Efficient Anytime Computation and Execution of Decoupled Robustness Envelopes for Temporal Plans

Michael Cashmore
Strathclyde University, UK

Alessandro Cimatti
Fondazione Bruno Kessler, Italy

Daniele Magazzeni
Kings College London, UK

Andrea Micheli
Fondazione Bruno Kessler, Italy

Parisa Zehtabi
Kings College London, UK

Abstract

One of the major limitations for the employment of model-based planning and scheduling in practical applications is the need of costly re-planning when an incongruence between the observed reality and the formal model is encountered during execution. Robustness Envelopes characterize the set of possible contingencies that a plan is able to address without re-planning, but their exact computation is expensive; furthermore, general robustness envelopes are not amenable for efficient execution.

In this paper, we present a novel, anytime algorithm to approximate Robustness Envelopes, making them scalable and executable. This is proven by an experimental analysis showing the efficiency of the algorithm, and by a concrete case study where the execution of robustness envelopes significantly reduces the number of re-plannings.

2012 ACM Subject Classification Computing methodologies → Robotic planning

Keywords and phrases Temporal Planning, Robustness Envelopes

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

1 Introduction

When planning and scheduling techniques are employed in practical applications, one of the major problems is the need for on-line re-planning when the observed contingencies are not aligned with the ones that were considered at planning time. These situations are common, because it is arguably impossible to predict the entire range of situations an autonomous system can encounter, especially when the planning domain encompasses time and temporal constraints. Unfortunately, re-planning can be costly in terms of time, and computational resources can be scarce on-board, so limiting the use of re-planning is very important for practical purposes. In principle, it is also possible to continue with the execution of a plan even when the observed contingencies are unexpected, optimistically hoping for a successful completion. However, this approach offers no formal guarantee, and is prone to the risk of continuing execution of a plan that is bound to fail.

Several approaches have been proposed in the literature to address this problem (see [13] for a survey focused on robotics). Some authors propose to post-process plans and generalize them relying on the scheduling constraints that are relevant for execution [17, 15, 10]. Another line of research focuses on the creation of "least commitment plans", i.e. plans that are left

partially open by the planner so that the execution can be adapted to some variation in the contingencies [11, 20, 9, 4, 19]. Others tackled the idea of transforming temporal plans with no adaptability into flexible plans [7]. Finally, one can explicitly model the uncertainties in the planning problem and construct a plan that offers formal guarantees with respect to such a model. Examples include Conformant and Contingent Planning [12], Probabilistic Planning [14] and Strong Temporal Planning with Uncertain Durations [5] that considers temporal uncertainty in the durations of actions.

Recently, *Robustness Envelopes* (REs) have been proposed to overcome several limitations of the approaches mentioned above. REs formally capture the possible contingencies that a given temporal plan, obtained by planning in a *deterministic domain*, can deal with, without having to re-plan [2]. REs are regions defined over a set of numeric parameters that represent possible contingencies, and contain all the parameter valuations ensuring plan validity. In general, REs may be non-convex, and can express dependencies between the parameters. However, the technique proposed in [2] has two main drawbacks limiting its practical applicability. First, the exact computation of REs is extremely expensive: the proposed approach is doubly exponential in the size of the planning problem. Second, REs in their general form are not suited for efficient execution: the dependencies among parameters might require run-time reasoning.

In this paper, we overcome these limitations, achieving scalability and executability. We focus on *Decoupled Robustness Envelopes* (DREs), i.e. hyper-rectangular REs where the dependencies among parameters are not present, and are thus much easier to execute. Our first contribution is a novel and scalable algorithm for computing DREs as sound approximations of REs. A sound approximation ensures that every point within the DRE belongs within the RE. The algorithm is anytime, and proceeds by incrementally under-approximating the RE with increasingly large DREs. The algorithm can be stopped at any time, providing a meaningful result already amenable to start execution. In its general formulation, the RE for a given plan is naturally modeled as a quantified first order formula in the theory of Linear Real Arithmetic. Our algorithm does not need to precisely compute the quantifier-free description of the RE (which requires an expensive step of quantifier elimination, and is ultimately responsible for the inefficiency demonstrated in [2]). Rather, it starts from a degenerate DRE consisting of a single point, and progressively tries to enlarge it along different dimensions, checking if each extension is contained in the RE, until a given precision is reached. The algorithm relies on *quantifier-free* queries to a Satisfiability Modulo Theory [1] solver.

Our second contribution is to demonstrate the practical use of DREs in a robotic executor, extending the classical flow from planning to execution to re-planning, as follows. First, a plan is generated from a deterministic model using temporal planning technologies, and transformed into a Simple Temporal Network (STN) formulation [6]; at this point, we parametrize the durations of some of the actions in the plan and/or the consumption rates in the domain specification. DREs are then computed for the introduced parameters and passed to the executor. In turn, the dispatching of the actions in the STN plan begins and continues until one observed duration or consumption rate happens to be outside of the DRE. At this point, the executor detects that the plan is no longer guaranteed to succeed, and re-planning is triggered.

The proposed approach was implemented in the ROSPlan [3] framework, and experimentally evaluated along two directions. The algorithm for DRE generation was compared against the base line in [2], demonstrating orders-of-magnitude improvements compared to the exact computation of REs, and the ability to deal with a much larger number of parameters. The

overall execution loop has been evaluated on a family of concrete case studies in a logistic domain, showing that the use of DREs, compared to the optimistic executor in ROSPlan, significantly reduces the number of re-plannings and improves the execution success-rate.

2 Background

We consider planning problems expressed in the PDDL 2.1 [8] temporal planning language; for the sake of brevity we do not report the full syntax and semantics of such planning problems, but we directly introduce the parametrized planning problem idea adapted from [2].

