QoS-Aware Approximate Real-time tasks execution on Multiprocessor Systems using PDDL+ Planning

Francesco Percassi, Sangeet Saha

School of Computing and Engineering, University of Huddersfield, UK
f.percassi@hud.ac.uk,s.saha@hud.ac.uk

Abstract

Historically, planning and scheduling were considered separate branches of computer science, addressing different problems. Planning concerns what actions to carry out to achieve a goal, whereas scheduling concerns determining when and how to perform actions to optimise a metric criterion. However, in many real-world applications, this distinction has become increasingly blurred, with planning and scheduling often being integrated effectively.

This paper considers a relevant approximate computing-based real-time task scheduling problem in which there are multiple variants of each task, sharing the same mandatory part, but with different optional components. In terms of quality of service (QoS), the longer the optional component for each task is, the higher the quality of the resulting application is. However, choosing the best variant may violate the given deadline due to its longer execution time. Therefore, there is a trade-off between the faster completion time (makespan time) versus the achieved QoS. In such a scenario, a planning algorithm can be designed to work with different types of given constraints to provide guarantees regarding the delivered QoS. To promote the use and integration of planning-based techniques in scheduling, we propose a translation of the above scheduling and task mapping problem into a planning problem expressed in the PDDL+ formalism.

Introduction

A real-time embedded system is one type of embedded system dedicated to particular applications with real-time constraints in an embedded environment. In such scenarios, the system's correctness depends on the precision of the results and the time they are produced. In today's real-time embedded systems, real-time applications are often represented as Precedence-constrained Task Graphs (PTGs). Precisely, the entire application consists of a collection of tasks (aka nodes) under precedence constraints and dependencies between tasks (Stavrinides and Karatza 2010).

For such critical systems, approximated results achieved in time are preferable to accurate results obtained after the deadline. For instance, in a real-time video application, initially, an imperfect but acceptable quality video is produced from the received data in each period (Roy and Raghunathan 2015). Further computation can be carried out for better results if additional processing time is available. Hence, current safety-critical real-time embedded systems demand a trade-off between time budget and task execution quality.

In the approximate computation approach, each task contains a mandatory part followed by an optional part. The mandatory part must be executed to completion to produce an acceptable result, while the optional part refines the generated result. By evaluating the number of processor cycles assigned to the optional part, the quality of service (QoS) can be defined (Cao et al. 2018).

In this QoS-aware context, each task of the application is equipped with multiple distinct implementations represented by different versions. Although all the versions of a task produce the same output by completing the mandatory part, their total execution length and accuracy of results may vary depending upon the amount of optional part executed. The higher version will return higher QoS by completing additional optional parts. However, results with higher accuracy can typically only be generated at the expense of more intensive computation, leading to deadline miss.

Some previous works (Cao et al. 2018; Saha et al. 2022) have focused on maximising the QoS of such an application through judicious selection of task versions so that execution of all the nodes in the task graph may be completed within the given deadline while satisfying all dependency, and resource (limited processor) related constraints, formalising this scheduling problem as an Integer Linear Programming (ILP).

Scheduling of tasks on multi-processors with timing constraints, precedence constraints, and resource constraints is a challenging problem. Planning algorithms may be a useful approach to solving those challenges. Planning and scheduling are historically considered as two independent branches addressing different problems which can be defined roughly as follows: *planning* concerns *what* actions to be taken to achieve a goal, whereas *scheduling* concerns determining *when* and *how* to perform actions to optimise a metric criterion. Despite this, in many real-world applications, this distinction has become more blurred (Barták 1999) and planning and scheduling are often integrated effectively (Gaudreault et al. 2011). Planning-based algorithms can provide assurances to tasks concerning the ability to meet time constraints. Generally, planning to determine schedulability is similar to the *admission control* mechanism in real-time systems. Depending on the required Quality of Service (QoS), a planning algorithm can be developed to satisfy the different

types of given constraints to guarantee the achieved QoS or to maximise the achievable QoS.

