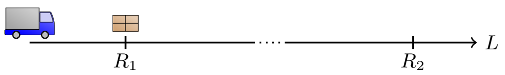

Figure 1: A truck executing a plan when a new goal appears.

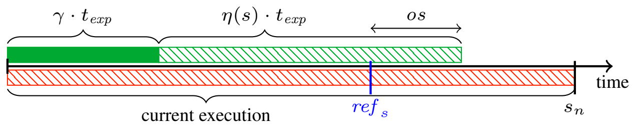

Figure 2: Overshot.

that we call MIST$_{\text{rec}}$, as an algorithmic solution for a class of COP tasks that satisfy the following form of recoverability.

Definition 3 *For a COP task $(V, A, c, s_*^{OLD}, s_*^{NEW}, s_0, \pi_{s_0,s_*^{OLD}})$, with $s_0,\dots,s_n$ being the state sequence induced by $\pi_{s_0,s_*^{OLD}}$, we denote the action subsequence taking the agent from s_i to s_j, with $i < j$, by $\vec{\alpha}_{i,j}$. We call the task **recoverable** if, for every such $\vec{\alpha}_{i,j}$, there exists an action sequence $\overleftarrow{\alpha}_{i,j}$ such that every variable/value pair in s_i that appears as a precondition or goal also holds in $s_j[\![\overleftarrow{\alpha}_{i,j}]\!]$.*

This restriction gives the planner more room for error in the prediction of when re-planning will terminate: When a reference state s is passed, the planner may still finish computing a plan for s, because there is a recovery sequence that brings the agent to a state where that plan is applicable.

Many COP applications are naturally recoverable. Examples include warehouse logistics and various types of manufacturing problems. Furthermore, recoverability relates to known notions of invertibility and undoability, and prior works have established methods to test these properties (Hoffmann 2005; Daum et al. 2016). Our solution considers the recoverability sequence to be known. In our experiments, we focus on domains where each action has an immediate inverse action of the same cost.

In order to adapt Algorithm 1, two things need to be changed. First, the states of $\pi_{s_0,s_*^{\text{OLD}}}$ that the agent's execution already went through should not be discarded, because the agent can still come back to them. Second, we have to change the definition of the function f to reflect the possibility of returning to a reference state.

Reminding ourselves, f is defined as a sum of three parts: $C(\pi_{s_0,ref_s}) + [g_{ref_s}(s) + h(s)] + os(s, ref_s, \gamma)$, with os being either 0 (if planning is estimated to finish before the execution reaches ref_s), or infinity (otherwise). We adapt os to consider the possibility of using the recovery sequence to move back to ref_s, redefining it as $os(s, ref_s, \gamma) = C(\vec{\alpha}) + C(\overleftarrow{\alpha}) + \max((\gamma + \eta(s)) \cdot t_{exp} - C(\pi_{s_0,s_*^{\text{OLD}}}), 0)$. Like before, os evaluates to 0 if planning is expected to finish in time. Otherwise, it now describes the additional execution time incurred by moving past the reference state ($\vec{\alpha}$) and back ($\overleftarrow{\alpha}$). If planning takes longer than total execution of $\pi_{s_0,s_*^{\text{OLD}}}$, then the agent will additionally have to wait in s_n, the last state of $\pi_{s_0,s_*^{\text{OLD}}}$ (described by the last term of os).

Consider Figure 2: The dashed red bar denotes the time needed to execute the current plan leading to $s_n \supseteq s_*^{\text{OLD}}$. The green bar labeled by $\gamma \cdot t_{\text{exp}}$ is the time spent planning so far, and the dashed green bar ($\eta(s) \cdot t_{\text{exp}}$) shows the estimation on when the planning will finish. In the illustrated example, the planning time is estimated to exceed the time when the selected reference state ref_s is reached. The os function describes this additional execution time, plus the time it takes to go back to ref_s.