▶ **Definition 1.** *A **parametrized planning problem** $\mathcal{P}_\Gamma$ is a tuple $\langle \Gamma, \mathcal{P} \rangle$, where Γ is a finite set of real-valued parameters $\{\gamma_1, \cdots, \gamma_n\}$ and $\mathcal{P}$ is a PDDL 2.1 planning problem in which action conditions, action effects, goals and initial states can all contain parameters.*

Intuitively, symbols (from a known set Γ) can be used in expressions where real-typed constants are usually allowed.

As customary in many cases of plan execution, we use plans expressed as Simple Temporal Networks (STN) [6]. An STN plan is a set of constraints of the form $t_i - t_j \leq k$ where t_i and t_j are time points linked to action happenings (i.e. either the start or the end of an action instance in the plan) and $k \in \mathbb{Q}$. In addition, we allow parameters in the plan specification by generalizing the notion of STN plans.

▶ **Definition 2.** *A **parametrized STN plan** π_Γ for a parametrized planning problem $\mathcal{P}_\Gamma \doteq \langle \Gamma, \mathcal{P} \rangle$ is an STN Plan where some constraints are in the form $t_i - t_j \leq \gamma_i$ where t_i and t_j are time-points of the STN plan and $\gamma_i \in \Gamma$.*

We define the Robustness Envelope (RE) for a parametrized problem and plan as the set of possible values for the parameters that make the plan valid when the symbols are substituted with values in the plan and problem specifications. In order to compute the RE, [2] defines a set of logical formulae that characterize the RE and use quantifier elimination techniques (e.g. [18]) to explicitly construct the region. The encoding is divided in three expressions: indicated as $enc^{\pi_\Gamma}_{tn}$, $enc^{\pi_\Gamma}_{eff}$ and $enc^{\pi_\Gamma}_{proofs}$. The formula $enc^{\pi_\Gamma}_{tn}$ encodes the temporal constraints imposed by π_Γ limiting the possible orderings of time points. The formula $enc^{\pi_\Gamma}_{eff}$ encodes the effects of each time point on the state variables, while $enc^{\pi_\Gamma}_{proofs}$ encodes the validity properties of the plan, namely that the conditions of each executed action are satisfied, that the goal is reached, and that the ϵ-separation constraint [8] imposed by PDDL 2.1 is respected. Then, let $\bar{X}$ be the set of all the variables appearing in the formulae above except the parameter values, the RE is characterized by all the models of the following formula.

$$\exists \bar{X}.(enc^{\pi_\Gamma}_{tn} \wedge enc^{\pi_\Gamma}_{eff}) \wedge \forall \bar{X}.((enc^{\pi_\Gamma}_{tn} \wedge enc^{\pi_\Gamma}_{eff}) \rightarrow enc^{\pi_\Gamma}_{proofs})$$

As observed in [2], any under-approximation of the RE gives sound information on the contingencies in which the plan is guaranteed to be valid; in particular, a convenient restriction for the representation and handling of REs is to associate a closed interval of possible values to each parameter, defining an hyper-rectangle. If such hyper-rectangle is contained in the RE, we have a "Decoupled Robustness Envelope" (DRE) that retains the guarantees of the RE but avoids the complexity of inter-dependencies among parameters.

▶ **Definition 3.** *A **Decoupled Robustness Envelope** for a parametrized planning problem $\mathcal{P}_\Gamma$ and STN plan π_Γ is a bound assignment $\rho : \Gamma \rightarrow \mathbb{Q}_{>=0} \times \mathbb{Q}_{>=0}$, such that any parameter assignment $v : \Gamma \rightarrow \mathbb{Q}_{>=0}$, with $l \leq v(\gamma) \leq u$ and $\langle l, u \rangle \doteq \rho(\gamma)$, is contained in the RE for $\mathcal{P}_\Gamma$ and π_Γ.*

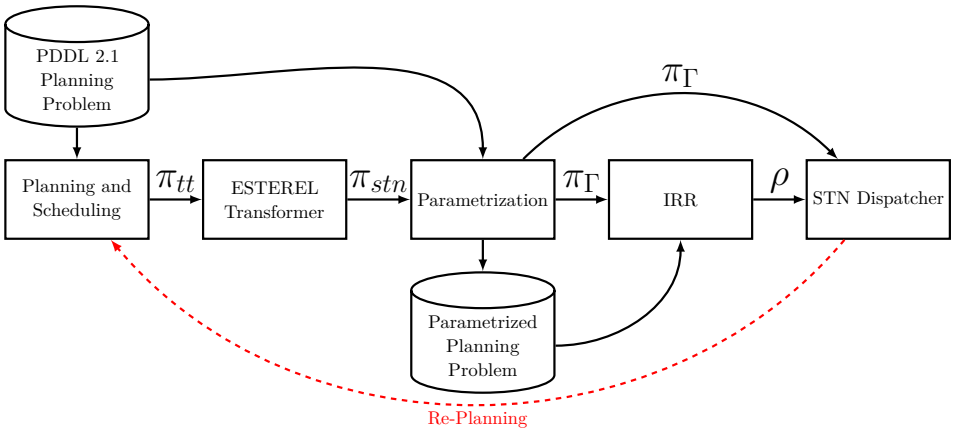


Figure 1 Overview of the proposed flow.

Note that many DREs are possible for a given problem and plan: it suffices that all the assignments allowed by the DRE are points in the RE. In this paper, we elaborate on this idea and propose an algorithm that incrementally builds DREs that are contained within the unconstrained RE without paying the cost of explicitly computing the RE itself.

Finally, we highlight that given any two DREs ρ_1 and ρ_2 three cases are possible: either ρ_1 is subsumed by ρ_2 (i.e. for each parameter γ, $\rho_1(\gamma) \subseteq \rho_2(\gamma)$), or ρ_1 subsumes ρ_2, or the two DREs are incomparable. Hence, there is no absolute best DRE in general: we aim for a DRE that is not subsumed by any other, but there can be multiple DREs with this property.

