# Monte Carlo Tree Search for Priced Timed Automata

Jensen, Peter Gjøl; Kiviriga, Andrej; Guldstrand Larsen, Kim; Nyman, Ulrik; Mijačika, Adriana; Høiriis Mortensen, Jeppe

*Published in:*
Quantitative Evaluation of Systems

*DOI (link to publication from Publisher):*
10.1007/978-3-031-16336-4_19

*Publication date:*
2022

*Document Version*
Accepted author manuscript, peer reviewed version

Link to publication from Aalborg University

*Citation for published version (APA):*
Jensen, P. G., Kiviriga, A., Guldstrand Larsen, K., Nyman, U., Mijačika, A., & Høiriis Mortensen, J. (2022). Monte Carlo Tree Search for Priced Timed Automata. In E. Ábrahám, & M. Paolieri (Eds.), *Quantitative Evaluation of Systems: 19th International Conference, QEST 2022, Proceedings* (pp. 381-398). Springer. https://doi.org/10.1007/978-3-031-16336-4_19

# Monte Carlo Tree Search for Priced Timed Automata

Peter Gjøl Jensen, Andrej Kiviriga(✉), Kim Guldstrand Larsen, Ulrik Nyman,
Adriana Mijačika, and Jeppe Høiriis Mortensen

Aalborg University, Selma Lagerløfs Vej 300, 9220 Aalborg, Denmark
{pgj, kiviriga, kgl, ulrik}@cs.aau.dk,
adrianamijacika@gmail.com, Jeppe.h.m7997@gmail.com

**Abstract.** Priced timed automata (PTA) were introduced in the early 2000s to allow for generic modelling of resource-consumption problems for systems with real-time constraints. Optimal schedules for allocation of resources may here be recast as optimal reachability problems. In the setting of PTA this problem has been shown decidable and efficient symbolic reachability algorithms have been developed. Moreover, PTA has been successfully applied in a variety of applications. Still, we believe that using techniques from the planning community may provide further improvements. Thus, in this paper we consider exploiting Monte Carlo Tree Search (MCTS), adapting it to problems formulated as PTA reachability problems. We evaluate our approach on a large benchmark set of PTAs modelling either Task graph or Job-shop scheduling problems. We discuss and implement different complete and incomplete exploration policies and study their performance on the benchmark. In addition, we experiment with both well-established and our novel MTCS-based optimizations of PTA and study their impact. We compare our method to the existing symbolic optimal reachability engines for PTAs and demonstrate that our method (1) finds near-optimal plans, and (2) can construct plans for problems infeasible to solve with existing symbolic planners for PTA.

**Keywords:** Priced Timed Automata (PTA) · Model-checking · Monte Carlo Tree Search (MCTS) · Planning · Upper confidence bounds for trees (UCT)

QEST Artifact Evaluation 2022 Accepted

## 1 Introduction

The world is full of planning and scheduling problems that have impact on the real world. Finding optimal solutions for such problems can be of great importance for profit maximization or resource minimization, affecting financial success and sustainable development. In general such problems do not just have one solution, but many solutions – with varying cost. These scheduling problems are one sub-field within operations research, and lots of effort has been put into finding both optimal and near optimal solutions to them.

One technique that has been successfully applied to planning is that of model checking, e.g. BDD based model checking [19]. For optimal planning problems involving timing constraints, the notion of priced timed automata was introduced in the early 2000s, with initial decidability results [7, 4] based on so-called *corner-point regions* and later with efficient symbolic forward reachability algorithms using so-called *priced* *zones* made available in the tool UPPAAL CORA. Here a generic and highly expressive modeling formalism is provided, extending the classical notion of timed automata [3] with a cost-variable (to be optimized), but also providing support for discrete variables over structured (user-defined) types, as well as user-specified procedures [8]. In fact, the notion of PTA allows for an extension of Planning Domain Definition Language (PDDL) 2.1 at level 3 towards duration-dependent and continuous effects to be encoded as demonstrated by [18]. Most recently so-called *extrapolation* techniques have been introduced for more efficient analysis of PTA, implemented in the tool TiaMo [13].

Applications of PTA and UPPAAL CORA are several and from a variety of areas [14], e.g. power optimization of dataflow applications [2], battery scheduling [27], planning of nano-satelites [23, 30], grape harvest logistic [33], programmable logic controllers [35], smart grids [21], service oriented systems [17], and optimal multicore mapping of spreadsheets [11] to mention a few.

Despite the success of PTA and UPPAAL CORA, we still believe that the performance may be improved by exploiting advances made by the planning community. Thus, we consider in this paper various ways of exploiting Monte Carlo Tree Search (MCTS) to further improve performance of PTA optimization. MCTS is a powerful technique that has seen application in many domains requiring (near-) optimal planning, including problem instances where the size of the search-space makes symbolic and complete methods infeasible. In particular, MCTS [16] has already been applied directly to Job-shop [5] scheduling problems. We benchmark our implementations of MCTS based analysis of PTA on Job-shop and Task graph problems and compare against the two tools UPPAAL CORA [9] and TiaMo [13].

The rest of the paper is organized as follows: First we formally define Priced Timed Automata, then we introduce a general formalization of Monte Carlo Tree Search along with specific PTA policies. Finally we discuss additional enhancements and present our experimental evaluation.

## 2 Priced Timed Automata

The priced timed automaton [6] is an extension of timed automaton [3] with prices on both locations and transitions. Delaying in locations entails a price growth based on fixed price (cost) rate, while taking transitions is associated with a fixed price. We now present the formal definition of priced time automaton and its semantics based on [9].

Let $\mathbb{C}$ be a set of clocks. The set of constraints over clocks $\mathbb{C}$, $B(\mathbb{C})$, are defined as the set of conjunctions of atomic constraints of the form $x \bowtie n$, where $x \in \mathbb{C}$, $\bowtie \in \{<, \leq, =, >, \geq\}$ and $n \in \mathbb{N}_{\geq 0}$. Such constraints – guards and invariants – allow to restrict the behavior w.r.t. the values of clocks. The power set of $\mathbb{C}$ is denoted as $2^{(\mathbb{C})}$.