Lemma 1 *Assume that η is admissible, and that for a path α that is a prefix of a path α' it holds that $C(\vec{\alpha}) + C(\overleftarrow{\alpha}) \leq C(\vec{\alpha'}) + C(\overleftarrow{\alpha'})$ (well-behaved recovery paths). Then os is admissible, i.e. $os(s, ref_s, \gamma) \leq os^*(s, ref_s, \gamma)$.*

Proof: Let $\vec{\alpha}$ be the subsequence of actions on $\pi_{s_0,s_*^{\text{OLD}}}$ taking the agent from the reference state ref_s to the state in which it would be at time $\gamma + \eta(s) \cdot t_{exp}$, and let $\vec{\alpha^*}$ be the subsequence of actions to the state at time $\gamma + \eta^*(s) \cdot t_{exp}$. Since $\eta \leq \eta^*$, $\vec{\alpha}$ must be a subsequence of $\vec{\alpha^*}$. With the assumption of well-behaved recovery paths, we have $C(\vec{\alpha}) + C(\overleftarrow{\alpha}) \leq C(\vec{\alpha^*}) + C(\overleftarrow{\alpha^*})$, and thus $os \leq os^*$. □

Since os is again admissible, the proof of Theorem 1 also applies to MIST$_{\text{rec}}$.

Consider again the logistics example at the beginning of this subsection. While MIST must continue along the plan until reaching R_2, MIST$_{\text{rec}}$ will be able to finish computing a new plan from the (already passed) reference state R_1, and can turn around without having to go to R_2 first.

Experiments

We implemented MIST in Fast Downward (Helmert 2006). As explained earlier, in our implementation, we use a standard A* open list for each reference state, using the MIST extensions to the f-function only to select the open list to be used for the next expansion to avoid having to re-sort the open list. For MIST$_{\text{rec}}$, our implementation assumes that each action has an inverse action with the same cost.

Like Bugsy, we estimate the remaining number of expansions as $\eta = delay * d$ (Burns, Ruml, and Do 2013; Dionne, Thayer, and Ruml 2011), where *delay* is the (moving) average number of expansions between inserting a node into the open list and expanding it, and d is an estimation of the remaining steps to the goal (like h, but ignoring action costs). The expansion delay is important to counteract *search vacillation* (Dionne, Thayer, and Ruml 2011), referring to the search fluctuating between different solution paths and, in our case, potentially of different reference states. For d, we don't use the distance estimate of the current state, but instead the minimal distance of any evaluated state that corresponds to the considered reference state to make the planning time estimations more stable.

Our key performance metric is the *goal achievement time* (GAT), i.e. overall time for online planning and execution, measured from the moment when the new goals appear. We measure this time as a number of expansions to make the experiments more robust. Action costs are translated into execution time using an instance-specific factor from cost to expansions (we give more details in the next subsection).

In all experiments, the popular FF heuristic (Hoffmann and Nebel 2001) is used to guide the search. For the expansion delay, we use a moving average over the last 100 expansions. The experiments were run on a cluster of 2.20 GHz Intel Xeon E5-2660 CPUs. The time and memory limits were set to 30 minutes and 4 GB, respectively.

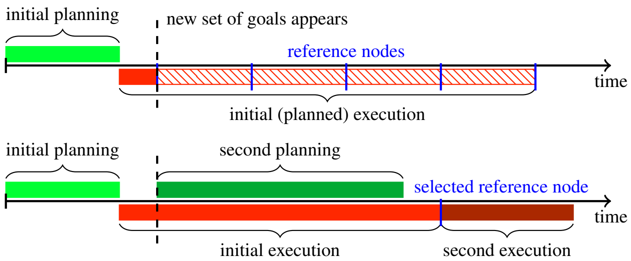

Figure 3: Generating benchmark instances.

Benchmarks

We adapted the IPC domains Elevators, Logistics, Rovers, Transport, and VisitAll to our setting, as representatives of applications where (i) goals are of an additive nature and there are no conflicts between them and (ii) all action sequences $\vec{\alpha}$ have a recovery sequence $\overleftarrow{\alpha}$ with the same cost. Criterion (ii) is required for our implementation of MIST$_{\text{rec}}$. We furthermore experiment with Tidybot, which we adapted to satisfy (ii). In Tidybot, there are cases where objects are placed behind each other, and the robot cannot reach behind the object in the front. We added an un-finish action to ensure recoverability. However, previously finished objects must be picked up again in these cases, necessitating the planner to falsify and re-achieve previously achieved goals. We assume actions to be non-interruptible.