3 Execution Flow

The general idea we pursue in this paper is to exploit the information and the generalization provided by the synthesis of REs to limit the number of re-plannings and increase the success-rate in execution. In particular, we propose the flow from planning to execution depicted in Figure 1. Starting from a planning problem formulation expressed in PDDL 2.1, we use any off-the-shelf temporal planner[1] to compute a timed sequence of actions that reaches the goal from the initial state. We call this plan "time-triggered" (indicated with π_{tt}) in the picture. This plan is not natively amenable for execution because it defines one specific trace that does not allow any adaptability: it is extremely unlikely for a real system to be perfectly controlled to satisfy a specific trace. Hence, π_{tt} needs to be converted in a flexible, executable STN (π_{stn}) by the ESTEREL transformer of ROSPlan. The usual flow would pass this STN directly to the dispatcher for translating the plan actions into commands for the robotic platform at the proper time. Instead, here we pre-process this plan using REs in the hope of generalizing its applicability and reducing the number of re-plannings. In particular, the STN plan is passed to a parametrization component that re-reads the planning problem formulation and enriches it with parameters, generating a Parametric Planning Problem and a parametric STN plan (π_Γ). Those are the inputs for the

[1] Several existing PDDL planners are unable to generate flexible STNs either because of an implementation limitation or because the technique does not allow it (e.g. SAT-based planners). Our approach is able to generate DREs from these planners as well, and work in concert with existing algorithms for the execution of STNs.

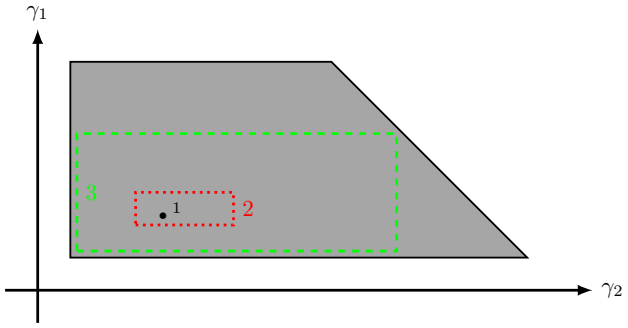


Figure 2 A graphical representation of IRR: starting from the parameter values from the original plan (depicted as the black point), IRR tries to construct increasingly better under-approximations (the colored rectangles) of the RE (the gray area), without actually computing it. Upon termination, each edge of the resulting DRE is guaranteed to be at most β apart from the border of the actual region.

computation of the RE. In our flow, for performance reasons and to avoid complex run-time reasoning, instead of computing the exact, unconstrained RE, we use a novel algorithm, called Incremental Rectangular-Robustification (IRR for short), that computes a DRE. The algorithm is anytime, so that it is possible to retrieve unfinished computations and exploit them in execution: in fact, any under-approximation of the final result retains the needed properties of the RE. At this point, we pass the DRE (ρ) together with the parametrized STN plan to the STN dispatcher. We modified the dispatching algorithm to exploit the information in the DRE to limit the re-plannings to situations where they are needed. In particular, the dispatcher translates the actions, while checking that the observed values for the parameters (being either action durations, resources or rates) fall within the bounds imposed by ρ. If this is not the case, re-planning is needed and the whole flow is re-executed.

Parametrization. The first non-standard step highlighted in Figure 1 is the parametrization. In fact, there are multiple ways in which parameters can be added to a deterministic temporal planning problem to characterize useful quantities for execution. In general, one can parametrize any numeric quantity in the planning problem whose value might differ from the environment in which the plan will be executed. In order to be useful for the STN dispatcher, however, such quantities must be eventually observable (directly or indirectly). Otherwise, it is impossible for the executor to check whether the RE is still satisfied or if a re-planning is needed. In this paper experimentation, we focused on two such quantities, namely the durations of actions and resource consumption rates. The former is a classical source of uncertainty when temporal planning is employed in a robotics scenario, the latter is another source of uncertainty that can perturbate the execution of a plan, for example when the resource harvesting is not fully controllable (e.g. a solar panel yield depends on the weather) or when the consumption is not fully predictable (e.g. the battery consumption is very hard to precisely estimate as it depends on temperature, exact capacity and so on).

4 Incremental Rectangular-Robustification

We now present our novel algorithm for incrementally computing decoupled robustness envelopes. We call this algorithm "Incremental Rectangular-Robustification" (IRR).

The idea behind the algorithm is to construct incrementally better hyper-rectangular under-approximations of the RE for a given problem and plan. In fact, this constitutes a

Algorithm 1 Incremental Rectangular-Robustification

1: $enc_{valid} \leftarrow$ QuantifierElimination$(\exists \bar{X}.enc_{tn}^{\pi_\Gamma} \wedge enc_{\mathit{eff}}^{\pi_\Gamma})$
2: **function** IRR$(\beta : \mathbb{Q}_{>0})$
3: $R \leftarrow \{\gamma \rightarrow [\pi(\gamma), \pi(\gamma)] \mid \gamma \in \Gamma\}$
4: $\Delta \leftarrow \{\gamma \rightarrow max(\pi(\gamma) \times \omega_\gamma, \beta) \mid \gamma \in \Gamma\}$
5: $\Theta \leftarrow \{\gamma \rightarrow \{\texttt{UB}, \texttt{LB}\} \mid \gamma \in \Gamma\}$
6: **while** $\exists \gamma \in \Gamma.\Delta(\gamma) \geq \beta$ **do**
7: $\tilde{\gamma} \leftarrow$ Pick$(\{\gamma \mid \gamma \in \Gamma \wedge \Theta(\gamma) \neq \emptyset \wedge \Delta(\gamma) \geq \beta\})$
8: $\theta \leftarrow$ Pick$(\Theta(\tilde{\gamma}))$
9: $[l, u] \leftarrow R(\tilde{\gamma})$
10: **if** $\theta = \texttt{UB}$ **then** $u \leftarrow (u + \Delta(\tilde{\gamma}))$ **else** $l \leftarrow (l - \Delta(\tilde{\gamma}))$
11: $R' \leftarrow \{\gamma \rightarrow R(\gamma) \mid \gamma \in \Gamma \wedge \gamma \neq \tilde{\gamma}\} \cup \{\tilde{\gamma} \rightarrow [l, u]\}$
12: **if** CheckInEnvelope(R') **then** $R \leftarrow R'$
13: **else**
14: $\Theta(\tilde{\gamma}) \leftarrow \Theta(\tilde{\gamma}) \setminus \theta$
15: **if** $\Theta(\tilde{\gamma}) = \emptyset$ **then**
16: $\Delta(\tilde{\gamma}) \leftarrow \Delta(\tilde{\gamma})/2; \quad \Theta(\tilde{\gamma}) \leftarrow \{\texttt{LB}, \texttt{UB}\}$
17: **return** R
18: **function** CheckInEnvelope(R)
19: $enc_R \leftarrow \bigwedge_{\gamma \in \Gamma, [l,u]=R(\gamma)} l \leq \bar{\gamma} \wedge \bar{\gamma} \leq u$
20: **if** IsSAT$(enc_R \wedge \neg enc_{valid})$ **then return** false
21: **else**
22: **return** IsValid$((enc_{tn}^{\pi_\Gamma} \wedge enc_{\mathit{eff}}^{\pi_\Gamma} \wedge enc_R) \rightarrow enc_{proofs}^{\pi_\Gamma})$