Throughout the years, the planning community has proposed several planning formalisms of increasing expressiveness, e.g., PDDL2.1 (Fox and Long 2003), PDDL+ (Fox and Long 2006) and PDDL3 (Gerevini et al. 2009), so that planning can be applied to a wider range of problems. PDDL2.1 was designed with the intention of enabling modelling of temporal and resource aspects. PDDL+ goes further, enabling the modelling of mixed discrete-continuous domains involving a logical and numeric component and providing a more natural way of modelling such temporal aspects. The core feature of PDDL+ is the sharp distinction between the agent, modelled as usual with actions, and the environment, modelled through *processes* and *events*. Processes are exogenous changes that arise when certain conditions hold and act continuously on the numeric variables when time flows. Events are instantaneous changes that act on the world when some conditions are triggered.

The introduction of consumable resources and temporal aspects of planning has reduced the gap that exists between planning and scheduling. Furthermore, in the formulation of the considered problem, it is possible to make decisions, i.e., to choose which version of a task to execute.

Due to the highlighted proximity between planning and scheduling in more advanced problems, in this paper, we propose a translation scheme to reformulate the QoS-Aware Approximate Real-time task execution problem (hereinafter shortened in QOS-ARTMS) into an equivalent PDDL+ problem.

The main contribution of this work is a novel translation that provides an (almost) solver-ready planning domain model. This reformulation highlights how the expressive capacity of modern planning formalisms is able to capture problems that natively include resources and temporal constraints. Furthermore, this translation enables the possibility of exploiting a broad spectrum of techniques designed for planning, such as forward heuristic search, local-search (Gerevini, Saetti, and Serina 2003; Hoffmann 2003), Satisfiability Modulo Theory problems (SMT) (Cashmore, Magazzeni, and Zehtabi 2020) to address the QOS-ARTMS problem.

System Model and Problem description

In this section we outline the formalisation of QOS-ARTMS problem according to (Cao et al. 2018). After that we outline the PDDL+ (Fox and Long 2006) problem, that is our target language, by adopting the assumptions and formalisation provided in (Shin and Davis 2005; Percassi, Scala, and Vallati 2021).

QOS-ARTMS formalisation

System Model This work assumes a homogeneous multiprocessor embedded processing platform. Thus, our system consists of ν resources, which correspond to processors.

Task Model We model a real-time application ($\mathcal{A}$) as a PTG, that is a Directed Acyclic Graph (DAG) defined as follows, $\mathcal{G} = \langle \mathcal{T}, \mathcal{E} \rangle$, where $\mathcal{T}$ is a set of tasks ($\mathcal{T} = \{T_i \mid 1 \leq i \leq |\mathcal{T}|\}$) and $\mathcal{E}$ is a set of directed edges ($\mathcal{E} = \{\langle T_i, T_j \rangle \mid 1 \leq i, j \leq |\mathcal{T}|; i \neq j\}$) representing the precedence relations between distinct pairs of tasks. An edge $\langle T_i, T_j \rangle$ refers to the fact that task T_j can begin execution only after the completion of T_i. A source task is a task with no predecessors and similarly, we can define, a *sink task* is the one without any successors. However, being a real-time application, the entire application ($\mathcal{A}$) must meet its deadline, denoted as Δ, by executing all the associated task nodes within the interval.

We have considered approximate computation tasks. Each task T_i ($1 \leq i \leq n$) contains a mandatory part with execution cycles M_i and an optional part with execution cycles O_i. Each task must execute M_i units to obtain the output of acceptable quality. The optional part O_i becomes ready for execution only when the mandatory part M_i is completed. The optional part refines and improves the result hence, it can be executed partially. The execution length l_i of task T_i can be defined as :

$$l_i = M_i + O_i \times \mu$$

where $\mu \in [0, 1]$ denotes the percentage of the optional part which is being executed. Thus, $\mu = 1$ denotes that we execute the entire O_i units and we will achieve the maximum possible accurate result. Note that, both M_i and O_i are natural numbers and μ is such that $O_i * \mu = O_i^j \in \mathbb{N}$.

