Olisipo: A Probabilistic Approach to the Adaptable Execution of Deterministic Temporal Plans

Tomás Ribeiro
Institute for Systems and Robotics, Instituto Superior Tecnico, Lisbon, Portugal

Oscar Lima
DFKI German Research Center for Artificial Intelligence, Saabrücken, Germany

Michael Cashmore
University of Strathclyde, Glasgow, UK

Andrea Micheli
Fondazione Bruno Kessler, Trento, Italy

Rodrigo Ventura
Institute for Systems and Robotics, Instituto Superior Tecnico, Lisbon, Portugal

Abstract

The robust execution of a temporal plan in a perturbed environment is a problem that remains to be solved. Perturbed environments, such as the real world, are non-deterministic and filled with uncertainty. Hence, the execution of a temporal plan presents several challenges and the employed solution often consists of replanning when the execution fails. In this paper, we propose a novel algorithm, named Olisipo, which aims to maximise the probability of a successful execution of a temporal plan in perturbed environments. To achieve this, a probabilistic model is used in the execution of the plan, instead of in the building of the plan. This approach enables Olisipo to dynamically adapt the plan to changes in the environment. In addition to this, the execution of the plan is also adapted to the probability of successfully executing each action. Olisipo was compared to a simple dispatcher and it was shown that it consistently had a higher probability of successfully reaching a goal state in uncertain environments, performed fewer replans and also executed fewer actions. Hence, Olisipo offers a substantial improvement in performance for disturbed environments.

2012 ACM Subject Classification Computing methodologies → Robotic planning

Keywords and phrases Temporal Planning, Temporal Plan Execution, Robotics

Digital Object Identifier 10.4230/LIPIcs.TIME.2021.15

Supplementary Material *Software (Source Code)*: `https://github.com/TomasRibeiro96/Olisipo-planner`; archived at `swh:1:dir:21c4dfd9815b8d086ab714a5c687224529ad90af`

Acknowledgements The open access publication of this article was supported by the Alpen-Adria-Universität Klagenfurt, Austria.

1 Introduction

Automated temporal planning is the technology of choice when it comes to automatically controlling systems subject to temporal constraints such as deadlines. However, one of the key underlying assumptions of (classical) temporal planning is the perfect knowledge of a deterministic environment. Formalisms such as PDDL 2.1 [14] allow the modeling of planning problems where actions to be executed have known and deterministic effects on the system and on its environment. Moreover, the world is assumed to be static, meaning that when no action is executed by the agent, then no change is possible.

28th International Symposium on Temporal Representation and Reasoning (TIME 2021).
Editors: Carlo Combi, Johann Eder, and Mark Reynolds; Article No. 15; pp. 15:1–15:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

All these assumptions greatly simplify the reasoning needed to efficiently find a plan, but rarely apply to application scenarios. In robotics, for example, the environment where the machine operates is almost never static and there is uncertainty on the actual effect of the actions, due to interference of other actors (e.g. humans) or simply because some actions may fail due to lack of dexterity or because of mishaps.

In this paper, we want to tackle the problem of executing a temporal plan generated under the assumptions above in a system where these assumptions may not hold, exploiting probabilistic information to maximize the likelihood of achieving the original plan goal. To this end, we follow the execution schema proposed by [21] but we revise the on-line execution part. In particular, we propose an execution algorithm, called Olisipo, that exploits a probabilistically-sound reasoning on Dynamic Bayesian Networks (DBNs) to select the next action to be executed. The algorithm is able to cope with the need of skipping planned actions, of repeating actions, and of re-ordering the actions to achieve the original goal taking into account the observed state of the world. The former case can be useful to avoid operations that become useless (and may fail) due to the non-static nature of the environment: for example, consider a plan prescribing to open a door that was supposedly closed, while it is in fact open because a human forgot to close it. The second feature allows to repeat actions that might have failed to achieve their effects or whose effects have been canceled by the dynamic nature of the environment. Finally, re-ordering actions allows the handling of mutated temporal and precedence constraints.

We implemented the proposed approach in the ROSPlan [5] framework and we experimentally compare its merits against the original ROSPlan dispatcher, showing that the use of our approach is able to reduce the number of re-planning invocations and to increase the probability of successfully reach a goal state at the end of the execution.

1.1 Related Work

How to robustly execute and adapt plans at runtime is a central problem for the development of autonomous systems [18]. In fact, when a plan is being generated it is often impossible to foresee all the situations the controlled agent could encounter during execution; therefore, different approaches have been explored to tackle these issues. A common technique consists in relying on runtime monitoring to check that the conditions under which the plan was generated hold at runtime and if they do not a new plan is generated online; this approach is usually called replanning [13, 4]. Another direction is to model some of the uncertainties at planning time (most notably the action durations) as uncontrollable entities in the model and use planning algorithms that synthesize strategies to deal with such uncertainties [23, 7]. Finally, the approach we also pursue in this paper consists in generalizing a plan that was optimistically generated and allow the executor to adapt to unforeseen circumstances.

Probabilistic planning is a standard approach for planning with discrete uncertainty. An overview of approaches to probabilistic planning is provided in [17]. A common approach is to model the task as a Markov Decision Problem, which can handle the kind of exogenous and unforeseen events that we consider in this paper. Solutions to the MDPs typically formulate policies with finite horizon [2]. In contrast we rely on the efficiency and speed of deterministic search to find a solution that can then be generalized for an execution.

Some authors [1, 16, 12, 3] have focused on the problem of generalizing a temporal plan to make it more likely to succeed when executed. On the same line, some planners try to produce "flexible" plans, meaning that they reduce the commitments in the plan to a minimum to allow the exploitation of more degrees of freedom at execution time [15, 6]. Generally, a flexible temporal plan is represented as Simple Temporal Network (STN) [11], a Disjunctive

Temporal Network (DTN) [28] or as a Temporal Plan Network [20]; possibly with an explicit representation of the uncertainty [23]. The executor is then responsible for dispatching the prescribed events at the right time given the available observations [24, 9, 22]. However, one key assumption of these approaches is that the plan is never invalidated according to the original model, only valid re-scheduling is admitted.