direct way of computing a DRE by generate-and-test. The starting point is the de-generated hyper-rectangle composed of the single point given by the parameter values of the original plan. The algorithm tries to extend the hyper-rectangle along one dimension (i.e. it tries to widen the interval of possibilities associated to one of the parameters) and checks if the resulting hyper-rectangle is in fact an under-approximation of the RE. If it is, the new hyper-rectangle is kept as it is guaranteed to be a valid DRE. Otherwise, another dimension or another increment is chosen for the algorithm to proceed. The general intuition behind the algorithm is depicted in Figure 2.

Algorithm 1 reports the pseudo-code of IRR. The formula enc_{valid} is computed once and off-line. It corresponds to the basic requirements for the hyper-rectangle to be a valid DRE: only parameter values that are not contradicting the STN plan and the causal flow of effects are admissible. This is the same as the first piece of the logical formulation in [2], but luckily it is the easier part of the quantification and can be efficiently computed. Then, the IRR function is in charge of computing a hyper-rectangle R maintaining the following invariant: at each step, R is a subset of the RE of the problem. The hyper-rectangle R is represented as a pair of bounds (lower- and upper-) assigned to each parameter (this directly models a DRE as per Definition 3), and is initialized (line 3) with the values of the non-parametric plan π. The algorithm uses two functions to control how the hyper-rectangle is transformed from one cycle to the next. Δ associates to each parameter a number that is the value used to increase the upper-bound or to decrease the lower-bound for that parameter. The initial value for Δ is the original value of the parameter scaled by a weight for such parameter, but any positive number bigger than β is enough to guarantee soundness and termination of the algorithm. Note that these weights can be used to express preferences on the parameters: a higher weight pushes the algorithm to expand a specific parameter more than others. The function Θ is used to decide in which direction the interval of a parameter can be extended.

Two directions are possible: `UB` indicates that we want to extend the upper-bound and `LB` indicates that we want to decrease the lower-bound (line 10). Initially both directions are possible, but when we discover (line 13) that one direction is infeasible with the current Δ, we remove this direction from the possibilities. This process will be continued with the current Δ values until expansion in all directions is infeasible. At this stage the value of Δ is halved, eventually reaching a value lower than β. Each time that the values of Δ are updated, we reset Θ to allow expansion in both directions once again.

The main loop of the algorithm continues until all the values of Δ are lower than β: this is to guarantee that the minimum distance from each border of the hyper-rectangle and the border of the actual RE is at most β. The algorithm picks a parameter $\tilde{\gamma}$ to be analyzed among the parameters having at least one direction available in Θ and that have not converged already (line 7); then, it generates a candidate hyper-rectangle R' by extending either the lower- or the upper- bound of $\tilde{\gamma}$. At this point, we check if R' is contained in the RE or not (line 12). If it is, we keep it and continue the loop, otherwise, we discard this hyper-rectangle and we record that with the current Δ we cannot extend $\tilde{\gamma}$ in this direction by removing the direction θ from $\Theta(\tilde{\gamma})$. Moreover, if no direction is left for $\tilde{\gamma}$, we halve its value of Δ and reset Θ so that $\tilde{\gamma}$ can be tentatively extended again using a smaller step (lines 15-16).

The core part of the algorithm consists in checking a candidate hyper-rectangle for containment in the actual RE, without explicitly computing the region itself. This is done via the CheckInEnvelope function that performs two SMT checks corresponding to the two quantifiers appearing in the RE logical formulation of [2]. The first check looks for points belonging to R that are not parts of the validity region enc_{valid}, the second checks if the rectangle (together with the guarantees from the plan and the effects) implies the proof requirements characterizing the REs. The important point here, is that both checks are quantifier-free, i.e. no quantifier elimination is involved.

▶ **Theorem 4.** *The CheckInEnvelope(R) function returns true if and only if R is a valid DRE*