It is further assumed that a task T_i may have k_i different versions; that is, $T_i = \{T_i^1, T_i^2, ..., T_i^{k_i}\}$. Although all versions of a task produce the same output by completing the mandatory part M_i their total execution length l_i, the accuracy of results may vary depending upon the amount of optional part is executed (i.e., different values of μ). Thus, the length (l_i^j) of j^{th} version of the task T_i can be defined as:

$$l_i^j = M_i + O_i \times \mu_i^j = M_i + O_i^j$$

Before proceeding we need to introduce the primitives for identifying the sink node (the final node to be performed) and, given a task $T_i \in \mathcal{T}$, the source tasks of T_i, the set of tasks that have to be mandatorily executed before T_i. Given a PTG $\mathcal{G}$, we denote the sink task of $\mathcal{G}$ as $sink(\mathcal{G})$, the unique task $T_i \in \mathcal{T}$ such that it does not exist an edge $\langle T_i, T_k \rangle \in \mathcal{E}$. Furthermore, given a task $T_i \in \mathcal{T}$ and a PTG $\mathcal{G}$, we define the source tasks set of T_i as:

$$\mathcal{S}(T_i, \mathcal{G}) = \{T_j : \langle T_j, T_k \rangle \in \mathcal{E} \wedge T_k = T_i\}.$$

Problem description We now present the required constraints to model this scheduling and allocation problem.

1. ***Unique Execute Start Time Constraint:*** Each task must start executing on a particular processor at a unique time step. The above constraint enforces the following for each task:
   - only one version will be selected for loading.
   - will start its execution on the processor at a unique time step.
   - can be mapped only to one processor.
2. **Resource Constraint:** At any time step t, the number of tasks executed in parallel is upper bounded by the number of processors.

3. ***Execution Dependency Constraint:*** Corresponding to each directed edge ($\langle T_i, T_j \rangle \in \mathcal{E}$) in the DAG, T_j must start its execution only after its predecessor, T_i finishes.
4. ***Deadline Constraint:*** To ensure that the application $\mathcal{A}$ meets its end-to-end absolute deadline Δ, the sink node $sink(\mathcal{G})$ must finish its execution within Δ.
5. ***Objective:*** The objective of the formulation is to choose a feasible solution that maximises QoS of the application through the appropriate choice of task versions (optional parts). Hence, the objective can be written as follows:

$$Maximize \quad QoS(\mathcal{A}) \tag{1}$$

$$QoS(\mathcal{A}) = \sum_{i=1}^{|\mathcal{T}|} O_i \times \mu_i^j \tag{2}$$

Finally, combining these elements, we formalise a QOS-ARTMS problem as a tuple $\mathcal{A} = \langle \mathcal{G} = \langle \mathcal{T}, \mathcal{E} \rangle, \nu, \Delta \rangle$, where $\mathcal{G}$ is a PTG, $\nu \in \mathbb{N}$ is the number of available resources and $\Delta \in \mathbb{N}$ is the deadline within the application has to be performed.

Example 1 (Running Example)**.** *Let $\mathcal{A} = \langle \mathcal{G} = \langle \mathcal{T}, \mathcal{E} \rangle, \nu = 2, \Delta = 100 \rangle$ be a real-time application where $\mathcal{G}$ is the TPG defined as $\mathcal{T} = \{T_1, T_2, T_3, T_4, T_5, T_6\}$ and $\mathcal{E} = \{\langle T_1, T_2 \rangle, \langle T_1, T_3 \rangle, \langle T_1, T_4 \rangle, \langle T_2, T_5 \rangle, \langle T_5, T_6 \rangle\}$. $\mathcal{A}$ encompasses the following available versions for each task, i.e., $T_1 = \{T_1^1\}$, $T_2 = \{T_2^1, T_2^2, T_2^3\}$, $T_3 = \{T_3^1\}$, $T_4 = \{T_4^1\}$, $T_5 = \{T_5^1\}$ and $T_6 = \{T_6^1\}$. The mandatory part has the same length for all tasks, i.e., $M_i = 15$ for each $i \in [1..6]$. For the optional part and for each $i \in \{1, 3, 4, 5, 6\}$ we have $O_i = 10$ and $\mu_i^1 = 1$ and then $O_i^1 = 10$. For T_2 we have $O_2 = 20$ with three different percentages of optional part, i.e., $\mu_2^1 = 0.2$, $\mu_2^2 = 0.5$ and $\mu_2^3 = 1$ and then $O_2^1 = 20 \times 0.2 = 4$, $O_2^2 = 10$ and $O_2^3 = 20$. The source tasks of $\mathcal{A}$ are defined as follows: $\mathcal{S}(T_1, \mathcal{G}) = \{\}$, $\mathcal{S}(T_2, \mathcal{G}) = \{T_1\}$, $\mathcal{S}(T_3, \mathcal{G}) = \{T_1\}$, $\mathcal{S}(T_3, \mathcal{G}) = \{T_1\}$, $\mathcal{S}(T_5, \mathcal{G}) = \{T_2\}$ and $\mathcal{S}(T_6, \mathcal{G}) = \{T_5, T_3, T_4\}$. Furthermore we have $sink(\mathcal{G}) = T_6$. Solving $\mathcal{A}$ consists in identifying, for each task, which version to use, the timing, i.e., at what time starting the task, such that the ordering constraints prescribed by $\mathcal{G}$ are satisfied, occupying at most $\nu = 2$ resources, and executing the sink task within deadline $\Delta = 10$. Figure 1 depicts a graphical representation of $\mathcal{A}$. Figure 2 depicts a possible schedule for $\mathcal{A}$ that is compatible with the precedence constraints prescribed by $\mathcal{G}$.*