In this paper, we take a radically different view: we want to exploit any PDDL 2.1 temporal planner and change the way in which the plan is dispatched by creating an action-selector that reasons over the uncertainties with the freedom to re-arrange, skip, and repeat the planned actions.

2 Algorithm

The Olisipo algorithm is an extension of the algorithm presented in [21]. The flow from planning to execution of Olisipo can be visualised in Figure 1. The approach is essentially composed of an offline and an online phase. The offline phase of Olisipo remains unchanged from [21], while the online phase is completely reformulated.

In summary, the offline phase, performed before starting plan execution, is responsible for converting a totally-ordered plan $\prod^{tt}$, generated by an external planner, to an adaptable partially-ordered plan $\prod'$. The adaptable partially-ordered plan is passed on to the online phase. The offline generation of the partially-ordered plan allows for a more flexible execution, namely by skipping and reordering actions during plan execution.

PDDL 2.1 planners output is commonly represented as time-triggered plans (e.g. [8, 26]). A time-triggered plan is a set of tuples $\langle t, a, d\rangle$, where a is an action, t is the time at which the action should start execution and d is the prescribed duration. An adaptable partially-ordered plan is defined below, and a complete description can be found in [21].

▶ **Definition 1.** *An **Adaptable Partially-Ordered Plan** $\prod'$ is the graph $\langle N, C\rangle$, where each node $n \in N$ represents either the plan start, an instantaneous action, or the start or end of a durative action; and each edge $c \in C$ represents a temporal relation: $x < time(n_1) - time(n_2) < y$ for $n_1, n_2 \in N$ and $x, y \in R$. Each edge is labelled as either **causal**, **interference**, or **action duration**. Action duration edges express the temporal constraints between the start and end of durative actions. Causal edges express temporal relationships inferred from the causal support between actions. Similarly, interference edges express the temporal relationships inferred from the interference between actions.*

The online phase has two sub-phases: the Plan Generation Sub-Phase (PGen) and the Dynamic Plan Execution Sub-Phase (PExec). PGen is used to generate the totally-ordered plan Γ with the highest probability of a successful execution, taking into consideration the current state of the world, say S_0, and the partially-ordered plan $\prod'$. In our approach, each fact in S_0 is represented by a probability value referring to the degree of belief of that fact being present in the world. In addition to this, the first action of Γ is dispatched for execution. PExec concerns the dynamic and robust execution of the plan Γ. These 2 sub-phases are further explained in the next sections.

The online phase of Olisipo controls the action deliberation depending on the observations made by the agent. As shown in Figure 1, the agent reads the state of the world S_0 and checks whether a totally-ordered plan Γ has already been determined or not. At the start, no Γ has been determined so the algorithm enters PGen. In PGen, the algorithm uses the *Total-Order Extractor* to build a set of totally-ordered plans. Then, the *Plan Selector* chooses the totally-ordered plan with the highest success probability and sets it as Γ, the

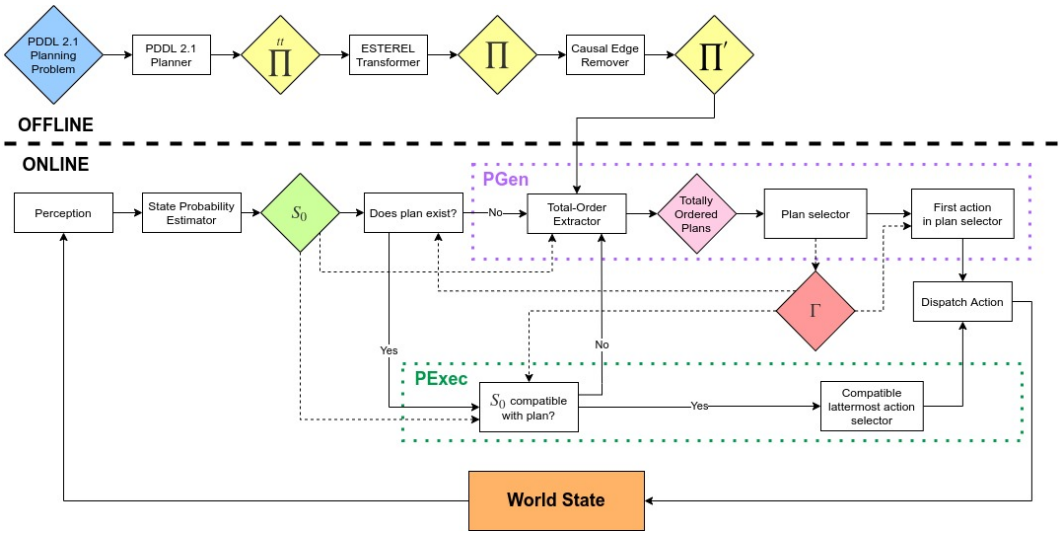

Figure 1 High-level overview of Olisipo's flow from planning to execution. The offline phase consists of converting a totally-ordered plan $\prod^{tt}$ into the partially-ordered plan $\prod'$. The online phase consists of two sub-phases, PGen and PExec. The PGen consists of finding the totally-ordered plan with the highest success probability, Γ, and dispatching its first action for execution. The PExec occurs when Γ has already been found and is compatible with the state of the world S_0. If Γ is compatible with S_0, then the compatible lattermost action of Γ is dispatched for execution.

totally-ordered plan to be used. Lastly, the first action of Γ is dispatched for execution. After executing this action, the system updates its world state representation and checks whether a totally-ordered plan has already been found. Since Γ has already been found, the algorithm advances to PExec. In this sub-phase, it first checks whether S_0 is compatible with Γ. If it is not compatible, then PGen is repeated and a new Γ is computed. If it is compatible, then the compatible lattermost action is dispatched for execution. The lattermost, as opposed to first, compatible action is chosen so that it becomes possible to skip actions that no longer contribute to achieving the goals. The agent will keep on reading the state of the world, and executing PGen and PExec, until the state of the world matches the goal.

2.1 Offline Phase: Generating an Adaptable Partially-Ordered Plan

The totally-ordered plan $\prod^{tt}$, generated by an external planner, is converted into a partially-ordered plan $\prod$ by creating a node for each instantaneous action and each durative action start and end. Then, temporal, durative and interference relations are generated between the nodes. Before starting plan execution, the causal support edges of $\prod$ are removed to give more flexibility in action selection, therefore resulting in the partially-ordered plan $\prod'$. A more detailed description of this process can be found on [21].