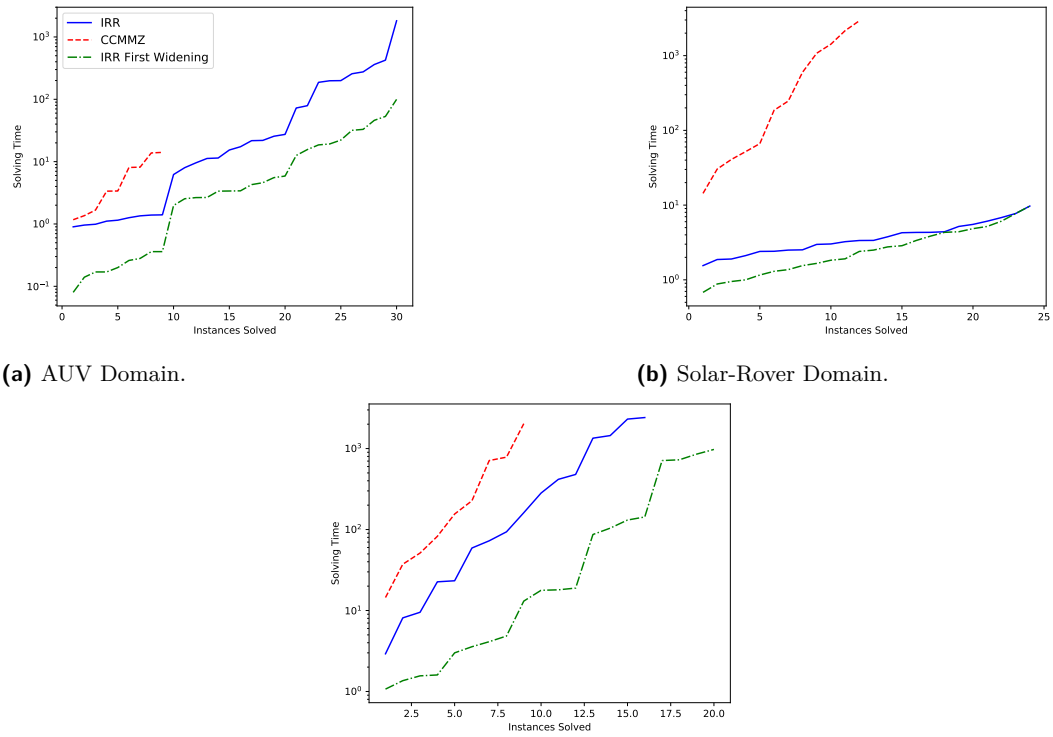
Proof. The algorithm logically checks the following formula: $\neg(\exists\bar{\Gamma}.enc_R \wedge \neg enc_{valid}) \wedge \forall\bar{\Gamma},\bar{X}.(enc_{tn}^{\pi_\Gamma} \wedge enc_{\mathit{eff}}^{\pi_\Gamma} \wedge enc_R) \rightarrow enc_{proofs}^{\pi_\Gamma}$, that can be rewritten as $\forall\bar{\Gamma}.enc_R \rightarrow (enc_{valid} \wedge (\forall\bar{X}.(enc_{tn}^{\pi_\Gamma} \wedge enc_{\mathit{eff}}^{\pi_\Gamma}) \rightarrow enc_{proofs}))$ that states that enc_R is a subset of the encoding of the RE. Then, for Definition 3, R is the encoding of a valid DRE. ◀

An interesting feature of the algorithm is that it is "anytime", i.e. at each time, we can take the hyper-rectangle R and we have the guarantee that R is contained in the RE and is thus a valid DRE. Moreover, the algorithm is guaranteed to terminate if the RE is finite in all dimensions.

▶ **Theorem 5.** *If the robustness envelope is bounded in all dimensions, IRR always terminates.*

Proof. All the values in Δ are initially positive and whenever the candidate rectangle is found to exit the RE (line 13) one of the values in Δ is halved. Eventually all the parameters will be considered and they will be eventually found to exit the RE because it is bounded in all dimensions. Therefore, all the values of Δ will become smaller than β. ◀

We highlight that IRR is in fact an optimization procedure that incrementally maximizes the size of a starting DRE, terminating when a maximal DRE is found within the given precision limit β.



(a) AUV Domain.

(b) Solar-Rover Domain.

(c) Generator Linear Domain.

Figure 3 Scalability experiments on [Cashmore *et al.*, 2019] domains: the number of solved instances (sorted by difficulty for each solver) is considered on the X axis and is compared with the logarithmic time needed to solve each instance (lower, longer lines are better).

5 Experiments

We now present our empirical analysis that comprises three sets of experiments. The first aims at showing the superior performance of IRR as compared with the logical approach of [2]. The second shows a practical use-case of the execution flow proposed in this paper when the duration of actions is uncertain. In the third experimentation we use our DRE technique to execute plans when the consumption rates of resources is uncertain.

IRR. We start by considering the experimental dataset and the tool (hereafter called CCMMZ) provided in [2]. The benchmarks use a varying number of parameters; in particular, AUV ranges from 1 to 8 parameters, Generator Linear from 1 to 4 and Solar Rover between 1 and 4. We compare our IRR implementation with CCMMZ on all the available instances and domains, measuring the total run-time and using the "decoupled envelope generation" functionality of the tool. Moreover, in order to take into account the anytime nature of IRR, we also measure the time at which the rectangle R in IRR widens and becomes different than a single point (i.e. we measure the first time the Algorithm 1 reaches line 17) and we call this timing "IRR First Widening". In all our experiments we set $\beta = 1$ and all $\omega_i = 1$ to find the decoupled region approximated to a single unit with no preferences among the parameters (obviously, we set the same parameter preference also in CCMMZ). We executed all of the instances on a Xeon E5-2620 2.10GHz machine setting a time limit of 3600s and a RAM memory limit of 20GB.

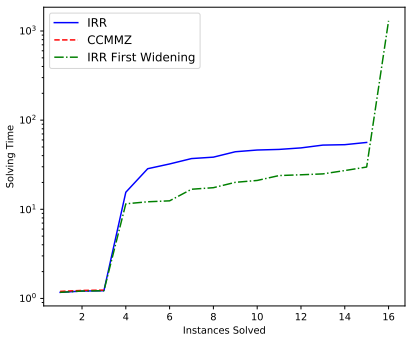


Figure 4 Scalability experiments on the delivery domain: the number of solved instances (sorted by difficulty for each solver) is considered on the X axis and is compared with the logarithmic time needed to solve each instance (lower, longer lines are better).

Figure 3 shows the result of this analysis. IRR is able to solve many more instances than CCMMZ and is consistently quicker. Moreover, we note how the first widening is often encountered quite early in the execution, marking the margin for anytime exploitation of IRR. In fact, after the first widening, IRR already computed a meaningful and non-trivial under-approximation of the RE that can be used for execution. This is particularly evident in the Generator Linear domain where the algorithm is unable to fully terminate in some cases, but the first widening point is reached.