**Definition 1 (Priced Timed Automaton).** *A Priced Timed Automaton (PTA) over clocks $\mathbb{C}$ and actions $Act$ is represented as a tuple $A = (L, l_0, E, I, P)$ where:*

- *$L$ is a finite set of locations,*
- *$l_0 \in L$ is the initial location,*
- *$E \subseteq L \times \mathcal{B}(\mathbb{C}) \times Act \times 2^{(\mathbb{C})} \times L$ is a set of edges where an edge connects two locations and contains a guard, an action, and a set of clocks to be reset,*

- $I\colon L \to, \mathcal{B}(\mathbb{C})$ *is a set of location invariants, and*
- $P\colon (L \cup E) \to \mathbb{N}$ *assigns cost rates and cost increments to locations and edges, respectively.*

In the case of $(l, g, a, r, l') \in E$, we write $l \xrightarrow{g,a,r} l'$. A clock valuation $v$ over $\mathbb{C}$ is a mapping $v\colon \mathbb{C} \to \mathbb{R}_{\geq 0}$ and $\mathbb{R}^{\mathbb{C}}$ denotes a set of all clock valuations. The semantics of a PTA is defined in terms of a priced transition system:

**Definition 2 (Priced Transition System).** *A Priced Transition System (PTS) over actions $Act$ is a tuple $T = (S, s_0, \Sigma, \to)$ where:*

- *$S$ is a set of states*
- *$s_0$ is an inital state,*
- *$\Sigma = Act \cup \mathbb{R}_{\geq 0}$ is the set of labels, and*
- *$\to \subseteq (S \times \Sigma \times \mathbb{R}_{\geq 0} \times S)$ is a set of labelled and priced transitions. We write $s \xrightarrow{a}_p s'$ whenever $(s, a, p, s') \in \to$.*

Now a PTA $A = (L, l_0, E, I, P)$ defines a PTS $T_A = (S, s_0, \Sigma, \to)$, where the set of states $S$ are pairs $(l, v)$, with $l \in L$ is a location and $v$ is a clock valuation s.t. the invariant $I(l)$ of $l$ is satisfied by $v$, denoted $v \models I(l)$.

There are two possible types of transitions between states: *action transitions* and *delay transitions*. Action transitions are the result of following an enabled edge in the PTA $A$. As a result, the destination location is activated and the clocks in the reset set are set to zero, and the price of the transition is given by the cost of the edge. Formally:

$$(l, v) \xrightarrow{a}_p (l', v') \text{ iff } \exists (l, g, a, r, l') \in E, \text{such that}$$
$$v \models g \wedge v' = v[r] \wedge v' \models I(l) \wedge p = P((l, g, a, r, l'))$$

where $v[r]$ is the valuation given by $v[r](x) = 0$ if $x \in r$ and $v[r](x) = v(x)$ otherwise.

Delay transitions allow the time to pass resulting in an increase of the value of all clocks, but with no change of the location. The cost of a delay transition is the product of the duration of the delay and the cost rate of the active location. Formally:

$$(l, v) \xrightarrow{d}_p (l, v') \text{ iff } v' = v + d \wedge v \models I(l) \wedge v' \models I(l) \wedge p = d \cdot P(l)$$

where $v + d$ is the valuation given by $(v + d)(x) = v(x) + d$ for all $x$. Finally, the initial state is $s_0 = (l_0, v_0)$, where $l_0$ is the initial location, and $v_0(x) = 0$ for all clocks $x$. For networks of priced timed automata we use vectors of locations and the cost rate of a vector is the sum of the cost rates of individual locations.

An example of a PTA is shown in Figure 1 with clocks $x$ and $y$ and five locations – $\ell_0$ (initial), $\ell_1, \ell_2, \ell_3$, and $\ell_g$ (goal), with cost rates $P(\ell_0) = +5$, $P(\ell_2) = +10$ and $P(\ell_3) = +1$, and the cost of the edge from $\ell_2$ ($\ell_3$) to $\ell_g$ is +1 (+7). Note that the invariant $y = 0$ in $\ell_1$ enforces that the location must be left immediately. Below we show two example traces for the automaton:

$$\pi_1 = (\ell_0, x = 0, y = 0) \to_0 (\ell_1, x = 0, y = 0) \to_0 (\ell_3, x = 0, y = 0)$$
$$\xrightarrow{2}_2 (\ell_3, x = 2, y = 2) \to_7 (\ell_g, x = 2, y = 2)$$
$$\pi_2 = (\ell_0, x = 0, y = 0) \xrightarrow{1.5}_{7.5} (\ell_0, x = 1.5, y = 1.5) \to_0 (\ell_1, x = 1.5, y = 0)$$
$$\to_0 (\ell_2, x = 1.5, y = 0) \xrightarrow{0.5}_5 (\ell_2, x = 2, y = 0.5) \to_1 (\ell_g, x = 2, y = 0.5)$$

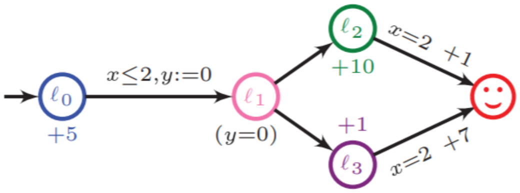

Fig. 1: Priced Timed Automata example

We see that $\pi_1$ reaches $\ell_g$ with a total cost of $2 + 7 = 9$, whereas the reachability cost of $\pi_2$ is $7.5 + 5 + 1 = 13.5$. In fact, among the infinitely many traces that reach $\ell_g$, $\pi_1$ has the minimum cost. The question of cost-optimal reachability was shown decidable by [7] and later proven to be PSPACE-complete [12]. Here, extending the result for reachability of TAs in [15], it is observed that a PTS semantics with natural-valued delays is complete for PTAs with non-strict guards. Moreover, if $k$ is the maximum constant to which clocks are compared to in guards and invariants, it suffices to consider delays no greater than $k + 1$. In short, in Definition 2 it suffices to consider finite-state PTS with $\Sigma = Act \cup \mathbb{N}_{\leq k+1}$[1] – as in the PTA of Figure 1, where $k = 2$.

These observations are crucial for our developments of non-symbolic MCTS-based methods for optimal reachability of PTA as we shall see.

## 3 Monte Carlo Tree Search

Monte-Carlo Tree Search (MCTS) is a family of algorithms that has been intensely studied in the last decades due to its high success in a range of domains, in particular - game playing. MCTS works on a search tree that grows in asymmetric fashion and in accordance to the results of random samples (or heuristics) that are used to estimate the reward (potential) of the action taken. The tree is iteratively expanded starting from the root node according to four steps:

- Selection: Descend down the tree by selecting the best child according to the chosen policy and until a first unexplored node is met. The selection process typically tries to balance between exploration (visiting promising nodes) and exploitation (visiting nodes with least visits).
- Expansion: Generate a successor of the given state according to the chosen policy.
- Simulation: Estimate the reward of the expanded node by performing simulations, aka roll-outs until the terminal node is reached. Typically, the performance of the algorithm can be drastically improved by a smart simulation strategy.
- Backpropagation: The estimated reward is ”backed up” through the tree to update reward estimates.

The first two steps (selection and expansion) are often referred to as *tree policy*, whereas the simulation (roll-out) step is called *default policy*. The algorithm does not

[1] $\mathbb{N}_{\leq k+1}$ are all natural numbers less than or equal to $k + 1$.

have a predefined termination condition and is typically running until either a computational budget (time, memory, etc.) is reached or some different, domain-specific condition is met.

Some of the characteristics that have made MCTS popular in other domains are particularly relevant in the setting of PTA. Tree policy allows to favor more promising regions of the model which over time leads to *asymmetric* tree growth. This helps alleviate the state-space explosion – the most prominent obstacle in model-checking. Moreover, MCTS being *aheuristic* – easily applicable without the need for domain-specific knowledge – it can be applied to any problem domain as long as it can be modelled as PTA.