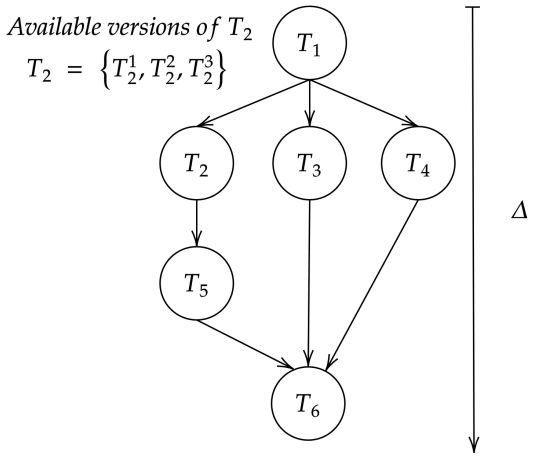

Figure 1: Graphical representation of a QOS-ARTMS problem $\mathcal{A}$.

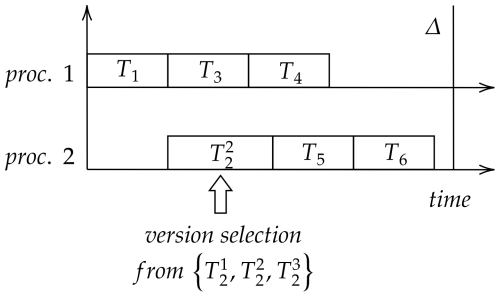

Figure 2: Graphical representation of a possible schedule for $\mathcal{A}$ compatible with its precedence constraints. Each task in $\{T_1, T_3, T_4, T_5, T_6\}$ admits only one version, while for T_2 there are three variants, i.e., $\{T_2^1, T_2^2, T_2^3\}$ and one must be chosen.

PDDL+ formalisation

In this section we report on the PDDL+ problem (Fox and Long 2006). We specify our problems using propositional formulas over numeric and Boolean conditions defined over sets of numeric and Boolean variables. A numeric condition is of the form $\langle \xi \bowtie 0 \rangle$ with ξ being a numeric expression, and $\bowtie \in \{\leq, <, =, >, \geq\}$. A Boolean condition is of the form $f = \{\top, \bot\}$ with f being a Boolean variable.

A PDDL+ planning problem Π is the tuple $\langle F, X, I, G, A, E, P \rangle$ in which each element is detailed in the following. F and X are the Boolean and numeric variables. Numeric variables take values from $\mathbb{R}$. I is the description of the initial state, expressed as a full assignment to all variables in X and F. G is the description of the goal, expressed as a formula. A and E are the sets of actions and events, respectively. Actions and events are pairs $\langle p, e \rangle$ where p is a formula and e is a set of conditional effects of the form $c \triangleright e$. Each conditional effect $c \triangleright e$ is such that (i) c is a formula and (ii) e is a set of Boolean assignments of the form $\langle f := \{\bot, \top\} \rangle$ or numeric assignments of the form $\langle \{asgn, inc, dec\}, x, \xi \rangle$ where ξ is a numeric expression. P is a set of processes. A process is a pair $\langle p, e' \rangle$ where p is a formula and e' is a set of numeric continuous effects expressed as pairs $\langle x, \xi \rangle$ where ξ is the net derivative of x.