In addition to these domains, we also experiment with several instances of a service-robot domain that we also use for the following execution experimental analysis. The domain, called "Robot Delivery" is a simplified version[2] of the domain used in the Planning and Execution Competition for Logistics Robots in Simulation [16]. The domain comprises a fleet of small robots that can navigate in an euclidean graph. These robots are tasked to pick and deliver orders within a deadline. Collecting orders requires two robots present at a machine. We scaled the number of parameters in the instances between 1 and 33. Figure 4 shows the scalability of IRR and CCMMZ on this domain. These instances are much harder for both the solvers compared to the previous domains; in fact, CCMMZ is only able to solve 3 instances, while IRR is able to solve 15 of them. Also in this case, the anytime nature of IRR is evident by observing the difference from the first widening and the algorithm completion.

Finally, we investigate how quickly the IRR algorithm converges in our experiments. We define convergence at step i in a run of IRR that terminates with hyper-rectangle R_{end} as follows (R_i indicates the hyper-rectangle at step i).

$$Convergence(i) = \frac{\sum_{[l,u]\in R_i} u - l}{\sum_{[l,u]\in R_{end}} u - l} \times 100$$

Intuitively, this gives the percentage of the region covered by R_i with respect to R_{end}. (Note that R_{end} contains R_i because the IRR algorithm only expands previous hyper-rectangles.) Figure 5 shows, for all problems solved by IRR in our benchmark set, the percentage of convergence achieved after any number of steps of the IRR algorithm. Clearly from the plot,

[2] A simplified RCLL domain was used because the PDDL provided in the RCLL image is not complete and the RCLL simulation requires external processes, e.g. a referee box. We are interested in the flexible execution success rate, so we created PDDL instances encoding logistics problems without any external processes.

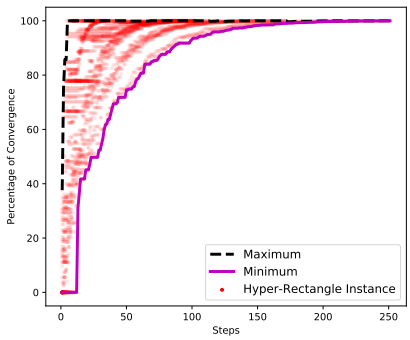


Figure 5 Convergence of IRR in terms of steps: each red dot is a DRR computed by IRR and the plot shows the progression in terms of convergence at each step of the algorithm. The purple line indicates the poorest convergence percentage for each step in any experiment; similarly the black, dashed line shows the best convergence.

Executor	1 Parameter		2 Parameters		3 Parameters		4 Parameters		5 Parameters	
	Coverage	Avg Replans	Coverage	Avg Replans	Coverage	Avg Replans	Coverage	Avg Replans	Coverage	Avg Replans
DREEx	**92.2%**	**0.1**	**85.8%**	**0.2**	**83.2%**	**0.2**	**72.8%**	**0.1**	**63.0%**	**0.1**
BLEx(0)	0.0%	NA	0.0%	NA	0.0%	NA	0.0%	NA	0.0%	NA
BLEx(10)	24.5%	1.0	4.8%	1.0	0.8%	1.2	0.2%	0.8	0.0%	NA
BLEx(20)	44.4%	0.8	19.9%	1.0	6.3%	1.0	2.4%	1.6	0.8%	1.9
BLEx(30)	58.9%	0.7	34.0%	0.9	18.8%	1.2	9.2%	1.2	4.5%	1.4
BLEx(40)	68.8%	0.6	52.2%	0.8	34.7%	1.0	23.8%	1.1	11.8%	1.5
BLEx(50)	75.0%	0.5	62.2%	0.7	49.0%	0.9	37.8%	1.1	27.9%	1.2
BLEx(60)	78.8%	0.4	69.0%	0.5	59.4%	0.7	53.4%	0.9	44.5%	1.1

Table 1 Coverage and average number of re-plans in the duration-uncertain delivery domain.

in a limited number of steps we often approximate very well the final intervals; in particular, within 50 steps we already cover more than 70% of the final sum of the interval sizes in all the cases.

Duration-Uncertain Flexible Execution. We use the Robot Delivery domain to investigate the merits of an on-line plan executor equipped with our IRR algorithm. In particular, we begin by focusing the analysis on the number of re-plans and on the plan execution success rate when only the duration of actions is uncertain during execution. In this domain, a robot has to collect a spindle from a shelf, construct a base by performing six steps (possibly in parallel), then mount a number of rings, and finally deliver the order. Orders have deadlines that must be met for delivery. The domain allows the agent to drop an order and restart from scratch with a new one at any time, but this disposal action takes some time (10 seconds in our case) and the robot needs to navigate on a symbolic euclidean graph to pick the parts, assemble and deliver the order. Each instance is simulated in an environment where actions have a non-deterministic duration described by a normal distribution with a minimum value. Due to the difficulty in manipulation tasks, the actions executed for preparing the base (in which the robots interact with machines) have the highest degree of variance. These actions have mean durations of 120, 130, 140, 150, 160 and 170 seconds, and a standard deviation of 70. Due to this uncertainty and the presence of deadlines for the order delivery, the execution of a plan can fail even when a re-planning schema is employed. We generated a total of 100 problems by varying the deadlines for the orders.