By first relaxing the problem to become deterministic, we are changing the problem's state-space and enabling the use of deterministic planners, which are able to scale for larger problem instances than undeterministic planners. This relaxation also means that we are moving the probability reasoning to the execution phase. Inspiration for this was taken from FF-Replan [30], which was the winner of the 2004 International Probabilistic Planning Competion (IPPC-04) and a top performer on IPPC-06. FF-Replan consisted of using a deterministic planner to solve a carefully constructed deterministic variant of the planning problem and replanning when an unexpected effect is observed. Similarly, we first solve a deterministic variant of the planning problem and obtain the plan $\prod^{tt}$, which is then converted to $\prod'$.

2.2 Online Phase

2.2.1 PGen: Building the Set of Totally-Ordered Plans and Calculating a Plan's Success Probability

The set of totally-ordered plans is built using a Branch and Bound search [19] through the nodes of $\prod'$, excluding the action start nodes which have already been dispatched but their respective action end nodes have not. This enables the repetition of an already executed action but prevents the algorithm from dispatching the action start of an action that has already started but has not finished yet.

Essentially, the search takes every node in $\prod'$ and orders them sequentially, hence building a totally-ordered plan. During search a Dynamic Bayesian Network (DBN) [10] is used to compute the joint probability of all actions being successful. When a solution is found, its probability is calculated as the joint probability of all actions being successful and of the facts of the goal being present in the solution. This solution probability is then used as reference value to prune the remaining search.

The search is pruned at a certain state S_i when one of the following conditions is verified:

- If the state S_i matches the goal;
- If the sequence of actions leading to S_i violates the temporal constraints;
- If the joint probability of the actions leading to S_i is lower than the probability of the best previously found solution (as Propositions 2 and 3 later explain, the joint probability of the actions is monotonically decreasing and is an upper bound to the solution probability);
- If there are no more applicable actions to S_i.

Our intention is to compute the success probability of a totally-ordered plan. We assume that it is possible to determine whether the execution of an action was successful. In addition to this, we say that a plan is successful if all the prescribed actions are applicable (i.e. the plan can be simulated) and if the goal condition is achieved in the final state of the plan.

A DBN, such as the one presented in Figure 2, is built to calculate a plan's success probability. This DBN contains two types of nodes: propositional nodes, which represent propositions at a certain time instant; and action nodes, which represent actions executed at a certain time instant. As it can be seen by the structure of the example DBN from Figure 2, the propositional nodes are divided into layers with one action node between each layer. Propositional nodes are propagated from layer to layer, representing their probability of spontaneously changing value with time. Action nodes have as parents and children their corresponding conditions and effects, respectively. We use the time and causal model of PDDL 2.1 [14], where durative actions are split into two instantaneous actions, corresponding to the start and end of a durative action. In the DBN of Figure 2, the action node *goto_waypoint_start(r0, w0, m2)* corresponds to the start of the durative action *goto_waypoint(r0, w0, m2)*, with its parents being the nodes corresponding to the *at start* conditions of the durative action, in the previous layer, and its children being the nodes corresponding to the *at start* effects, in the following layer. The node corresponding to the start of an action is a parent of the node corresponding to the end of an action. The nodes, in the layers between the action start and action end nodes, corresponding to the *over all* conditions of an action are parents of the action end node.

The uncertainty in the environment is represented in the Conditional Probability Distribution (CPD) of each node. Propositional nodes, from the first layer, do not have any parents so their CPD is represented by a single value, being 1 or 0 if they are *True* or *False* in the initial state, respectively. Propositional nodes with another propositional node as the only parent have a probability of spontaneously becoming *True* or *False*. Propositional

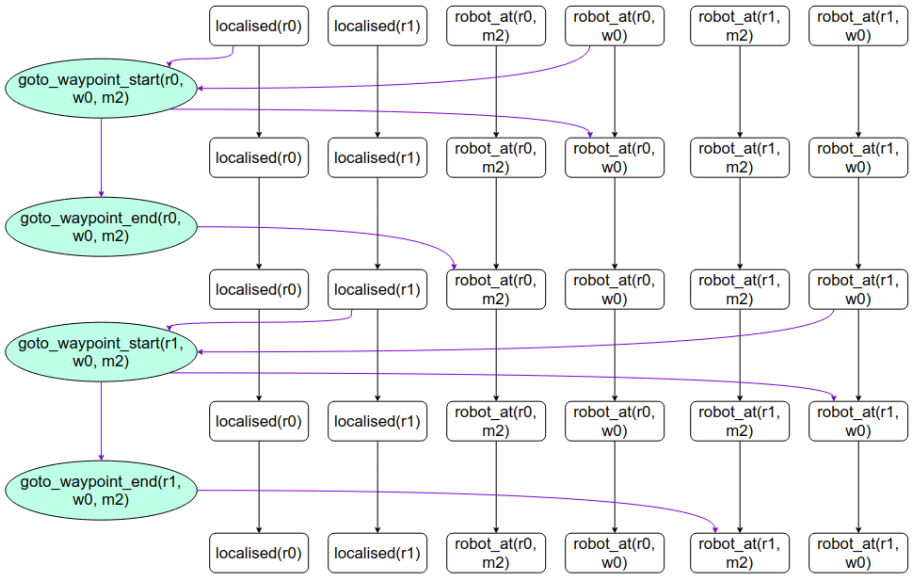

Figure 2 Example of the DBN's structure. Rectangular nodes symbolise proposition nodes and ellipsoidal nodes represent action nodes.

nodes with an action node and another propositional node as parents, have a probability of spontaneously becoming *True* or *False*, if the action node is *False*, and have a probability of being *True* corresponding to the effects of the action, if the action node is *True*. Action start nodes have a probability of being *True*, corresponding to the success probability of the action if all its preconditions are met, and have a probability of 0 of being *True* otherwise. Action end nodes have a probability of 1 of being *True* if all its preconditons are met and the action start node is *True*, and have a probability of 0 of being *True* otherwise.

The success probability of each action, of the effects of each action and of facts spontaneously changing in the world are an input from the user.