We now introduce the formal definition of MCTS and then give the pseudocode of the algorithm – both adapted for the setting of PTA with non-strict guards. Recall that for PTA $A$ with non-strict guards and with maximum constant $k$ (to which clocks are compared) it suffices to consider the *finite* set of labels $\Sigma = Act \cup \mathbb{N}_{\leq k+1}$ to get a finite and complete PTS $F_A$. We let $\Sigma^*$ denote the language of finite (natural-valued and bounded) timed strings over $\Sigma$ and let $\epsilon \in \Sigma^*$ denote the empty string.

By convention we let $|\epsilon| = 0$ and otherwise define $|a_0 \dots a_n| = n$ to be the length of a word. We denote by $w_i \in \Sigma$ the $i$'th index of the word $w \in \Sigma^*$.

A timed word $w \in \Sigma^*$ of a PTS $T = (S, s_0, \Sigma, \rightarrow)$ is valid iff for $n = |w|$ we have:

$$s_0 \xrightarrow{w_0} s_1 \xrightarrow{w_1} \dots \xrightarrow{w_n} s_{n+1}$$

We let the function $O : \Sigma^* \rightharpoonup S$ denote the outcome of such a valid trace $w$ be $O(w) = s_{n+1}$. By convention we let $O(\epsilon) = s_0$.

**Definition 3 (Search Tree).** *We define $\Upsilon_T = (N, n_0, \Rightarrow)$ to be the search-tree for a natural- and bounded-valued PTS $T = (S, s_0, \Sigma, \rightarrow)$ as follows:*

- *$N = \Sigma^*$ is set of nodes,*
- *$n_0 = \epsilon$ is the root node, and*
- *$\Rightarrow \subseteq N \times \Sigma \times N$ is the transition relation such that $(n, b, n') \in \Rightarrow$ if and only if $nb = n'$ with $b \in \Sigma$ and $(O(n), b, O(n')) \in \rightarrow$.*

We delimit our attention to the most popular MCTS algorithm – the upper confidence bound for trees (UCT) [29]. UCT uses upper confidence bound (UCB1) formula as the tree policy, which addresses the exploration-exploitation dilemma of selecting the most promising paths by treating it as a multiarmed bandit problem. UCB1 makes a good candidate since it is guaranteed to be within a constant factor of the best bound for regret.

Let us define the global functions of the MCTS algorithm. Let $V : N \rightarrow \mathbb{N}$ assign the number of node visits, $Q : N \rightarrow \mathbb{R}$ assign the accumulative reward of the node, $P : N \rightarrow N$ maps to the parent of a node s.t. $P(n) = n'$ where $n' = n\alpha$ and $(n, \alpha, n') \in \Rightarrow$, and $Y_X : N \rightarrow \mathcal{P}(N)$ defines all children of the node that are valid according to the policy transition relation $\underset{X}{\Rightarrow}$, s.t. $Y_X(n) = \{n' \mid n \underset{X}{\Rightarrow} n'\}$. The definitions for each policy and respective transition relations are given in the following sections. Children are also partitioned into unexplored $(Y^U)$ and explored $(Y^E)$ ones s.t. $Y_X(n) = Y_X^U(n) \cup Y_X^E(n)$ and $Y_X^U(n) \cap Y_X^E(n) = \emptyset$.

Algorithm 1: The UCT Algorithm. This is a PTA-adapted redefinition of the Algorithm from [16].

```
function UctSearch(An initial state $s_0$, a set of goal-states $\mathcal{G}$, an empty set of solved nodes $\mathcal{S}$, an empty set of dead nodes $\mathcal{D}$, and a $C_p$ constant)
    $n_0 \leftarrow s_0$
    while budget remaining do
        $n \leftarrow$ TreePolicy($n_0, \mathcal{G}, C_p, \mathcal{S}, \mathcal{D}$)
        $\Delta \leftarrow$ DefaultPolicy($n, \mathcal{G}$)
        BackUp($n, \Delta$)
        if $O(n) \in \mathcal{G}$ then
            MarkSolved($n, \mathcal{S}$)
        if $O(n) \notin \mathcal{G}$ and $Y(n) = \emptyset$ then
            Prune($n, \mathcal{D}$)
    return BestChild($n_0, 0, \emptyset, \mathcal{D}$)
function TreePolicy($n$, $\mathcal{G}$, $C_p$)
    while $O(n) \notin \mathcal{G}$ do
        if $Y_X^U(n) \neq \emptyset$ then
            return Expand($n$)
        else
            $n \leftarrow$ BestChild($n, C_p, \mathcal{S}, \mathcal{D}$)
    return $n$
function Expand($n$)
    sample $n' \in Y_X^U(n)$
    $V(n') = Q(n') = 0$
    $Y_X^E(n') = \emptyset$
    add $n'$ to $Y_X^E(n)$
    return $n'$
function BestChild($n, C_p, \mathcal{S}, \mathcal{D}$)
    return $\underset{n' \in Y_X^E(n) \setminus (\mathcal{S} \cup \mathcal{D})}{\text{argmax}} \; Q_B \frac{V(n')}{Q(n')} + C\sqrt{\frac{\ln V(n)}{V(n')}}$
function DefaultPolicy($n, \mathcal{G}$)
    while $n \notin \mathcal{G}$ and within roll-out budget and
    $Y_X(n) \neq \emptyset$ do
        sample $n' \in Y_X(n)$ uniformly
        $n \leftarrow n'$
    return reward for $n$
function BackUp($n, reward$)
    while $n \neq \epsilon$ do
        $V(n) \leftarrow V(n) + 1$
        $Q(n) \leftarrow Q(n) + reward$
        $n \leftarrow P(n)$
function MarkSolved($n, \mathcal{S}$)
    while $n \in \mathcal{G}$ or $n' \in \mathcal{S}$ for all $n' \in Y_X(n)$ do
        $\mathcal{S} \leftarrow \mathcal{S} \cup \{n\}$
        $n \leftarrow P(n)$
function Prune($n, \mathcal{D}$)
    if $n \neq \epsilon$ and $Y_X(n) = \emptyset$ then
        Prune($P(n)$)
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{n\}$
```

Algorithm 1 gives a pseudocode for our PTA-adapted version of the UCT algorithm. The selection strategy used is a standard UCT formula (line 26). The expected reward of a node, determined by the exploitation factor $Q_B \frac{V(n')}{Q(n')}$, is inversely proportional to the average cost found so far which is normalized according to the currently best solution $Q_B$. The normalization ensures the reward value to be in range between 0 and 1 and thus supports domain (cost range) independence and eliminates the need for any prior knowledge about the reward distribution, which is also apriori unknown for PTAs. The significance of the exploration term is controlled by the value of $C$ constant.

Once a solution is found, we mark the given node terminal to avoid re-exploration (lines 7 and 38-41). As long as the underlying search-tree is complete (determined by the variant of $\underset{X}{\Rightarrow}$) , the algorithm is guaranteed to (eventually) provide an optimal solution given that one exists.

## 4 General PTA Challenges

**Infinite transition sequences:** MCTS algorithms have in large parts been developed for game playing, probabilistic planning or other, typically finite, state-space problems. However, in the setting of PTA, infinite transition sequences are possible, e.g. due to loops in the model. First and foremost it means that traditional roll-outs, directed at reward estimation, might never come to a halt. To overcome this problem we introduce a maximum budget for a roll-out (line 28). An example of the budget is an upper bound on maximum allowed steps that can be done in the default policy before the simulation is terminated.