Let $a = \langle p, e \rangle$ be an action/event/process, we use *pre*(a) to refer to the precondition p of a, and *eff*(a) to the effect e of a. Moreover, in the following we will use a, ρ, and ε to refer to a generic action, process, and event, respectively. In order to make the notation more concise, Boolean conditions and assignments of the form $\langle f = \bot \rangle$ ($\langle f := \bot \rangle$) and $\langle f = \top \rangle$ ($\langle f := \top \rangle$) are shortened to f and $\neg f$, and conditional effects of the form $\top \triangleright e$ are rewritten as e.

A PDDL+ plan π_t is a pair $\langle \pi, \langle t_s, t_e \rangle \rangle$ where: $\pi =$

$\langle a_1, t_1\rangle, ..., \langle a_n, t_n\rangle$ with $t_i \in \mathbb{Q}$ is a sequence of time-stamped actions; $\langle t_s, t_e\rangle$, with $t_s, t_e \in \mathbb{Q}$ and $t_s \leq t_e$, is the envelope within which π is performed. We say that π_t is well-formed iff $\forall\, i, j \in [1..n]$ and $i < j$, then $t_i \leq t_j$ and $t_s \leq t_i \leq t_e$ hold. Hereinafter we consider just well-formed plans.

In the following translation, we use a fragment of the expressive power of PDDL+ essentially to represent the flow of time. Suppose having a process $\rho = \langle \top, \{\langle \mathit{time}, 1\rangle\}\rangle$. This process should be understood as: when time flows, the precondition of ρ is always satisfied and the the variable *time*change according to $\mathit{time} = 1$ and then $\mathit{time}(t) = t$.

Intuitively, a PDDL+ problem consists in finding plans along a potentially infinite timeline, whilst conforming to a number of processes and events that may change the state of the world as time goes by. Both processes and events are applied as soon as their preconditions become satisfied (must transitions); differently, actions are decisions that need to be taken (may transitions).

Since QOS-ARTMS is natively formulated in discrete time, in the following we adopt the discrete PDDL+ semantics formalised by (Percassi, Scala, and Vallati 2021).

From QOS-ARTMS to PDDL+ planning

In this section, we describe how to reformulate a QOS-ARTMS problem into an equivalent PDDL+ planning task. In the proposed translation, tasks are initialised through actions. Processes are used in combination with events, according to the *start-process-stop* model (Fox and Long 2006), for modelling the temporal execution of tasks whose termination is captured through events. Further details are provided below.

Given a QOS-ARTMS task $\mathcal{A} = \langle \mathcal{G}, \nu, \Delta\rangle$, the corresponding PDDL+ problem is defined as $\Pi_{\text{QOS-ARTMS}} = \langle F, X, A, I, G, E, P\rangle$ where:

$$F = D \cup S \cup \overline{S}$$
$$D = \{d_{T_i} : 1 \leq i \leq |\mathcal{T}|\}$$
$$S = \{s_{T_i^j} : 1 \leq i \leq |\mathcal{T}|, 1 \leq j \leq |T_i|\}$$
$$\overline{S} = \{s_{T_i} : 1 \leq i \leq |\mathcal{T}|\}$$
$$X = X_{\mathit{time}} \cup \{\mathit{count}, \mathit{time}\}$$
$$X_{\mathit{time}} = \{\mathit{time}_{T_i} : 1 \leq i \leq |\mathcal{T}|\}$$
$$A = \{a_{T_i^j} : 1 \leq i \leq |\mathcal{T}|, 1 \leq j \leq |T_i|\}$$
$$\mathit{pre}(a_{T_i^j}) = \neg d_{T_i} \wedge \bigwedge_{T_z \in \mathcal{S}(T_i, \mathcal{G})} d_{T_z} \wedge \neg s_{T_i} \wedge \langle \mathit{count} + 1 \leq \nu\rangle$$
$$\mathit{eff}(a_{T_i^j}) = \{s_{T_i^j}, s_{T_i}, \langle \mathit{inc}, \mathit{count}, 1\rangle\}$$
$$I = \{\langle x := 0\rangle : x \in X_{\mathit{time}} \cup \{\mathit{time}, \mathit{count}\}\}$$
$$P = \{\rho_{T_i} : 1 \leq i \leq |\mathcal{T}|\} \cup \{\rho_{\mathit{time}}\}$$
$$\rho_{T_i} = \langle s_{T_i}, \{\langle \mathit{time}_{T_i}, 1\rangle\}\rangle$$
$$\rho_{\mathit{time}} = \langle \top, \langle \mathit{time}, 1\rangle\rangle$$
$$E = \{\varepsilon_{T_i^j} : 1 \leq i \leq |\mathcal{T}|, 1 \leq j \leq |T_i|\}$$
$$\mathit{pre}(\varepsilon_{T_i^j}) = \langle \mathit{time}_{T_i} = l_i^j\rangle \wedge s_{T_i^j}$$
$$\mathit{eff}(\varepsilon_{T_i^j}) = \{\neg s_{T_i^j}, \neg s_{T_i}, d_{T_i}, \langle \mathit{dec}, \mathit{count}, 1\rangle\}$$
$$G = d_{sink(\mathcal{G})} \wedge \langle \mathit{time} \leq \Delta\rangle$$