With the DBN fully built, we can compute the success probability of a plan. This success probability is defined as the conjunction of all actions in the plan being successful and of the goal facts being in the final state, $P(\mathcal{A} \cap \mathcal{G})$. In other words, it is defined as the joint probability of all action nodes, in the DBN, being *True* and of the facts from the goal being *True* in the last layer, marginalised over all other nodes. This calculation is represented in Eq. (1), with $\mathcal{A}$ being the set of action nodes and $\mathcal{G}$ being the set of nodes corresponding to the goal propositions in the last layer of the DBN. In addition to this, we also calculate the joint probability of all action nodes being *True*, $P(\mathcal{A})$, as displayed in Eq. (2), to prune the search.

$$P(\mathcal{A} \cap \mathcal{G}) = \sum_{\mathcal{S}_0,\ldots,\mathcal{S}_{N-1}} \sum_{\mathcal{S}_N \setminus \mathcal{G}} \left[\prod_{S \in \mathcal{S}_0} \rho(\pi(S)) \prod_{T \in \mathcal{A}} P(T|Pa(T)) \prod_{S \in \mathcal{S}_1 \cup \ldots \cup \mathcal{S}_N} P(S|Pa(S)) \right] \quad (1)$$

$$P(\mathcal{A}) = \sum_{\mathcal{S}_0,\ldots,\mathcal{S}_{N-1}} \sum_{\mathcal{S}_N} \left[\prod_{S \in \mathcal{S}_0} \rho(\pi(S)) \prod_{T \in \mathcal{A}} P(T|Pa(T)) \prod_{S \in \mathcal{S}_1 \cup \ldots \cup \mathcal{S}_N} P(S|Pa(S)) \right] \quad (2)$$

In Eqs. (1) and (2), we calculate $P(\mathcal{A} \cap \mathcal{G})$ and $P(\mathcal{A})$, respectively, by marginalising over all other nodes in the network. The term $\prod_{S \in \mathcal{S}_1 \cup \ldots \cup \mathcal{S}_N} P(S|Pa(S))$ represents the marginalisation of proposition nodes over its parents, The term $\prod_{T \in \mathcal{A}} P(T|Pa(T))$ represents marginalising an action node T over its parents $Pa(T)$ and, lastly, the term $\prod_{S \in \mathcal{S}_0} \rho(\pi(S))$ represents marginalising over the nodes of S_0 (initial state).

From Equations (2) and (1), it is possible to infer the following two propositions.

▶ **Proposition 2.** *$P(\mathcal{A})$ is monotonically decreasing as more actions are added to the DBN.*

Proof. As more actions are considered in $\mathcal{A}$, more factors are considered in $\prod_{T\in\mathcal{A}} P(T|Pa(T))$ and, since $P(T|Pa(T)) \leq 1$, the computed value will reduce or stay the same. ◀

▶ **Proposition 3.** *$P(\mathcal{A})$ is an upper bound for $P(\mathcal{A}\cap\mathcal{G})$, meaning $P(\mathcal{A}) \geq P(\mathcal{A}\cap\mathcal{G})$.*

Proof. This proposition comes from the fact that Equation (2) marginalises over a greater number of nodes than Equation (1). This can can be seen on the second sum of each Equation, since the sum $\sum_{S_N}$, from Equation (2), yields a greater or equal value than the sum $\sum_{S_N\setminus\mathcal{G}}$, from Equation (1). ◀

These two propositions enable the use of $P(\mathcal{A})$ to prune the Branch and Bound search, since it is an upper bound for $P(\mathcal{A}\cap\mathcal{G})$ and is monotonically decreasing as more actions are added to $\mathcal{A}$.

The DBN is then simplified by the iterative application of the 3 pruning rules defined below; these rules remove nodes and marginalisations which do not affect the probabilities calculation.

1. A node that is not the goal, is not an action and is not a parent of any other node can be discarded by marginalisation.
   Proof: We marginalise over all nodes which are neither the goal nor the actions. If a marginalised node has no children, then no other node depends on its value and it does not affect the probability calculation. The example below shows the effect of marginalising over node a, which has no children.

$$\sum_{a,b,c,\ldots} P(a|Pa(a)) \cdot \lambda(b,c,..) = \sum_{b,c} \lambda(b,c,..) \tag{3}$$

2. The marginalisation sum corresponding to a node that is precondition of an action can be removed.
   Proof: The marginalisation of an action's parent includes two terms, when it is *True* and when it is *False*. Only one of these terms satisfies the precondition of the action. The term which does not satisfy the action's precondition corresponds to the action having a null success probability, according to its CPD. Hence, we can discard the term which does not satisfy the action's preconditions and not marginalise over the parents of an action.
3. For all proposition nodes with an action node T as parent, the edges from any other parent can be removed.
   Proof: In our model, we have defined that the probability of a propositional node, with both another propositional node and an action node as parents, does not depend on the value of the propositional parent if the action parent is *True*, according to its CPD. Since we are only interested in calculating the case where all action nodes are *True*, then the edges from any other parents can be removed and the action node can be considered as the single parent of the propositional node.

Figure 3 illustrates which nodes can be removed, according to the 3 pruning rules, from the DBN of Figure 2.

Since we are building a DBN and calculating a probability at every step of the B&B search, the DBN is expected to be a bottleneck in terms of computing time and resources. Hence, it is vital to make this process efficient and iterative. Given this, instead of first building