In some of our benchmark domains, recovery sequences with lower cost can exist. For example, there could be shortcuts to inverting the agent's movements, and in Rovers photos would not need to be "un-taken". In such cases, our implementation of MIST$_{\text{rec}}$ is pessimistic and more practical implementations may achieve lower plan costs.

The instances were adapted by splitting the set of goals in two: the first half is available in the beginning, and the other one becomes available later. The second set of goals is scheduled to appear during the execution of the first computed plan to obtain interesting instances. The process is illustrated in Figure 3. The initially computed plan is being executed as a new job arrives (Figure 3, top). Here, the planner considers 5 reference states as potential initial states for the new plan. The planner has computed an updated plan that starts from the second-to-last reference state (Figure 3, bottom). The initial plan is executed until that point before switching to the new plan. The goal achievement time is the time from the start of the second planning phase to the end of the overall execution.

In order to obtain interesting benchmark instances, we tried to ensure that the second planning phase starts and ends during the first planned execution. Thus, we generated the instances such that the second set of goals appears after 10% of the initial plan is executed. Furthermore, we estimated the length of the second planning phase by running the planner offline for both goals from the original start state, and used that to generate different experimental setups where the second planning phase is estimated to end at $E = 0.2, 0.3, \ldots, 0.9$ of the initially planned execution. This is achieved by adjusting the factor for the translation of the action cost to execution time, thereby changing the duration of the initial execution.

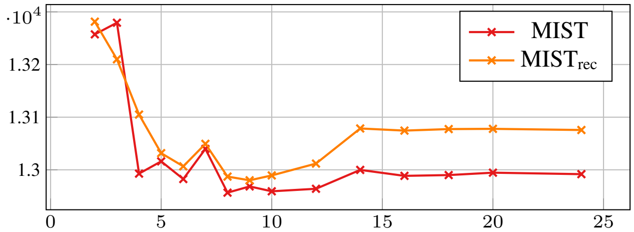

Figure 4: GAT as geom. mean over all instances (Y-axis) for MIST with different numbers of reference states (X-axis).

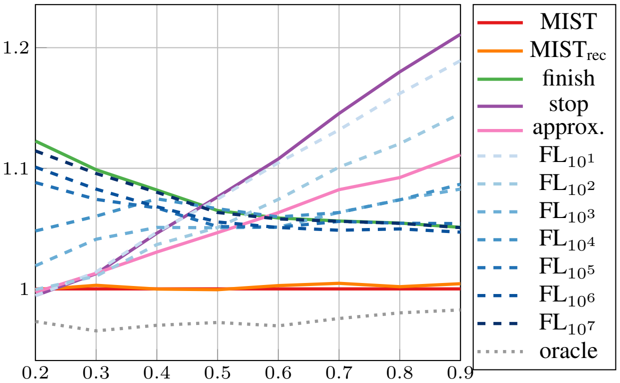

Figure 5: GAT as geom. mean over all instances relative to MIST (Y-axis) for $E = 0.2, 0.3, \ldots, 0.9$ (X-axis).

Results

We compare MIST to the following baselines:

- *finish*: Finish execution and plan only for the new goals.
- *stop*: Stop execution and re-plan from the current state.
- *approximate*: Approximate the duration of the re-planning phase, and use the state where the agent is expected to be at that time as the deviation state. We use the same estimation for the number of expansions as MIST, i.e. $\eta(m) = \mathit{delay} * d(m)$, using the average expansion delay from the initial planning phase and the estimated distance of the current state.
- *fixed latency*: Stop execution at a fixed point in time (we test values of $10^1, 10^2, \ldots, 10^7$ expansions for this time point). We also consider a theoretical oracle configuration that chooses the best-performing time point to stop the execution (out of the tested values) per instance.