Variables The translation make use of the novel sets of variables D, S, $\overline{S}$ and X_{time}. The Boolean variables in D are used to attest that each task in T have been executed, S and $\overline{S}$ are used for correctly handling the execution of the tasks (explained below), while the numeric variables X_{time} are *local* timers used for keep track of the temporal advancement of the tasks. There are also two additional numeric variables, i.e., *count* and *time*. The first one is used for representing the number of occupied processors, while the second one is the *global* timer which is always linearly increased by the process ρ_{time}.

Initial State In the initial state both local and global timers, i.e., $X_{\mathit{time}} \cup \{\mathit{time}\}$ are initialised to 0. Since in the starting state, none of the processors is occupied, then also *count* is initialised to 0.

Tasks execution The execution of each task $T_i \in \mathcal{T}$ is modelled through a set of k_i actions, i.e., $\{a_{T_i^1}, ..., a_{T_i^{k_i}}\} \subset A$, a single process $\rho_{T_i} \in P$ and a set of k_i events, i.e., $\{\varepsilon_{T_i^1}, ..., \varepsilon_{T_i^{k_1}}\} \subset E$. Such actions are used for starting the execution of a version of T_i, the process ρ_{T_i} is simultaneously activated when one of the k_i action is performed and it is used for counting how much time is passed from the start. Finally, one and only one event, depending on the chosen version of T_i, ends the execution of the task. A more detailed explanation follows.

The execution of a task version T_i^j of T_i can be started in $t' \in [0, \Delta]$ by executing the action $a_{T_i^j}$ in a state $s \models \langle \mathit{time} = t'\rangle$ if the following preconditions holds. Firstly, all the source tasks of T_i have been executed, i.e., $s \models \bigwedge_{T_z \in \mathcal{S}(T_i)} d_{T_z}$. Secondly, to ensure that the resource constraint is satisfied, $a_{T_i^j}$ can be executed iff $s \models \langle \mathit{count} + 1 \leq \nu\rangle$. Once $a_{T_i^j}$ is executed, making s_{T_i} true, none of the versions of the task can be started. The execution of $a_{T_i^j}$ causes the increment of *count*, which models the occupation of a processor ($\langle \mathit{inc}, \mathit{count}, 1\rangle$). From t' onwards, independently from the chosen version of T_i, the related process ρ_{T_i} is activated and linearly increases the local timer time_{T_i}. The activation of ρ_{T_i} and the linear increase of time_{T_i} simulate the progress of the execution of task T_i^j over time. The process ρ_{T_i} is finally deactivated by the event $\varepsilon_{T_i^j} \in \{\varepsilon_{T_i^1}, ..., \varepsilon_{T_i^{k_i}}\}$, which is triggered when T_i^j have been executed for the corresponding time, that is l_i^j ($\langle \mathit{time}_{T_i} = l_i^j\rangle$). The aforementioned event, stops the process ($\neg s_{T_i^j} \in \mathit{eff}(\varepsilon_{T_i^j})$), attest the completion of the execution of T_i ($d_{T_i} \in \mathit{eff}(\varepsilon_{T_i^j})$) and frees the occupied resource by decreasing *count* ($\langle \mathit{dec}, \mathit{count}, 1\rangle \in \mathit{eff}(\varepsilon_{T_i^j})$. Once a task has been completed, in any version, it cannot be executed again (for each $j \in [1..k_i]$, then $\mathit{pre}(a_{T_i^j}) \models \neg d_{T_i}$).