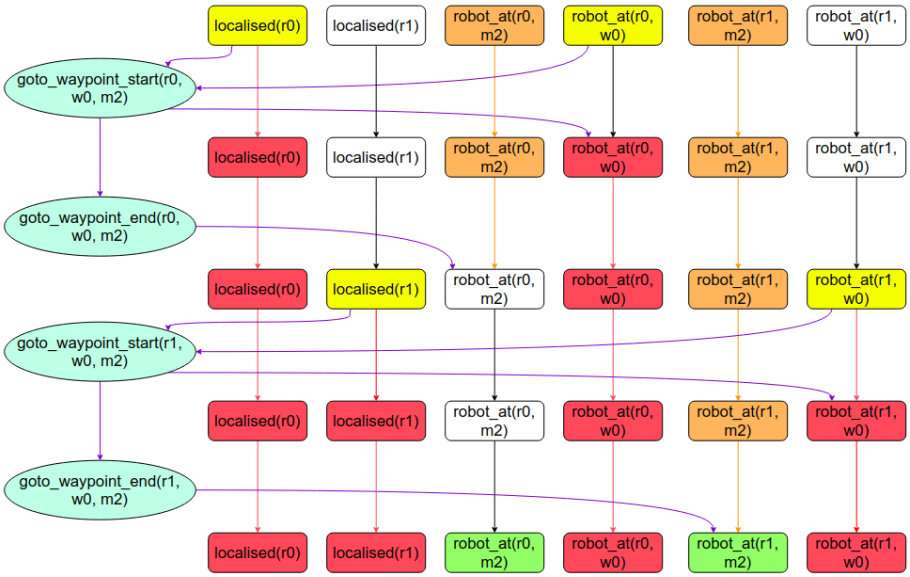

Figure 3 DBN representing which nodes can be removed from the DBN of Figure 2, by applying the 3 pruning rules. Rectangular nodes symbolise proposition nodes and ellipsoidal nodes represent action nodes. The colours of the nodes illustrate under which pruning rule they can be removed. Nodes and edges in red can be removed by Rule 1. Nodes in yellow have their marginalisation discarded, according to rule 2. Nodes and edges in orange can be removed after applying rule 3, followed by rule 1.

the entire DBN and then simplifying it, we build the already simplified DBN iteratively. In addition to this, we also use dynamic programming to calculate the values of $P(\mathcal{A})$ and $P(\mathcal{A} \cap \mathcal{G})$ iteratively.

The computation of the probabilities can be verified by resorting to any exact inference method, e.g., the well-known exact variable elimination.

2.2.2 PExec: Checking if a Plan is Compatible with the State of the World and Dispatching the Compatible Lattermost Action

In the PExec sub-phase, we wish to check if the state of the world is compatible with the plan Γ and to dispatch the lattermost action from Γ compatible with the state of the world. To achieve this, Algorithm 1 is used.

The pruning rules remove all irrelevant proposition nodes from the DBN, so the nodes remaining in the DBN, at a certain layer, must be verified in the state of the world to enable the execution of the action node that follows that layer.

Let us now consider the function *findCompatibleLattermostLayer* from Algorithm 1. This function traverses the DBN of plan Γ from the last layer to the first one, from S_N to S_0, searching for a layer which only contains propositions that match the state of the world, $\mathbb{S}$. After finding a layer i that matches this condition, it dispatches the action node after it for execution, a_i. If the action node a_i corresponds to the end of an action, then its corresponding action start must have already been dispatched. If all layers are verified and no layer satisfies these conditions, then the function returns -1, meaning that no layer is compatible with the state of the world.

As it can be seen in Figure 1, this verification will occur after the execution of each action. By considering the lattermost compatible action we are allowing for the skipping of actions. One important detail is that, due to the formalism of *PDDL*, every action that has been

Algorithm 1 Find the Lattermost Layer in the DBN Compatible with the State of the Sorld.

```
1: global 𝕊                                          ▷ Current state of the world.
2: global index_last_layer                           ▷ Index of the last DBN layer.
3: global actions_executing                          ▷ Set of executing actions.

4: function findCompatibleLattermostLayer
5:     if G ⊂ 𝕊 and actions_executing ≠ [] then
6:         return index_last_layer
7:     for i = index_last_layer-1 to 0 do
8:         if a_i is an action start node then
9:             if S_i ⊂ 𝕊 then
10:                return i
11:        else
12:            a_st = getCorrespondingActionStart(a_i)
13:            if S_i ⊂ 𝕊 and a_st ⊂ actions_executing then
14:                return i
15:    return −1                                     ▷ No layer matches 𝕊
```

started needs to be finished. So, even if the goal is already verified in the state of the world, the algorithm will first finish the actions that are still executing and only then declare that the goal has been reached. Another advantage that comes from using the pruned DBN is that we are only considering the propositions which are relevant for the execution of certain actions and for reaching the goal. If irrelevant propositions in the world change, then the same plan will remain valid. Plus, if the execution of a certain action fails, then the same plan might remain valid and the algorithm might retry the same action. Therefore, this algorithm can resist unexpected changes in the world and action failures without the need of replanning.

We allow for the repetition of actions because, in the interaction between a robot and the real world, it is very rare to have the exact same conditions. In fact, the failing of an action might be enough to slightly alter the environment. For example, a robot might fail to grasp a certain object, but in doing so change the object's pose such that it would succeed if it tried to grasp the object again. As another example, a robot might fail to navigate from room A to room B because someone stood on its way. However, if the robot tried to navigate again the person might no longer be there and the robot would navigate successfully. It is to account for situations like these that we allow for the repetition of actions.

3 Experimental Evaluation

Since the main feature of Olisipo consists in coping with unexpected changes in the world and action failures, it was necessary to develop a simulation environment capable of replicating these events. Hence, the environment developed for the original paper [21] was extended to account for simulated state perturbations.

The extended environment uses ROSPlan [5] to parse the domain and problems files. ROSPlan was chosen because it is compatible with ROS [25] and it is able to parse the PDDL language. The planner POPF [8] was used to build the totally-ordered plan $\prod^{tt}$. ROS Services were used to handle the communication between different scripts. The ROS version utilised was ROS Kinetic Kame[1].

[1] http://wiki.ros.org/kinetic

Scripts were developed to simulate perturbations in the environment. These perturbations occur after the execution of the start or end of an action and consist in randomly adding and removing propositions, as well as accounting for the failure of actions and of the effects of actions. The user defines the perturbations' probabilities in a configuration file which is then used to build and make calculations in the DBN.

Olisipo was compared to a simple dispatcher, which tries to execute a temporal plan and replans when the execution fails. Hence, the Esterel Dispatcher, from ROSPlan, was used and POPF was used to build the plan, since it was the same planner used in the offline phase of Olisipo. Ten replans was defined as the maximum amount of allowed replans both for the Esterel dispatcher and for the Olisipo algorithm.