**Reward evaluation:** In turn, capped roll-out length can pose a problem by introducing the need to evaluate non-terminal states. Fortunately, PTA contains all the necessary information needed to evaluate the current cost of any state, including non-terminals. We evaluate and back-propagate the reward regardless of whether the rollout has reached a terminal state.

**'Dead' states:** Apart from infinite transition sequences, it is possible to encounter states with no possible successors in PTA. In most MCTS algorithm domains such no successor states are also terminal states; however, it is not necessarily the case for PTA. This is an issue for UCT as it is not equipped to deal with such *dead* states. In UCT, a dead state can be encountered either during expansion or simulations step. For the latter we simply terminate the roll-out upon reaching a dead state (line 29). In case of the former, if UCT expands into a dead state, it must have highest so far expected reward. Simulating from a dead state will not generate any new information, resulting in that state being the best-so-far. To avoid computational overhead, we prune dead states and their parent states from the search tree (lines 9 and 42-45) until no dead states remain in the current branch of the tree.

## 5 Policies

In MCTS, the structure of the search tree is decided by the unfolding mechanism of the tree policy. The same unfolding strategy is also used during the simulation process of the default policy. In this section we discuss different unfolding strategies that we refer to as *policies*. The specific choice of policy can have a dramatic effect on the performance of MCTS (as we shall demonstrate in the experiments). In particular, for PTA, the search-tree transition function $\Rightarrow$ for the PTA in Figure 1 would for the state $(\ell_0, x = 0, y = 0)$ contain both the delay-action of 2 time units and the delay-action of 1 time unit (which would be repeatable), leading to the exact same configuration with the same total cost, namely $(\ell_0, x = 2, y = 2)$ at cost 10.

We thus explore both incomplete and complete policies, all restrictions over the full search-tree transition function $\Rightarrow$, with the latter category quarantining the existence of at least one optimal trace. Here, an incomplete policy does not retain the entire search-tree and does not guarantee preservation of an optimal solution. As the first policy, we introduce the Unit Delay Policy.

**Definition 4 (Unit Delay Policy).** *The transition function* $\underset{UDP}{\Rightarrow}$ *is given directly by* $\underset{UDP}{\Rightarrow} = (N \times (Act \cup \{1\}) \times N) \cap \Rightarrow$.

While the $UDP$ policy streamlines the application of delays, we observe a decreasing probability to pick larger delays. A child node (in tree and default policies) is chosen randomly between all available actions from that state and a delay of a single time unit; consequently, the probability for sequential choice of $d$ unit-delay transitions at state $s$, i.e. delaying $d$ time units, can be captured as follows:

$$Pr(s, d) = \left(\frac{1}{|Act_s| + 1}\right)^d$$

where $s \in S$, $d \in \mathbb{N}$ and assuming that all actions $Act_s$ are available from state $s$ at all times. If a state has actions that are only valid after a certain amount of time, then those actions are considerably less likely to be explored. We anticipate that such a skewed construction of the tree severely affects the ability of MCTS to find optimal solutions.

To alleviate this, we introduce a *Delay Sampling* policy (DSP) that allows to choose delays according to a more favorable probability distribution by enforcing a particular structure where delay and action transitions are always alternated. We also use this node layer alternation in the policies following the DSP policy giving a clear cut between transitioning by delay or action. Let $X : S \to \mathcal{P}(\mathbb{N})$ be a function that given a state returns a set of natural-valued delays w.r.t. to location-based constants, which includes the smallest possible delay, the largest possible delay, and a certain amount of delays from in between the bounds. We include only a subset of possible delays, which is limited to contain at most 100 values and at most $30\%$ of the number of possible values (excluding bounds). The set of possible delays is selected in an attempt to reduce potentially huge branching factor due to delay-actions as to direct the search towards more cost-promising paths. Notice that $X$ may change with each subsequent execution of the algorithm, but will not change during. Formally, DSP is defined as follows.

**Definition 5 (Delay Sampling Policy).** *The DSP policy* $\underset{DSP}{\Rightarrow}$ *is defined s.t. if* $(n, \alpha, n') \in \Rightarrow$ *then* $(n, \alpha, n') \in \underset{DSP}{\Rightarrow}$ *iff:*

- $n' = na$, $a \in Act$, $n = n''d$, $d \in \mathbb{N}$, *or*
- $n' = nd$, $d \in X(O(n))$, $a \in Act$ *and either* $n = \epsilon$ *or* $n = n''a$.

The policy solves the issue of uneven probability distribution for larger delays. However, it is incomplete in the function $X$ not guaranteeing preservation of key delay values. In addition, we note that the policy still considers a fair degree of delay values (up to 100), quickly leading to a significant degree of branching in the search-tree.

As an alternative, we introduce a policy with the behavior inspired by *Non-lazy schedules* of [1]. The idea behind non-laziness is to avoid unnecessary simultaneous idling of both jobs and corresponding resources. If the resource is available, the job should claim the resource unless some other job can also use it. In the latter case, the first job can be delayed to 'pass' the resource to the second job. We do not give the formal definition of Non-lazy schedules here to maintain readability and refer the interested reader to the mentioned paper for more details.

We introduce our Non-Lazy policy with delays restricted to being either zero, to mimic no delay, or a non-lazy delay, representing the smallest non-zero delay leading to some action becoming enabled, similarly to non-lazy schedules. In comparison to DSP this drastically reduces the breadth of the search tree to at most 2 children and in part alleviates the state-space explosion problem. Let $NLD : S \to \mathcal{P}(\mathbb{N})$ give a set of zero and non-lazy delay, and $A' = \{\alpha \in \Sigma \mid s \overset{\alpha}{\not\rightarrow}\}$ be a set of actions that are not immediately enabled from a given state.

$$\begin{aligned} NLD(s) = &\{0 \mid \exists \alpha \in \Sigma \text{ s.t. } s \xrightarrow{\alpha} s'\} \cup \\ &\qquad \{d' \mid d' = \underset{d \in \mathbb{N}_{>0}}{\mathbf{arg\,min}} \{\exists \alpha \in A' \text{ s.t. } s \xrightarrow{d} s'' \xrightarrow{\alpha} s'\}\} \end{aligned}$$

We now give a formal definition of the policy.

**Definition 6 (Non Lazy Policy).** *The NLP policy* $\underset{NLP}{\Rightarrow}$ *is defined s.t. if* $(n, \alpha, n') \in \Rightarrow$ *then* $(n, \alpha, n') \in \underset{NLP}{\Rightarrow}$ *iff:*

- $n' = na$, $a \in Act$, $n = n''d$, $d \in \mathbb{N}$, *or*
- $n' = nd$, $d \in NLD(O(n))$, $a \in Act$ *and either* $n = \epsilon$ *or* $n = n''a$.

In [1] it is shown that non-lazy schedulers preserve optimal solutions for Job-shop scheduling problems; however, this is not the case for all problems expressible as PTA – implying that the method is incomplete for general PTAs.