Algorithm 2 STN Dispatch

```
1:  function STNDispatch(π_stn, ρ)
2:    finished = false
3:    while ¬finished do
4:      for each node n ∈ π_stn do
5:        min, max ← MinMaxDispatchTime(n, π_stn, ρ)
6:        if n is action start then
7:          if (min ≤ n ≤ max) ∧ ¬started(n) then
8:            StartExecuting(n)
9:          else if (n ≥ max) ∧ ¬started(n) then
10:           finished = true
11:       else if n is action end then
12:         if (n ≥ max) ∧ ¬Completed(n) then
13:           finished = true
14:         else if (n ≤ min)∧ Completed(n) then
15:           finished = true
16:   return GoalsAchieved( )
```

Our DRE-based approach was implemented in ROSPlan, as described in Section 3. The STN dispatcher starts the execution of actions following the temporal constraints of the STN: the process is illustrated in Algorithm 2. For each node, the minimum and maximum dispatch times are calculated during execution (line 5). The dispatch ends when an action completes outside of the temporal constraints allowed by the STN, or has not been started after the maximum allowed dispatch time. When the dispatch ends, it returns *true* if the goals have been achieved; otherwise, re-planning is triggered as shown in Figure 1. The system will continuously attempt to re-plan until the deadlines make the PDDL planning problem unsolvable.

We compare the executor described in Section 3 (indicated as DREEx) against several baselines in which we dispatch the STN plan π_{stn} without parameterization. In such baselines, the executor dispatches the STN plan allowing for a fixed deviation in the duration of actions and ends dispatch only when the action duration falls outside of this interval. This is the optimistic technique for execution implemented in ROSPlan that, differently from DREEx, offers no formal guarantees. We consider baseline executors named BlEx(0) to BlEx(60) allowing for 0% to 60% variability in action duration before triggering a re-plan. For example, given an action with a predicted duration 100 seconds, BlEx(0) will re-plan if the duration is not exactly 100; BlEx(20) will re-plan if the duration is outside of the interval $[80, 120]$. The baseline BlEx(0) corresponds to formally executing the time-triggered plan π_{tt}: re-planning happens if any action duration differs from what was expected in π_{tt}. We highlight that, when DREEx is employed and the observation is within the envelope computed by IRR, we have the formal guarantee of plan success; as soon as one observation is outside of the envelope, we choose to re-plan.

The overarching idea in these experiments is that the planner usually optimistically selects the easier, quicker goal and the agent starts to execute the plan. If the execution of the preparation actions goes overlong, it might become impossible to deliver the order, so the only way to successfully recover is to immediately dispose the current order and switch to another one with a less imminent deadline. If the executor fails in realizing this situation, it continues to execute the plan until it tries to deliver the order, at which point it realizes that the deadline is not met. Since a lot of time has been wasted in the preparation, it might be impossible to recover from this situation. Ideally, we expect that the predictive power of

Executor	Coverage	Avg Replans
DREEx	**99.2%**	**0.1**
BLEx(0)	1.0%	2.0
BLEx(10)	25.6%	**0.1**
BLEx(20)	50.7%	**0.1**
BLEx(30)	66.4%	**0.1**
BLEx(40)	77.9%	**0.1**
BLEx(50)	82.1%	**0.1**
BLEx(60)	86.8%	**0.1**

Table 2 Coverage and average number of re-plans in the resource-uncertain delivery domain.

DREs allows the identification of situations where the deadline cannot be met and a swift re-planning to change the objective order is needed.

Table 1 reports the results of the experiment. We report the coverage percentage (i.e. the percentage of problems successfully executed over the benchmark set) as well as the average number of re-plannings for successful runs. The baseline BLEx(0), not accounting for any variance in action duration, was unable to solve any problem successfully. Allowing for more flexibility in the duration of actions increases the coverage as should be expected. However, the DREEx approach achieves greater coverage than all baselines in all the cases. This is because in this problem, the ability to realize early that the agent is late for the first order and change course of actions to achieve the second order is pivotal for achieving a good success rate.

Resource-Uncertain Flexible Execution. Finally, we show that our flow can be used when parameters are not just action durations. We expanded the delivery domain to consider the battery consumption of the robots. In particular, each action in the revised domain checks that enough battery is present upon start and consumes a fixed amount of battery. We parametrized the consumption rate of actions, so that the DRE will compute the possible consumption values for which a given plan is valid. The executor is then demanded to observe the contingent consumption and possibly invoke a re-planning if the observation does not fall in the DRE prescription. Also in this case, the baselines BLEx(X) invoke the replanning when the battery consumption is observed to be $X\%$ higher or lower than the nominal value.

Table 2 reports the results of the experiment, and shows how the use of DREEx is beneficial for the success-rate achieving an almost perfect success-rate with very few replannings on average.

6 Conclusion

In this paper, we make the case for the use of Robustness Envelopes (RE) in a plan execution framework. We present a novel, anytime algorithm to compute Decoupled Robustness Envelopes (DRE) that is empirically superior to the previously known logic-based construction. Moreover, we demonstrate the usefulness of the produced artifacts by integrating them in the ROSPlan framework and showing on a concrete example the positive impact on the number of re-plannings and the plan success-rate.

In the future, we will consider other kinds of approximations for the robustness envelope (e.g. hyper-octagons instead of hyper-rectangles). We will also explore the link to temporal uncontrollability and non-deterministic planning. Finally, using Incremental Rectangular-Robustification (IRR) in parallel with the dispatcher could allow variation in parameters being considered during execution.

References

1 C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability*, pages 825–885. IOS Press, 2009.

2 M. Cashmore, A. Cimatti, D. Magazzeni, A. Micheli, and P.a Zehtabi. Robustness envelopes for temporal plans. In *AAAI*, 2019.

3 M. Cashmore, M. Fox, D.Long, D. Magazzeni, B. Ridder, A. Carrera, N. Palomeras, N. Hurtós, and M. Carreras. Rosplan: Planning in the robot operating system. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015.*, pages 333–341, 2015.

4 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.

5 A. Cimatti, M. Do, A. Micheli, M. Roveri, and D. Smith. Strong temporal planning with uncontrollable durations. *Artif. Intell.*, 256:1–34, 2018. doi:10.1016/j.artint.2017.11.006.

6 R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, 1991.

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

8 M. Fox and D. Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

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

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

11 M. Ghallab and H. Laruelle. Representation and control in IxTeT, a temporal planner. In *AIPS*, pages 61–67, 1994.

12 M. Ghallab, D. Nau, and P. Traverso. *Automated planning - theory and practice*. Elsevier, 2004.

13 Félix Ingrand and Malik Ghallab. Deliberation for autonomous robots: A survey. *Artificial Intelligence*, 247:10–44, 2017.