3.1 Results

The Olisipo and Esterel Dispatchers were compared by performing 2000 trials on 10 different problems of *Simple Factory Robot Domain with 3 machines* (SF3) and 2000 trials on 8 different problems of *Advanced Factory Robot Domain with 3 machines* (AF3). The SF3 domain only has one action, where the AF3 domain has two actions. Each problem was manually generated and represents a different perturbed environment[2].

The chosen metrics to compare both dispatchers are:

- **M1**: Estimated probability of a successful execution;
- **M2**: Average number of replans;
- **M3**: Average number of actions executed.

Considering that each algorithm is either able or not able to solve a problem, its success distribution is a discrete binary distribution, namely a Bernoulli distribution. Hence, the Wilson Score Interval [29], presented in Equation (4), can be used to estimate the probability of each algorithm solving a problem, corresponding to M1.

In Equation (4), n is the total number of trials, n_s is the number of successful trials, n_f is the number of failed trials and z is the quantile function, with $z = 1.9599$ for a 95% confidence interval.

$$\hat{p} = \frac{n_s + \frac{1}{2}z^2}{n + z^2} \pm \frac{z}{n + z^2}\sqrt{\frac{n_s n_f}{n} + \frac{z^2}{4}} \tag{4}$$

Table 1 Tables showing the estimated probabilities of the Esterel dispatcher and of the Olisipo algorithm solving each problem from the SF3 and AF3 domains. These probabilities were calculated by making use of Equation (4). The last column displays the range of values where the estimated improvement, due to the use of Olisipo, lays.

SF3			
Problem	**Esterel**	**Olisipo**	**Improvement**
p1	0.13 ± 0.01	0.23 ± 0.02	$[0.07, 0.13]$
p2	0.10 ± 0.01	0.22 ± 0.02	$[0.09, 0.15]$
p3	0.07 ± 0.01	0.18 ± 0.02	$[0.08, 0.14]$
p4	0.043 ± 0.009	0.17 ± 0.02	$[0.10, 0.16]$
p5	0.024 ± 0.007	0.12 ± 0.01	$[0.08, 0.11]$
p6	0.06 ± 0.01	0.08 ± 0.01	$[0.00, 0.04]$
p7	0.020 ± 0.006	0.043 ± 0.009	$[0.008, 0, 038]$
p8	0.12 ± 0.01	0.16 ± 0.02	$[0.01, 0.07]$
p9	0.20 ± 0.02	0.23 ± 0.02	$[-0.01, 0.07]$
p10	0.025 ± 0.007	0.038 ± 0.008	$[-0.002, 0.028]$

AF3			
Problem	**Esterel**	**Olisipo**	**Improvement**
p1	0.17 ± 0.02	0.22 ± 0.02	$[0.01, 0.09]$
p2	0.15 ± 0.02	0.20 ± 0.02	$[0.01, 0.09]$
p3	0.16 ± 0.02	0.23 ± 0.02	$[0.03, 0.11]$
p4	0.17 ± 0.02	0.21 ± 0.02	$[0.00, 0.08]$
p5	0.09 ± 0.01	0.09 ± 0.01	$[-0.02, 0.02]$
p6	0.038 ± 0.008	0.047 ± 0.009	$[-0.008, 0.026]$
p7	0.029 ± 0.007	0.038 ± 0.008	$[-0.006, 0.024]$
p8	0.024 ± 0.007	0.031 ± 0.008	$[-0.008, 0.022]$

[2] Domains, problems, and configurations are available online [27].

The results referring to M1 can be seen in Table 1. From these tables, it is possible to verify that, overall, the Olisipo algorithm achieved a higher probability of success than the Esterel dispatcher. On the SF3 domain, the Olisipo algorithm presented an estimated improvement between -0.02 and 0.16. On the AF3 domain, the improvement was between -0.02 and 0.11. Despite the presence of some negative values, indicating a possible worst performance for the Olisipo algorithm, it is worth noting that on 12 of the 18 problems used, the improvement interval contained only positive values, indicating that Olisipo does present an improvement. Regarding the other problems, with the exception of problem 5 of AF3, at least 73% of the estimated interval is positive.

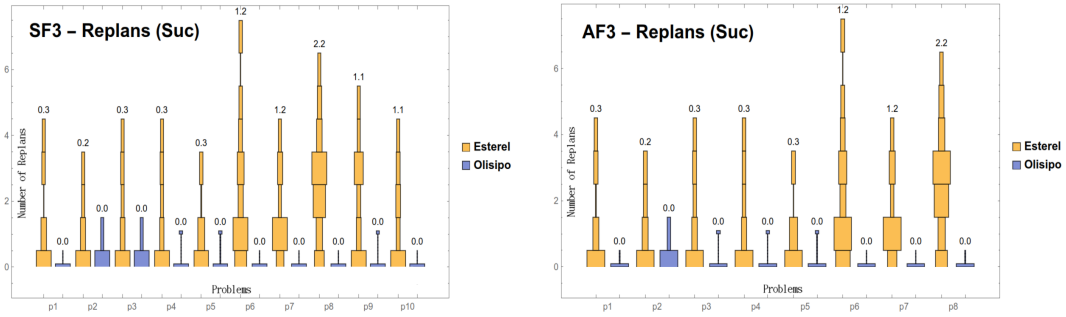

Figure 4 Distribution chart of the number of replans performed in the successful executions of the 2000 trials, for each problem of the SF3 and AF3 domains and for each dispatcher. The number on top of each distribution corresponds to its median.

Figure 4 presents the results referring to M2, namely the distribution of the number of replans, on successful executions, for each algorithm and for each problem. A successful execution occurs when a problem is successfully solved, with 10 or fewer replans performed. In these Figures, it is possible to verify that, for successful executions, the mean of the number of replans on the Olisipo algorithm is consistently lower than on the Esterel dispatcher. The mean of the number of replans in successful executions on the Olisipo algorithm was consistently zero, whereas the Esterel dispatcher often needed to replan, even achieving more than 5 replans on some problems.