MIST has one important parameter: the selection of the reference states. In our implementation, we set a number of reference states n_R, which are then selected in uniform intervals from the current plan. Figure 4 shows the goal achievement time (in number of expansions) for different values of n_R across our full benchmark set. If there are too few reference states, the algorithm does not have the best

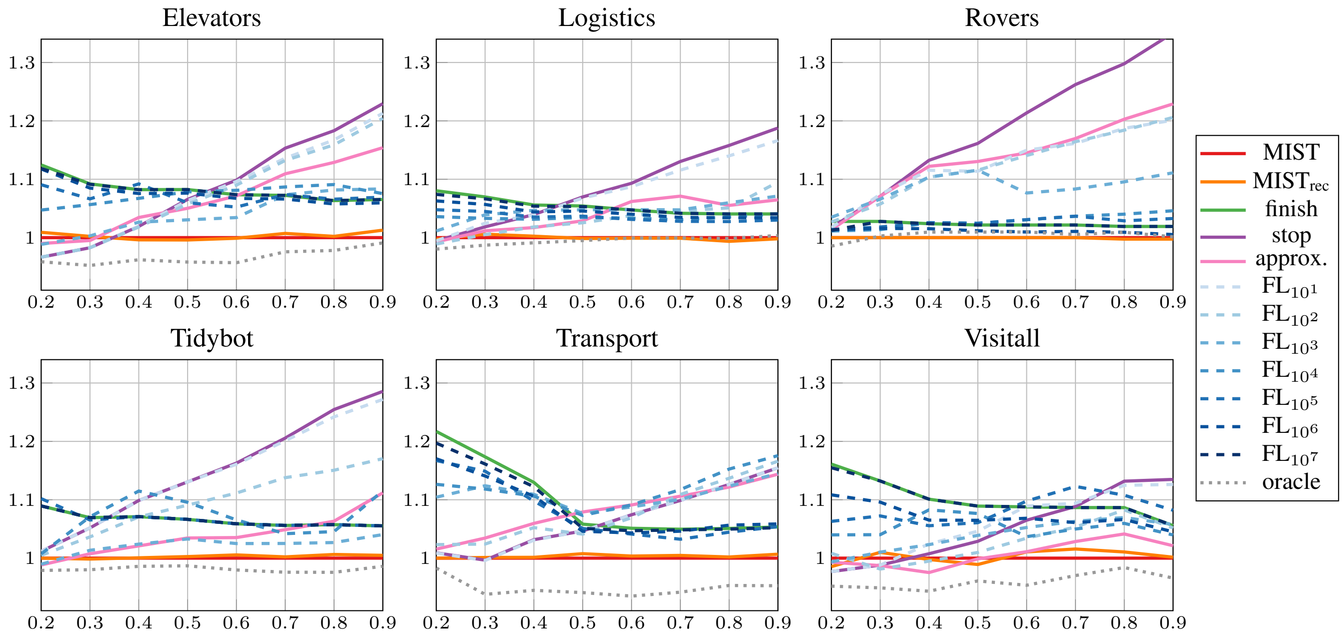

Figure 6: GAT relative to MIST (Y-axis) for $E = 0.2, 0.3, \ldots, 0.9$ (X-axis).

starting point for the next plan available. On the other hand, the performance also decreases if too many reference states are used, as it becomes more difficult to settle on the most promising one quickly (especially if the planning time estimation is not very accurate). Across the tested numbers of reference states, MIST chooses nodes for expansion corresponding to the reference state which is used for the solution 38% of the time on average, more for fewer reference states (55% for $n_R = 3$), and less the more reference states are used (30% for $n_R = 24$). The overall best results are obtained with $n_R = 8$ for MIST and $n_R = 9$ for MIST$_{\text{rec}}$, and we use these settings for the remaining experiments.