Lastly we introduce a policy inspired by *Randomized Reachability Analysis* heuristics from [28]. The idea is to consider action transitions and select delays based on availability range of the chosen action transition. This supports an equal probability distribution to traverse each individual action transition irrespective of its availability range in terms of delays and overall provides a 'fair' exploration. The authors of this heuristics demonstrated its efficiency in finding rare events. We here adapt the idea for finding cost-optimal plans under the heuristic that taking only the smallest possible delay for each transition will often lead to a lower cost.

We now give a formal definition of the Enabled Transition policy. Let $LB : S \times \Sigma \rightarrow \mathbb{N}$ give the lower bound of the transition's availability range over the actions of a given PTS. Simply put, $LB$ gives the smallest delay after which a certain action can be taken. Formally:

$$LB(s, \alpha) = \begin{cases} 0 \text{ if } \nexists\, d \in \mathbb{N} \text{ s.t. } s \xrightarrow{d} s' \xrightarrow{\alpha} s'' \\ \underset{d \in \mathbb{N}}{\mathbf{arg\,min}}\; s \xrightarrow{d} s_1 \xrightarrow{\alpha} s_2 \text{ otherwise} \end{cases}$$

**Definition 7 (Enabled Transition Policy).** *The ETP policy* $\underset{ETP}{\Rightarrow}$ *is defined s.t. if* $(n, \alpha, n') \in \Rightarrow$ *then* $(n, \alpha, n') \in \underset{ETP}{\Rightarrow}$ *iff:*

- $n' = na$, $a \in Act$, $d \in \mathbb{N}$, $n = n''d$, $d = LB(O(n''), a)$, *or*
- $n' = nd$, $a \in Act$, $d \in \{LB(O(n), a') \mid a' \in Act\}$ *and either* $n = n''a$ *or* $n = \epsilon$.

Similarly to NLP, ETP is also an incomplete policy but with more relaxed conditions allowing it to consider all eventually enabled (either now or after delay) actions from a given state.

## 6 Enhancements

To improve on the performance of the MCTS algorithm, we propose the following modifications over the standard MCTS algorithm presented in Algorithm 1.

***Building Rollouts***. The standard UCT algorithms uses rollouts to estimate the reward of a node, but strictly in a way s.t. the tree is not expanded, as to preserve memory. We propose to add a rollout to the tree under two conditions: if 1. a roll-out reaches the terminal state, and 2. it does so with the so-far-best cost. We denote such configuration as BR.

***Tree pruning with steps***. It can be beneficial to perform a step (advance the root) once 'enough' information has been gathered to ensure near-optimal action choice in the root of the search-tree. Two domain-independent techniques – *Absolute pruning* and *Relative pruning* – have been introduced in [25]. They have shown that the Absolute pruning in fact preserves the optimality of the search tree, but concluded that rather few nodes are actually being pruned due to pruning conditions being too strict. We will thus only study the Relative pruning technique.

We briefly recall the condition for Relative pruning (RP), which is dependent on the tunable parameter $\mu$.

**Condition 1 (Relative pruning condition)** *Node* $n_i$ *can be pruned if* $\exists j$ *such that* $V(n_j) > V(n_i) + \mu$, *where* $i \in \{1, \cdots, k\}, j \in \{1, \cdots, k\}, i \neq j$ *and for all* $i$ *we have* $(n, \alpha, n_i) \in \Rightarrow$ *with* $\alpha \in \Sigma$.

We also propose a simpler method of pruning based on a constant *stepping value*, i.e. a number of samples required in the current root-node before advancing the root of the tree. We denote this pruning technique Stepping pruning (SP).

## 7 Experiments

We perform experiments on three benchmarks:

1. Job-shop scheduling[2] problems,
2. Task graph scheduling[3] problems of [34] translated to PTA by [20], and
3. satellite mission scheduling problems [10, 31].

We select 120 Task graph models (of thousands) and use all 162 Job-shop models, and all of the satellite models. The largest Job-shop model contains 100 jobs using 20 machines and the largest Task graph consists of 300 tasks (83 chains) executed on 16 machines. To account for randomness of the MCTS and random-search methods, we report the average of 10 executions. For symbolic methods (which are deterministic) we only conduct one execution. All experiments are limited to 10 minutes and the best found solution is reported (if any). The experiments are conducted on AMD Opteron 6376 processors with frequency-scaling disabled running Debian with a Linux 5.8 kernel and limited to 8 GB of memory (except for experiments with TiaMo which is given sufficient memory).

***Solving using PDDL (Planning Domain Definition Language) Planners***. As a consequence of our restriction to natural-valued delays, it is possible to compile the PTA models into (classical, deterministic) planning problems and apply well-studied classic planning algorithms. To study this, we convert the Job-shop PTA models to PDDL 2.2 with action costs from PDDL 3.1 and use the Fast Downward[4] planner to find cost-efficient plans. We apply some classical algorithms, e.g. greedy best-first search with the FF heuristic for sub-optimal plans [24] and A$^*$ with LM-Cut for optimal plans [22]. However, the so-called grounding phase never terminates within the time and memory limit, even for the smallest Job-shop model consisting of 6 jobs and 6 machines. Scaling down the models further (by gradually removing jobs) reveals that the complexity of the model with 3 jobs already surpasses the capabilities of the planner to find a solution in allotted time. It is well-known that if the parameter-space of the actions in PDDL encoding grows large, which is the case for our models, the state-space suffers from an exponential explosion. We thus refrain from comparing to classical planners in the remainder of this section and leave comparison to more complex planners (e.g. temporal planning algorithms) to future work.

***Presentation of results***. In our graphs we present the relative performance of a method against *Best Known Solutions* (BKS) which is known for the Job-shop and Task graph problems. A $0\%$ deviation indicates that the BKS was found and a $10\%$ deviation denotes a solution that is $110\%$ of the BKS. We refer to the BKS as the reference value. For all but the last experiment we present the results over both benchmarks in one single plot. Figures 2-9 are plotted as “Cactus” or “Survival” plots. The y-axis shows the quality of the solution as ”% worse than the BKS” (Fig. 2-7). Each method is sorted

[2] https://github.com/tamy0612/JSPLIB
[3] https://github.com/marmux/spreadsheets
[4] https://www.fast-downward.org/HomePage

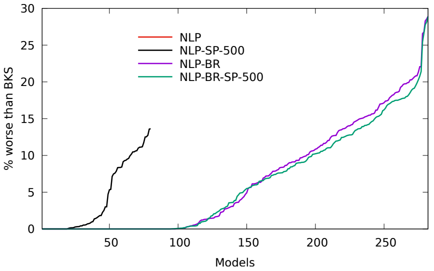

Fig. 2: The effect of BR and SP on the NLP policy.

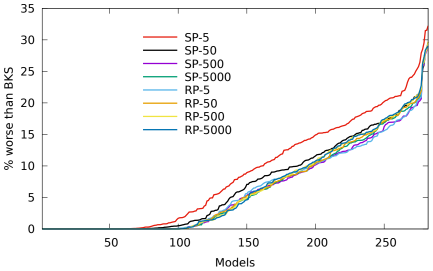

Fig. 3: Comparison of stepping values for NLP using BR and with $C_p = \sqrt{2}$.

individually, resulting in monotonically increasing lines. Therefore, data-points from different methods for a given x-value can be produced by different models, showcasing the general trend of each individual method over the benchmark.