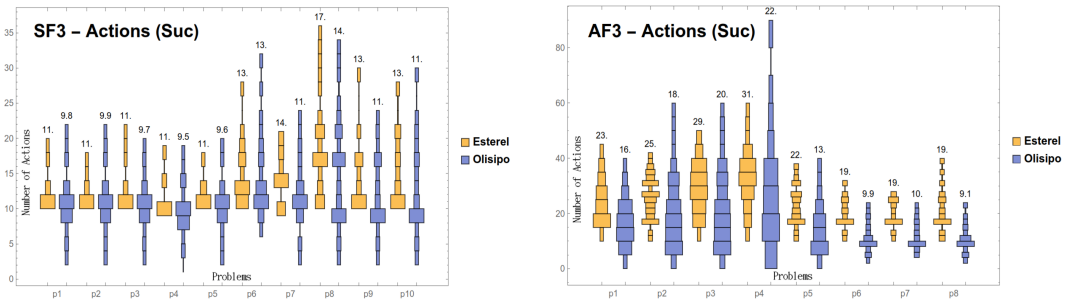

Figure 5 Distribution chart of the number of actions executed in the successful executions of the 2000 trials, for each problem of the SF3 and AF3 domains and for each dispatcher. The number on top of each distribution corresponds to its median.

Figure 5 presents the results corresponding to M3, more specifically to the distribution for the number of actions executed on successful executions. From these results, is possible to verify that the mean for the number of actions in successful executions on the Olisipo algorithm was consistently lower than on the Esterel dispatcher. For all problems of SF3,

except problem 6, the Olisipo algorithm offered a relative improvement of, at least, 10% in the mean of the number of actions for successful executions. The biggest relative improvements were achieved on problems 7 and 8, where the improvements were 21% and 18%, respectively. For all problems of AF3, the Olisipo algorithm offered an improvement of, at least, 28%. The biggest relative improvements were achieved on problems 6, 7 and 8, where the improvements were 48%, 47% and 52%, respectively.

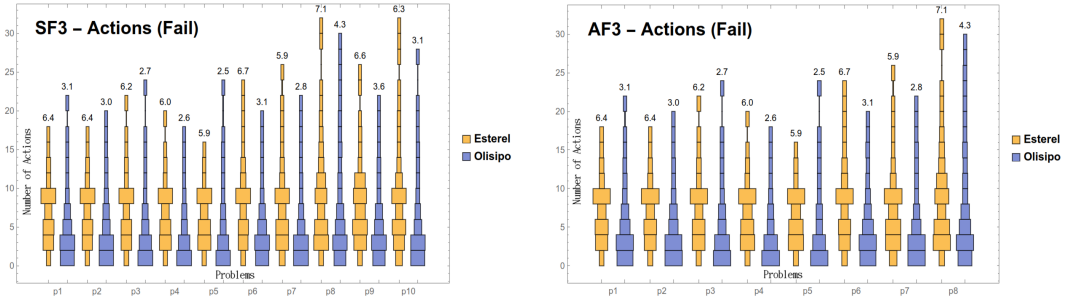

Figure 6 Distribution chart of the number of actions executed in the failed executions of the 2000 trials, for each problem of the SF3 and AF3 domains and for each dispatcher. The number on top of each distribution corresponds to its median.

Lastly, we will analyse the distribution of the number of actions on unsuccessful executions, from Figure 6, which also corresponds to M3. The mean and mode of executed actions before failing was lower on the Olisipo algorithm than on the Esterel dispatcher, for all problems of SF3 and AF3. The mean of executed actions before failure on SF3 was improved by, at least, 39% with the Olisipo algorithm. The biggest relative improvements occurred on problems 3, 4 and 5, where they were 56%, 57% and 58%, respectively. On the problems of AF3, the Olisipo algorithm also offered an improvement of at least 39% on the mean of executed actions before failure. The biggest improvements were achieved on problems 3, 4 and 5, with an improvement of 56%, 57% and 58% respectively.

Olisipo executes less actions than the Esterel dispatcher on failed executions because it requires the conditions of future actions to hold in the current state. This is achieved through the propagation of the parents of an action node to the previous layers in the DBN. Hence, Olisipo avoids the partial execution of plans which will not achieve the goal.

Regarding the execution time of the algorithm, it was found that the execution of the PGen sub-phase, which is the most resource consuming sub-phase, consistently had a duration under 2 seconds for all of the problems used.

From these results, it is possible to conclude that, in comparison to a simple dispatcher, the Olisipo algorithm has a higher probability of solving a given problem, needs to replan less times to successfully solve a problem and executes less actions on successful and failed executions. The consistency in the obtained results is especially useful to support this claim, since the results were obtained from 18 different problems from 2 different domains.

There were only two instances where the performance of Olisipo was similar to the Esterel dispatcher, which was the success probability of problem 5 of AF3 and the number of actions on successful executions of problem 6 of SF3. However, the performance of Olisipo in the other metrics, for each problem, surpassed the performance of the Esterel dispatcher. Hence, even in problems where the performance of Olisipo, in one metric, is similar to that of the Esterel dispatcher, the performance in the other metrics favour Olisipo.

Regarding the mean of executed actions in successful executions, the difference between Olisipo and Esterel was greater in AF3 than in SF3. Considering that SF3 only has one action on its domain, whereas AF3 has two actions, it is possible that this difference increases for domains with a greater number of actions.

Regarding the number of replans on successful executions, the mean for Olisipo was consistently zero for all problems. This means that Olisipo almost never needed to replan to achieve a successful execution. At the same time, it also means that Olisipo's replans rarely resulted in a successful execution. This implies that the replanning effort requested by the Esterel dispatcher was largely to recompute the same (sub)set of actions as was originally generated. In scenarios with limited computational time to spend planning, or larger problems for which full replanning is not feasible, the flexible approach of Olisipo is an obvious solution. More tests to evaluate this metric would be interesting, since this could mean that repeating the PGen sub-phase does not achieve any additional successful executions.

4 Conclusion

The robust execution of a temporal plan in a perturbed environment is a problem that remains to be solved. Perturbed environments, such as the real world, are non-deterministic and filled with uncertainty. Hence, the execution of a temporal plan presents several challenges and the employed solution often consists of replanning when the execution fails. In this paper, we propose a novel algorithm, named Olisipo, which aims to maximise the probability of a successful execution of a temporal plan in perturbed environments. To achieve this, a probabilistic model is used in the execution of the plan, instead of in the building of the plan. This approach enables Olisipo to dynamically adapt the plan to changes in the environment. In addition to this, the execution of the plan is also adapted to the probability of successfully executing each action. Olisipo was compared to a simple dispatcher and it was shown that it consistently had a higher probability of successfully solving a problem, performed fewer replans and executed fewer actions. Hence, Olisipo offers a substantial improvement in performance for disturbed environments.