Figure 5 shows the relative goal achievement time compared to MIST for the considered algorithms for different expected end points of the second planning phase. If the planning time is very short compared to the execution time (small values of E), stopping the execution as soon as possible works well, but loses out compared to MIST if planning is non-trivial ($E > 0.2$). As expected, finishing the execution becomes better with increasing expected planning times, though MIST always performs better. The fixed latency configurations offer some interpolation between the two extremes of stopping or finishing the execution. Given the diversity of our benchmark set, a fixed latency can not accurately predict the planning time, and these configurations are outperformed by MIST. The approximation baseline also works well for short planning times, but is prone to overestimate. On average, MIST reduces the goal achievement time by 8.6% compared to stopping and re-planning immediately, by 6.8% compared to finishing the planned execution, and by 5.1% compared to approximating the re-planning time.

Figure 6 gives more insight into the individual domains. The observations from the overall results hold across most domains, with minor exceptions. On VisitAll, the *approximate* baseline comes very close to MIST on average, beating it for some values of E. This can can be attributed to the planning time estimation being more accurate. While that also helps MIST to select the correct reference state to expand towards more frequently (46% of the time compared to 35% on other domains), MIST can still suffer from the added overhead. In the Rovers domain, MIST and MIST$_{\text{rec}}$ outperform all competitors for all values of E, and may even beat the oracle (which can be inaccurate if the best deviation state is between two of the considered time points). Both *stop* and *approximate* perform particularly poorly in that domain, with up to 35% respectively 23% worse goal achievement time compared to MIST when considering large expected planning times. Conversely, *finish* comes close to MIST, which indicates that interrupting the execution while re-planning is particularly costly in that domain.

MIST$_{\text{rec}}$ and MIST exhibit similar performance. This suggests two conclusions. First, the way we generate testing instances averages out the edge cases in which MIST$_{\text{rec}}$ significantly outperforms MIST. Second, even though MIST$_{\text{rec}}$ does not prune reference nodes, it is able to effectively focus its search effort just as well as MIST.

Conclusion

We proposed a general technique, MIST, that addresses the problem of selecting an initial state for planning in the context of an already-executing plan by planning simultaneously for multiple potential initial states. We investigated its performance in the specific context of online goal arrival in domain-independent planning and found that MIST outperformed both the approximate and the fixed latency baseline.

Acknowledgments

Maximilian Fickert was funded by DFG grant 389792660 as part of TRR 248 – CPEC, see https://perspicuous-computing.science. We also gratefully acknowledge support from NSF-BSF grant 2008594.

References

Bäckström, C.; and Nebel, B. 1995. Complexity Results for SAS$^+$ Planning. *Computational Intelligence* 11(4): 625–655.

Benton, J.; Do, M. B.; and Ruml, W. 2007. A Simple Testbed for On-line Planning. In *Proceedings of the ICAPS-07 Workshop on Moving Planning and Scheduling Systems into the Real World.*

Burns, E.; Benton, J.; Ruml, W.; Yoon, S. W.; and Do, M. B. 2012. Anticipatory On-Line Planning. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS-12).*

Burns, E.; Ruml, W.; and Do, M. B. 2013. Heuristic Search When Time Matters. *Journal Artificial Intelligence Research* 47: 697–740.

Cashmore, M.; Coles, A.; Cserna, B.; Karpas, E.; Magazzeni, D.; and Ruml, W. 2018. Temporal Planning while the Clock Ticks. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018*, 39–46.

Cashmore, M.; Coles, A.; Cserna, B.; Karpas, E.; Magazzeni, D.; and Ruml, W. 2019. Replanning for Situated Robots. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018*, 665–673.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; and Ridder, B. 2016. Opportunistic Planning for Increased Plan Utility. In *Proceedings of the ICAPS-16 Workshop on Planning and Robotics (PlanRob 2016).*

Coles, A.; Shperberg, S.; Karpas, E.; Shimony, S.; and Ruml, W. 2019. Beyond Cost-to-go Estimates in Situated Temporal Planning. In *Proceedings of the ICAPS Workshop on Heuristics and Search for Domain-independent Planning (HSDIP).*

Daum, J.; Torralba, Á.; Hoffmann, J.; Haslum, P.; and Weber, I. 2016. Practical Undoability Checking via Contingent Planning. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016*, 106–114.