We conduct the following sets of experiments:

- **Building Rollouts** where we construct the search-tree if a terminal node is found during rollout,
- **Impact of Stepping** where we experiment with pruning techniques,
- $C_p$ **Sensitivity** where we vary the exploration constant,
- **Policy Study** where we compare the proposed policies, and
- **Comparison w. Existing Methods** where we compare our best performing method with existing solvers for PTA, and
- a study of the methods on a set of more general PTA models stemming from the domain of satellite mission planning.

***Building Rollouts***. We initially study the impact of the BR enhancement as any configuration without this enhancement is unable to yield results for a significant portion of the benchmarks. As a representative configuration we here present the results with the NLP policy both with and without the SP pruning and the exploration constant $C$ fixed to $\sqrt{2}$. Other configurations demonstrate a similar tendency. We observe in Figure 2 that only versions with the BR optimization manage to find a solution to all the instances. In particular, we see that the version without both SP and BR produces no results at all (red line). We witness the effect of BR from the plot and see that the best performing configurations are deviating no more than 30% from the reference. In addition, for roughly 50% of the models, this deviation is less than 5%.

***Impact of Stepping***. In Figure 3 we compare different stepping sizes for SP and different upper-bounds number of visits ($\mu$) for RP. We here restrict the reported results to the BR variant of the NLP policy. We observe that SP is highly sensitive to the stepping size and see that the smallest step sizes result in worse performance due to a too rapid progression of the root-node while too high values fail to reduce the search-space to a feasible size. We observe a similar tendency with RP wrt. the sensitivity of the $\mu$-value, albeit to a lesser degree. Importantly we observe that SP (using a stepsize of $500$) and

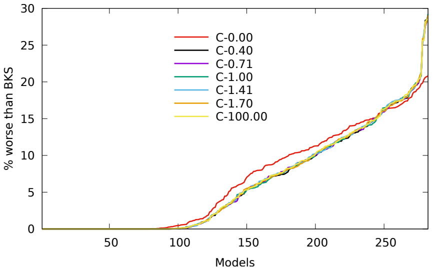

Fig. 4: Comparison of different $C_p$ values effect on NLP with BR and SP-500 options.

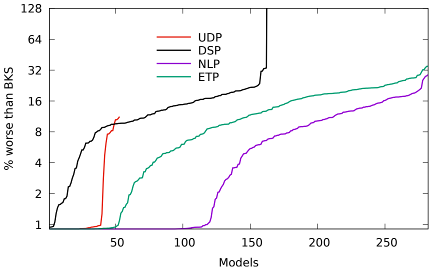

Fig. 5: Comparison of UDP, DSP, NLP, ETP policies with $C_p = \sqrt{2}$ and the best enhancements used: BR and SP-500.

RP (with $\mu = 5$) perform similarly well – and we delimit ourselves to reporting only on variants using SP in subsequent experiments.

While using $C_p = \sqrt{2}$ is often considered a good value to strike a balance between exploration and exploitation, we here study the sensitivity to changes in the $C_p$-value, in particular as our setting is a single-player setting. Specifically we can in Figure 4 observe the difference in performance when $C_p \in \{0, 0.4, \frac{1}{\sqrt{2}}, 1, \sqrt{2}, 1.70, 100\}$ where the value 100 is chosen arbitrarily as "a sufficiently large value" to force the algorithm to focus purely on exploration. From Figure 4 we observe that apart from $C_p = 0$, the choice of $C_p$ has little to no impact on the performance – likely due to the fact that our setting is a single player setting. Regarding $C_p = 0$, we conjecture that the effect observed stems from an intensive search around the initially found solution. For instances with a positive effect we believe that a (near-)optimal solution is found within the vicinity of *any* solution, where a negative effect indicate a larger difference between local minima in the search-space. While a small set of models clearly favor $C_p = 0$, we use $C_p = \sqrt{2}$ for the remainder of the experiments as it provides overall good performance and is the value recommended by literature.

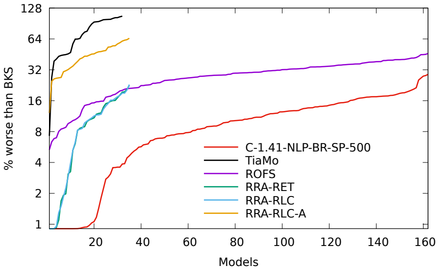

Fig. 6: Job-shop overview.

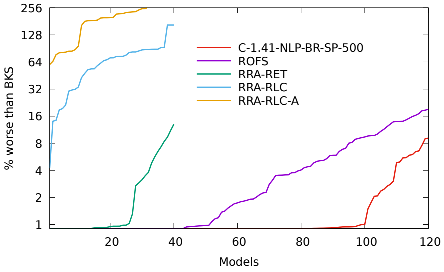

Fig. 7: Task graph overview.

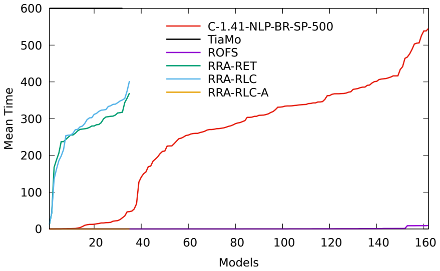

Fig. 8: Job-shop runtime overview.

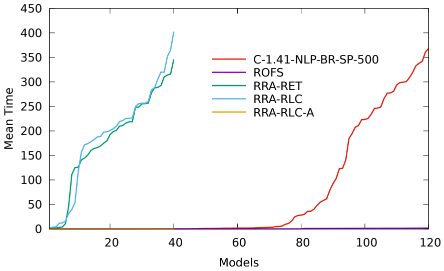

Fig. 9: Task graph runtime overview.

***Policy Study***. The summary on the performance of different policies is shown in Figure 5. Here we fix the configuration to use the BR and SP enhancements with a step-size of 500. We observe that UDP has the worst performance with less than $20\%$ of problem instances solved within the given time-frame - and significantly worse quality solutions. We believe this to be due to the low probability of selecting larger delays and the state-space explosion of having to consider all possible delays. While DSP is an improvement over UDP, it suffers from a similar problem in that the branching factor can explode leading to a performance degradation. Both NLP and ETP were able to solve all problem instances with near-optimal solutions of at most $28.88\%$ and $35.42\%$ away from the reference value, respectively, however with a clear advantage to NLP.

***Comparison w. Existing Methods***. Lastly we perform a comparison of our best configuration with other existing state-of-the-art solvers for PTA, namely Uppaal Cora and TiaMo. In addition, we have also adapted the *Randomized Reachability Analysis* (RRA) methods of [28] to search for optimal schedules rather than rare events. We experiment with several of the techniques proposed for RRA (RET, RLC and RLC-A) to search for optimal solutions. We refer the interested reader to the mentioned paper for more details.