Regarding the future work, there are several directions it could take. One possibility would be to save intermediate results, either from the DBN or the search tree, to cope with perturbations more efficiently. In addition to this, it would be interesting to study further the relevance of repeating the PGen sub-phase. Lastly, another possibility would be to test the Olisipo algorithm in a real world robot.

All source code and experimental setup is open source and available online [27]. Olisipo works as an add-on to any external planner and can easily be implemented by third-parties to improve execution performance.

References

1 Christer Bäckström. Computational aspects of reordering plans. *Journal of Artificial Intelligence Research*, 9:99–137, 1998.

2 Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

3 Michael Cashmore, Alessandro Cimatti, Daniele Magazzeni, , Andrea Micheli, and Parisa Zehtabi. Robustness envelopes for temporal plans. In *AAAI*, 2019.

4 Michael Cashmore, Andrew Coles, Bence Cserna, Erez Karpas, Daniele Magazzeni, and Wheeler Ruml. Replanning for situated robots. In J. Benton, Nir Lipovetzky, Eva Onaindia, David E. Smith, and Siddharth Srivastava, editors, *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019*, pages 665–673. AAAI Press, 2019. URL: `https://aaai.org/ojs/index.php/ICAPS/article/view/3534`.

5 Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carrera, Narcis Palomeras, Natalia Hurtos, and Marc Carreras. Rosplan: Planning in the robot operating system. In *ICAPS*, 2015.

6 A. Cesta, G. Cortellessa, S. Fratini, A. Oddi, and R. Rasconi. The APSI Framework: a Planning and Scheduling Software Development Environment. In *ICAPS (Application Showcase)*, 2009.

7 Alessandro Cimatti, Minh Do, Andrea Micheli, Marco Roveri, and David E. Smith. Strong temporal planning with uncontrollable durations. *Artif. Intell.*, 256:1–34, 2018. doi:10.1016/j.artint.2017.11.006.

8 Amanda Coles, Andrew Coles, Maria Fox, and Derek Long. Forward-chaining partial-order planning. *ICAPS 2010 - Proceedings of the 20th International Conference on Automated Planning and Scheduling*, pages 42–49, January 2010.

9 Patrick R. Conrad and Brian Charles Williams. Drake: An efficient executive for temporal plans with choice. *J. Artif. Intell. Res.*, 42:607–659, 2011. doi:10.1613/jair.3478.

10 Paul Dagum, Adam Galper, and Eric Horvitz. Dynamic network models for forecasting. In *Uncertainty in Artificial Intelligence*, pages 41–48. Morgan Kaufmann, 1992. doi:10.1016/B978-1-4832-8287-9.50010-4.

11 Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial intelligence*, 1991.

12 M. Do and S. Kambhampati. Improving temporal flexibility of position constrained metric temporal plans. In *ICAPS*, pages 42–51, 2003.

13 Maria Fox, Alfonso Gerevini, Derek Long, and Ivan Serina. Plan stability: Replanning versus plan repair. In Derek Long, Stephen F. Smith, Daniel Borrajo, and Lee McCluskey, editors, *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, ICAPS 2006, Cumbria, UK, June 6-10, 2006*, pages 212–221. AAAI, 2006.

14 Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of artificial intelligence research*, 2003.

15 J. Frank and A. Jónsson. Constraint-based Attribute and Interval Planning. *Constraints*, 8(4):339–364, 2003.

16 J. Frank and P. Morris. Bounding the resource availability of activities with linear resource impact. In *ICAPS*, pages 136–143, 2007.

17 M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Elsevier, 2004.

18 Félix Ingrand and Malik Ghallab. Deliberation for autonomous robots: A survey. *Artif. Intell.*, 247:10–44, 2017.

19 Ailsa H. Land and Alison G. Doig. An automatic method for solving discrete programming problems. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, pages 105–132. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. doi:10.1007/978-3-540-68279-0_5.

20 Steven Levine and Brian Williams. Watching and acting together: Concurrent plan recognition and adaptation for human-robot teams. *Journal of Artificial Intelligence Research*, 63, 2018.

21 Oscar Lima, Michael Cashmore, Daniele Magazzeni, Andrea Micheli, and Rodrigo Ventura. Robust plan execution with unexpected observations. *CoRR*, abs/2003.09401, 2020. arXiv:2003.09401.

22 Paul Morris. Dynamic controllability and dispatchability relationships. In Helmut Simonis, editor, *Integration of AI and OR Techniques in Constraint Programming*, pages 464–479, Cham, 2014. Springer International Publishing.

23 Paul H. Morris and Nicola Muscettola. Execution of temporal plans with uncertainty. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA.*, pages 491–496, 2000.

24 Nicola Muscettola, Paul H. Morris, and Ioannis Tsamardinos. Reformulating temporal plans for efficient execution. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento, Italy, June 2-5, 1998.*, pages 444–452, 1998.

25 Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. Ros: an open-source robot operating system. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, 2009.

26 Masood Feyzbakhsh Rankooh and Gholamreza Ghassem-Sani. Itsat: An efficient sat-based temporal planner. *Journal of Artificial Intelligence Research*, 53(1):541–632, 2015.

27 T. Ribeiro, O. Lima, M. Cashmore, A.Micheli, and R. Ventura. Experimental material for "A Probabilistic Approach to the Adaptable Execution of Deterministic Temporal Plans". URL: `https://github.com/TomasRibeiro96/Olisipo-planner`.

28 Ioannis Tsamardinos and Martha E. Pollack. Efficient solution techniques for disjunctive temporal reasoning problems. *Artif. Intell.*, 151(1-2):43–89, 2003. `doi:10.1016/S0004-3702(03)00113-9`.

29 Edwin B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927. `doi:10.1080/01621459.1927.10502953`.

30 Sung Yoon, Alan Fern, and Robert Givan. Ff-replan: A baseline for probabilistic planning. *ICAPS 2007*, pages 352–, 2007.