Dionne, A. J.; Thayer, J. T.; and Ruml, W. 2011. Deadline-Aware Search Using On-Line Measures of Behavior. In *Proceedings of the Fourth Annual Symposium on Combinatorial Search, SOCS 2011.*

Fox, M.; Gerevini, A.; Long, D.; and Serina, I. 2006. Plan Stability: Replanning versus Plan Repair. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS-06)*, 212–221.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning Through Stochastic Local Search and Temporal Action Graphs in LPG. *Journal of Artificial Intelligence Research* 20: 239–290.

Gregory, N. M.; Dorais, G. A.; Fry, C.; Levinson, R.; and Plaunt, C. 2002. IDEA: Planning at the Core of Autonomous Reactive Agents. In *in Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*.

Helmert, M. 2006. The Fast Downward Planning System. *J. Artif. Intell. Res.* 26: 191–246. doi:10.1613/jair.1705. URL https://doi.org/10.1613/jair.1705.

Helmert, M. 2009. Concise Finite-Domain Representations for PDDL Planning Tasks. *Artificial Intelligence* 173: 503–535.

Hoffmann, J. 2005. Where 'Ignoring Delete Lists' Works: Local Search Topology in Planning Benchmarks. *J. Artif. Intell. Res.* 24: 685–758. doi:10.1613/jair.1747. URL https://doi.org/10.1613/jair.1747.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research* 14: 253–302.

Knight, R.; Rabideau, G.; Chien, S. A.; Engelhardt, B.; and Sherwood, R. 2001. Casper: Space Exploration through Continuous Planning. *IEEE Intell. Syst.* 16(5): 70–75.

Lemons, S.; Benton, J.; Ruml, W.; Do, M. B.; and Yoon, S. W. 2010. Continual On-line Planning as Decision-Theoretic Incremental Heuristic Search. In *Embedded Reasoning, Papers from the 2010 AAAI Spring Symposium, Technical Report SS-10-04*.

Likhachev, M.; Gordon, G. J.; and Thrun, S. 2003. ARA*: Anytime A* with Provable Bounds on Sub-Optimality. In Thrun, S.; Saul, L. K.; and Schölkopf, B., eds., *Advances in Neural Information Processing Systems 16 [Neural Information Processing Systems, NIPS 2003*, 767–774. MIT Press.

Ma, H.; Hönig, W.; Kumar, T. S.; Ayanian, N.; and Koenig, S. 2019. Lifelong path planning with kinematic constraints for multi-agent pickup and delivery. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 7651–7658.

Ma, H.; Li, J.; Kumar, T. K. S.; and Koenig, S. 2017. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017*, 837–845.

McGann, C.; Py, F.; Rajan, K.; Thomas, H.; Henthorn, R.; and McEwen, R. 2007. T-REX: A model-based architecture for AUV control. In *3rd Workshop on Planning and Plan Execution for Real-World Systems*, volume 2007.

Molineaux, M.; Klenk, M.; and Aha, D. 2010. Goal-driven autonomy in a Navy strategy simulation. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Nebel, B.; and Koehler, J. 1995. Plan Reuse Versus Plan Generation: A Theoretical and Empirical Analysis. *Artif. Intell.* 76(1-2): 427–454. ISSN 0004-3702. doi:10.1016/0004-3702(94)00082-C. URL http://dx.doi.org/10.1016/0004-3702(94)00082-C.

Ruml, W.; Do, M. B.; Zhou, R.; and Fromherz, M. P. J. 2011. On-line Planning and Scheduling: An Application to Controlling Modular Printers. *Journal Artificial Intelligence Research* 40: 415–468.

Shperberg, S. S.; Coles, A.; Cserna, B.; Karpas, E.; Ruml, W.; and Shimony, S. E. 2019. Allocating Planning Effort When Actions Expire. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019*, 2371–2378.

Thayer, J. T.; and Ruml, W. 2009. Using Distance Estimates in Heuristic Search. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS-09)*.