In the case of Cora we use both the complete and optimal search-method as well as the incomplete *Random Optimal First Search* (ROFS) approach, which allows for a very lightweight search in a depth-first manner while choosing the most optimal action at each step but providing no guarantee wrt. optimality of the returned solution. It is important to note that both Cora (except for the ROFS version) and TiaMo are complete and able to find an optimal solution if given enough time and memory - and that both methods are relying on a symbolic representation of the search-space.

Figure 6 gives an overview of all the methods for Job-shop scheduling benchmark compared against BKS from [26]. Note that Cora has not managed to solve any instance for either of the benchmarks, primarily limited by the fact that it is a piece of 32bit software only capable of utilizing 4GB of memory. Unfortunately Cora does not provide anytime solutions in its current distribution. Both TiaMo and RRA methods solve less than $20\%$ of the instances, with TiaMo delivering sub-par solutions as it never completes the search within the time-limit, and thus provides only any-time solutions as they are found.

Table 1: Results for different PTA models of satellite problems. MCTS policies executed with $C_p = \sqrt{2}$, BR and SP-500 enhancements enabled. (oom = out of memory)

| | | DSP | NLP | ETP | ROFS | Cora |
|---|---|---|---|---|---|---|
| gomx3-1day | Mean cost | **186,007**<br>(±0.00%) | 188,408<br>(±1.95%) | **186,007**<br>(±0.00%) | 198,292<br>(±0.00%) | **186,007** |
| | Time | 40.2 | 49.8 | 61.5 | **0.05** | 5.12 |
| gomx3-2day | Mean cost | 442,190<br>(±0.04%) | 442,218<br>(±0.01%) | **442,080**<br>(±0.06%) | 478,002<br>(±0.17%) | oom |
| | Time | 223.3 | 268.0 | 230.7 | **0.05** | - |
| 5sat | Mean cost | 5,072,861<br>(±4.12%) | 5,961,014<br>(±1.72%) | **3,548,824**<br>(±0.77%) | 3,739,730<br>(±1.82%) | oom |
| | Time | 267.5 | 366.7 | 295.8 | **0.25** | - |
| 10sat | Mean cost | **5,632,414**<br>(±0.57%) | 6,130,961<br>(±0.68%) | nf | 5,687,131<br>(±2.47%) | oom |
| | Time | 232.4 | 232.6 | 600.0 | **0.56** | - |
| MaxData626 | Mean cost | nf | nf | nf | **7,458,522**<br>(±2.17%) | oom |
| | Time | 600.0 | 600.0 | 600.0 | **0.62** | - |

The ROFS algorithm of CORA outperforms both the random search and TiaMo in terms of solved instances, while having a drawback with the quality of the produced plans when compared to our MCTS implementation. In terms of time (Fig. 8, 9), the ROFS algorithm is the fastest overall, completing its search within single-digit seconds. We note that the overall quality of the schedules found by ROFS is within a surprisingly reasonable distance from the optimal, indicating that a greedy search strategy is well suited for the given benchmark. We observe the best performance of the proposed MCTS configuration using NLP, SP, BR and $C_p = \sqrt{2}$ and see a deviation of up to 28.88% of the BKS - with a median of deviations of no more than 10.3%. However, investigating the computation time, we can see that the best found solution is in the median produced at 289s and peaking at 546s.

The overview for the Task graph scheduling benchmark compared against BKS from [32] is shown in Figure 7. Due to its limited support of the PTA syntax, the TiaMo tool was not applicable. For over 80% of the benchmark (100/120 models) the solutions found by NLP are (near-)optimal with the quality of solutions of at most 1% away from BKS. For the rest of the benchmark the performance of NLP slightly worsens reaching at most 9.11% deviation from BKS. In general, the trends for different methods are very similar: RRA methods solve around 33% of models only, while ROFS finds solutions near instantly, but their quality degrades with increased model complexity.

***Satellite models***. Additionally, we experiment with two satellite cases - GomX-3 and Ulloriaq - designed, delivered, and operated by Danish satellite manufacturer GomSpace. The PTA models for these satellites have been developed in [10] and [31] studies, respectively, and analyzed with UPPAAL CORA (including ROFS). We show the results in Table 1, but exclude UDP as it produces no results within the time limit. For all models (but one) MCTS provides the best mean cost across all the methods; however, ROFS finds solutions up to 4 orders of magnitude faster and with a modest reduction of quality (up to 10% from the best MCTS method). We believe this is due to a generally small variance in the quality of solutions in the solution-space and the fact that ROFS performs only a single traversal of the model, immediately reporting the result upon reaching the terminal state. For "MaxData626" model MCTS methods timeout without a solution. Further experiments with an increased time-limit of 5 hours do not yield additional results indicating issues with the incompleteness of the methods rather than missing computation-time. The relative efficiency of the ROFS method demonstrates a potential for extending the MCTS method in the direction of a symbolic search, allowing for an efficient and complete MCTS tree-search method, and overcoming the current limitations of the discretized equivalents studied in this paper.

## 8 Conclusion

We have adapted the Monte Carlo Tree Search (MCTS) algorithm for the setting of problems described as Priced Timed Automata (PTA) – a formalism that can capture the behavior of a wide range of optimization problems such as resource-consumption or -allocation problems. PTA is a very versatile modeling formalism, facilitating more direct modeling of a problem domain. We introduced a number of complete and non-complete policies that act as unfolding mechanism and decide the structure of the tree. Some domain-independent enhancements to improve the performance and coverage of the algorithm are suggested.

We have evaluated the performance of our MCTS algorithm adapted to PTA on three benchmarks of Job-shop, Task graph and satellite mission scheduling problems and compared it against other state-of-the-art methods and tools. For the first two benchmarks, the results indicate that MCTS is able to find near-optimal solutions for all investigated problem instances. In general, we observed an up to 28.88% and 9.11% deviation (on average) from the best known solution in a set of Job-shop and Task graph scheduling problems, respectively. For satellite models, MCTS methods have found the best cost across all tested methods except for one model where only ROFS was able to produce results, hinting at issues with the incompleteness of MCTS methods.

All this suggests that MCTS is a promising alternative that copes well with the state-space explosion problem where other existing, exhaustive and complete methods perform poorly or fail. We note that the Random Optimal First Search strategy of the tool UPPAAL CORA performs well, even when compared to MCTS. The study of more symbolic approaches to MCTS for PTA is left as future work.