14 Mausam and A. Kolobov. Planning with markov decision processes: An ai perspective. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–210, 2012. doi:10.2200/S00426ED1V01Y201206AIM017.

15 N. Muscettola. Computing the envelope for stepwise-constant resource allocations. In *CP*, pages 139–154, 2002.

16 T. Niemueller, G. Lakemeyer, and A. Ferrein. The robocup logistics league as a benchmark for planning in robotics. *Planning and Robotics (PlanRob-15)*, page 63, 2015.

17 N. Policella, S. Smith, A. Cesta, and A. Oddi. Generating robust schedules through temporal flexibility. In *ICAPS*, pages 209–218, 2004.

18 A. Schrijver. *Theory of Linear and Integer Programming*. J. Wiley & Sons, 1998.

19 A. Umbrico, A. Cesta, M. Cialdea Mayer, and A. Orlandini. Integrating resource management and timeline-based planning. In *ICAPS*, pages 264–272, 2018.

20 B. Williams and V. Gupta. Unifying model-based and reactive programming within a model-based executive. In *Workshop on Principles of Diagnosis*, 1999.

A Planning Semantics

In this section we reproduce the additional syntax and semantics of planning problems presented in [2], which provide additional background on definitions 1 and 2. For a more complete description linking these concepts we refer the reader to the source. We start by defining our planning language: we adopt the full PDDL 2.1 [8] with continuous change.

▶ **Definition 6.** *A **planning problem** $\mathcal{P}$ is a tuple $\langle P, V, A, I, G\rangle$, where P is a set of propositions; V is a set of real variables, called fluents; A is a set of durative and instantaneous actions; $I : P \cup V \to \{\top, \bot\} \cup \mathbb{R}$ is the total function describing the initial state of the predicates and the fluents. $G : P \cup V \to \{\top, \bot\} \cup \mathbb{R}$ is a (possibly partial) function indicating the goal condition. A durative action a is a tuple $\langle pre_a, eff_a, dur_a\rangle$, where pre_a is a set of conditions for the actions partitioned in three subsets $pre_{\vdash a}$, $pre_{\leftrightarrow a}$ and $pre_{\dashv a}$ of at-start, over-all and at-end conditions; eff_a is the set of action effects, partitioned in seven sets: $eff^{+}_{\vdash a}$ (positive starting effects), $eff^{-}_{\vdash a}$ (negative starting effects), $eff^{num}_{\vdash a}$ (numeric starting effects), $eff^{+}_{\dashv a}$ (positive ending effects), $eff^{-}_{\dashv a}$ (negative ending effects), $eff^{num}_{\dashv a}$ (numeric ending effects) and $eff^{num}_{\leftrightarrow a}$ (continuous numeric effects); and dur_a is a set of duration constraints. An instantaneous action a is a tuple $\langle pre_a, eff_a\rangle$, where pre_a is a set of pre-conditions and eff_a is the set of action effects, partitioned in eff^{+}_{a} (positive effects), eff^{-}_{a} (negative effects) and eff^{num}_{a} (numeric effects).*

In the usual PDDL 2.1 setting, a plan is defined as a set of actions associated with a starting time and a duration. We define this kind of plans as time-triggered plans.

▶ **Definition 7.** *A **time-triggered plan** π for a planning problem $\mathcal{P} \doteq \langle P, V, A, I, G\rangle$ is a set of tuples $\langle t, a, d\rangle$, with $t \in \mathbb{R}_{\geq 0}$, $a \in A$ and $d \in \mathbb{R}_{>0}$ iff a is a durative action.*

For the sake of brevity, we omit the formal definition of validity for such a plan, which can be found in [8]. Here, it suffices to remind oneself that a plan is valid if by simulating the system controlled by the plan, all the prescribed actions are applicable (all their conditions are satisfied at the time the action is executed) and the goal is reached after the last action terminates.

We define an STN plan as a constraint network of time points indicating the starting or the ending of actions. Note that the STN plan contains all the information of, and is strictly more general than a time-triggered plan. Moreover, that it is not necessary to first find a time-triggered plan in order to generate an STN plan.

▶ **Definition 8.** *An **STN plan** π for $\mathcal{P} \doteq \langle P, V, A, I, G\rangle$ is a tuple $\langle T, C\rangle$, where T is the set of time points $\{z\} \cup \{t^s_{da}, t^e_{da} \mid da \text{ is a durative action instance}\} \cup \{t_a \mid a \text{ is an instantaneous action instance}\}$ and C is a set of constraints in the form $t_i - t_j \leq b$ with $t_i, t_j \in T$, and $b \in \mathbb{R}$.*

Finally, we can define the validity of an STN plan by considering the set of all possible time-triggered plans that are compatible with the STN specification. If all such plans are valid, we say that the STN plan is valid.

▶ **Definition 9.** *Given an STN plan $\pi \doteq \langle T, C\rangle$ and an assignment $\mu : T \to \mathbb{R}$ s.t. $\mu(z) = 0$, the **induced time-triggered plan** by μ is the time-triggered plan $tt(\mu) \doteq \{\langle \mu(t^s_{da}), da, \mu(t^e_{da}) - \mu(t^s_{da})\rangle \mid da \text{ is a durative action}\} \cup \{\langle \mu(t_a), a, 0\rangle \mid a \text{ is an instantaneous action}\}$.*

▶ **Definition 10.** *An STN plan $\pi \doteq \langle T, C\rangle$ for $\mathcal{P}$ is **valid** if for each assignment $\mu : T \to \mathbb{R}$ s.t. $\mu(z) = 0$ and for all $t_i - t_j \leq b \in C$ $\mu(t_i) - \mu(t_j) \leq b$, the time-triggered plan $tt(\mu)$ is valid for $\mathcal{P}$.*