*Data availability.* A reproducibility artifact, which contains binaries, models and scripts to reproduce results can be found at https://doi.org/10.6084/m9.figshare.19772926

## References

1. Abdeddaïm, Y., Maler, O.: Job-Shop Scheduling Using Timed Automata? In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. pp. 478–492. Springer (2001)
2. Ahmad, W., Hölzenspies, P.K.F., Stoelinga, M., van de Pol, J.: Green Computing: Power Optimisation of VFI-Based Real-Time Multiprocessor Dataflow Applications. In: DSD 2015. pp. 271–275. IEEE Computer Society (2015). https://doi.org/10.1109/DSD.2015.59
3. Alur, R., Dill, D.: The theory of timed automata. In: de Bakker, J.W., Huizing, C., de Roever, W.P., Rozenberg, G. (eds.) Real-Time: Theory in Practice. pp. 45–73. Springer (1992)
4. Alur, R., La Torre, S., Pappas, G.J.: Optimal Paths in Weighted Timed Automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A. (eds.) HSCC 2001. pp. 49–62. Springer (2001)
5. Banharnsakun, A., Sirinaovakul, B., Achalakul, T.: Job Shop Scheduling with the Best-so-far ABC. Engineering Applications of Artificial Intelligence **25**(3), 583–593 (2012). https://doi.org/10.1016/j.engappai.2011.08.003
6. Behrmann, G., Fehnker, A., Hune, T., Larsen, K., Pettersson, P., Romijn, J.: Efficient Guiding Towards Cost-Optimality in UPPAAL. In: Margaria, T., Yi, W. (eds.) TACAS 21. pp. 174–188. Springer (2001)
7. Behrmann, G., Fehnker, A., Hune, T., Larsen, K., Pettersson, P., Romijn, J., Vaandrager, F.: Minimum-Cost Reachability for Priced Time Automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A. (eds.) HSCC 2001. pp. 147–161. Springer (2001), https://doi.org/10.1007/3-540-45351-2_15
8. Behrmann, G., Larsen, K.G., Rasmussen, J.I.: Optimal scheduling using priced timed automata. SIGMETRICS Perform. Eval. Rev. **32**(4), 34–40 (mar 2005). https://doi.org/10.1145/1059816.1059823
9. Behrmann, G., Larsen, K.G., Rasmussen, J.I.: Priced Timed Automata: Algorithms and Applications. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.P. (eds.) Formal Methods for Components and Objects. pp. 162–182. Springer (2005)
10. Bisgaard, M., Gerhardt, D., Hermanns, H., Krčál, J., Nies, G., Stenger, M.: Battery-aware scheduling in low orbit: the GomX–3 case. Formal Aspects of Computing **31**(2), 261–285 (2019)
11. Bøgholm, T., Larsen, K.G., Muñiz, M., Thomsen, B., Thomsen, L.L.: Analyzing Spreadsheets for Parallel Execution via Model Checking, pp. 27–35. Springer International Publishing, Cham (2019), https://doi.org/10.1007/978-3-030-22348-9_3
12. Bouyer, P., Brihaye, T., Bruyère, V., Raskin, J.: On the optimal reachability problem of weighted timed automata. Formal Methods Syst. Des. **31**(2), 135–175 (2007). https://doi.org/10.1007/s10703-007-0035-4
13. Bouyer, P., Colange, M., Markey, N.: Symbolic Optimal Reachability in Weighted Timed Automata. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. pp. 513–530. Springer (2016)
14. Bouyer, P., Fahrenberg, U., Larsen, K.G., Markey, N.: Quantitative analysis of real-time systems using priced timed automata. Commun. ACM **54**(9), 78–87 (2011). https://doi.org/10.1145/1995376.1995396
15. Bozga, M., Maler, O., Tripakis, S.: Efficient Verification of Timed Automata Using Dense and Discrete Time Semantics. In: Pierre, L., Kropf, T. (eds.) Correct Hardware Design and Verification Methods. pp. 125–141. Springer Berlin Heidelberg, Berlin, Heidelberg (1999), https://doi.org/10.1007/3-540-48153-2_11
16. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in Games **4**(1), 1–43 (2012). https://doi.org/10.1109/TCIAIG.2012.2186810

17. Čaušević, A., Seceleanu, C., Pettersson, P.: Checking correctness of services modeled as priced timed automata. In: International Symposium On Leveraging Applications of Formal Methods, Verification and Validation. pp. 308–322. Springer (2012)
18. Dirks, H.: Finding Optimal Plans for Domains with Restricted Continuous Effects with UPPAAL CORA. ICAPS 2005, American Association for Artificial Intelligence (2005)
19. Edelkamp, S.: Heuristic Search Planning with BDDs. In: PuK2000 (2000), http://www.puk-workshop.de/puk2000/papers/edelkamp.pdf
20. Ejsing, A., Jensen, M., Muñiz, M., Nørhave, J., Rechter, L.: Near Optimal Task Graph Scheduling with Priced Timed Automata and Priced Timed Markov Decision Processes (2020)
21. Geuze, N.: Energy management in smart grids using timed automata. Master's thesis, University of Twente (2019)
22. Helmert, M., Domshlak, C.: Landmarks, critical paths and abstractions: what's the difference anyway? In: Nineteenth International Conference on Automated Planning and Scheduling (2009)
23. Hermanns, H., Krcál, J., Nies, G.: How Is Your Satellite Doing? Battery Kinetics with Recharging and Uncertainty. Leibniz Trans. Embed. Syst. **4**(1), 04:1–04:28 (2017). https://doi.org/10.4230/LITES-v004-i001-a004
24. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research **14**, 253–302 (2001)
25. Huang, J., Liu, Z., Lu, B., Xiao, F.: Pruning in uct algorithm. In: 2010 International Conference on Technologies and Applications of Artificial Intelligence. pp. 177–181 (2010). https://doi.org/10.1109/TAAI.2010.38
26. Jain, A., Meeran, S.: Deterministic job-shop scheduling: Past, present and future. European Journal of Operational Research **113**(2), 390–434 (1999). https://doi.org/10.1016/S0377-2217(98)00113-1
27. Jongerden, M.R., Haverkort, B.R., Bohnenkamp, H.C., Katoen, J.: Maximizing system lifetime by battery scheduling. In: IEEE/IFIP Int. Conf. DSN 2009. pp. 63–72. IEEE Computer Society (2009). https://doi.org/10.1109/DSN.2009.5270351
28. Kiviriga, A., Larsen, K.G., Nyman, U.: Randomized Reachability Analysis in Uppaal: Fast Error Detection in Timed Systems. In: Lluch Lafuente, A., Mavridou, A. (eds.) FMICS 2021. pp. 149–166. Springer (2021)
29. Kocsis, L., Szepesvári, C.: Bandit Based Monte-Carlo Planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) Machine Learning: ECML 2006. pp. 282–293. Springer (2006)
30. Korvell, A., Degn, K.: Designing a Tool-Chain for Generating Battery-Aware Contact Plans Using UPPAAL. Aalborg University, Master Thesis (2019)
31. Kørvell, A., Degn, K.: Designing a Tool-Chain For Generating Battery-Aware Contact Plans Using UPPAAL (2019)
32. Laboratory, K.: Standard task graph set, https://www.kasahara.cs.waseda.ac.jp/schedule/index.html
33. Saddem-Yagoubi, R., Naud, O., Godary-Dejean, K., Crestani, D.: Model-Checking precision agriculture logistics: the case of the differential harvest. In: Discrete Event Systems. Springer (2020)
34. Tobita, T., Kasahara, H.: A standard task graph set for fair evaluation of multiprocessor scheduling algorithms. Journal of Scheduling **5**(5), 379–394 (2002). https://doi.org/10.1002/jos.116
35. Vulgarakis, A., Čaušević, A.: Applying REMES behavioral modeling to PLC systems. In: 2009 XXII International Symposium on Information, Communication and Automation Technologies. pp. 1–8. IEEE (2009)