Goal state The goal is expressed as a conjunction in which it is required that the sink node has been executed ($d_{sink(\mathcal{G})}$) and the global timer is lesser than the deadline ($\langle \mathit{time} \leq \Delta\rangle$).

Example 2 (QOS-ARTMS Translation - Continuing on Ex. 1)**.** *Let* $\mathcal{A} = \langle \mathcal{G}, \nu = 2, \Delta = 10\rangle$ *be the* QOS-ARTMS

problem provided in 1. Note that, according to Ex. 1 we have for each $i \in \{1,3,4,5,6\}$ *that* $l_i^1 = M_i + O_i \times \mu_i^j = 15 + 10 \times 1 = 25$*, while, for* $i = 2$*, we have* $l_2^1 = M_2 + O_2 \times \mu_2^1 = 15 + 20 \times 0.2 = 19$*,* $l_2^2 = 25$ *and* $l_2^3 = 30$*. The corresponding* PDDL+ *problem is defined as* $\Pi_{\text{QOS-ARTMS}} = \langle F, X, A, I, G, E, P\rangle$ *where:*

$$F = \bigcup_{i=1}^{|\mathcal{T}|=6} \{d_{T_i}\} \cup \bigcup_{i=1}^{|\mathcal{T}|=6} \bigcup_{j=1}^{|T_i|} \{s_{T_i^j}\} \cup \bigcup_{i=1}^{|\mathcal{T}|=6} \{s_{T_i}\}$$

$$X = \bigcup_{i=1}^{|\mathcal{T}|=6} \{\mathit{time}_{T_i}\} \cup \{\mathit{time}, \mathit{count}\}$$

$$A = \bigcup_{i=1}^{|\mathcal{T}|=6} \bigcup_{j=1}^{|T_i|} \{a_{T_i^j}\} \text{ where:}$$

- $\mathit{pre}(a_{T_1^1}) = \neg d_{T_1} \wedge \neg s_{T_1} \wedge \langle \mathit{count} + 1 \leq 2\rangle$ *and* $\mathit{eff}(a_{T_1^1}) = \{s_{T_1^1}, s_{T_1}, \langle \mathit{inc}, \mathit{count}, 1\rangle\}$*;*
- $\mathit{pre}(a_{T_2^1}) = \neg d_{T_2} \wedge d_{T_1} \wedge \neg s_{T_2} \wedge \langle \mathit{count} + 1 \leq 2\rangle$ *and* $\mathit{eff}(a_{T_2^1}) = \{s_{T_2^1}, s_{T_2}, \langle \mathit{inc}, \mathit{count}, 1\rangle\}$*;*
- $\mathit{pre}(a_{T_2^2}) = \neg d_{T_2} \wedge d_{T_1} \wedge \neg s_{T_2} \wedge \langle \mathit{count} + 1 \leq 2\rangle$ *and* $\mathit{eff}(a_{T_2^2}) = \{s_{T_2^2}, s_{T_2}, \langle \mathit{inc}, \mathit{count}, 1\rangle\}$*;*
- $\mathit{pre}(a_{T_2^3}) = \neg d_{T_2} \wedge d_{T_1} \wedge \neg s_{T_2} \wedge \langle \mathit{count} + 1 \leq 2\rangle$ *and* $\mathit{eff}(a_{T_2^3}) = \{s_{T_2^3}, s_{T_2}, \langle \mathit{inc}, \mathit{count}, 1\rangle\}$*;*
- $\mathit{pre}(a_{T_3^1}) = \neg d_{T_3} \wedge d_{T_1} \wedge \neg s_{T_3} \wedge \langle \mathit{count} + 1 \leq 2\rangle$ *and* $\mathit{eff}(a_{T_3^1}) = \{s_{T_3^1}, s_{T_3}, \langle \mathit{inc}, \mathit{count}, 1\rangle\}$*;*
- $\mathit{pre}(a_{T_4^1}) = \neg d_{T_4} \wedge d_{T_1} \wedge \neg s_{T_4} \wedge \langle \mathit{count} + 1 \leq 2\rangle$ *and* $\mathit{eff}(a_{T_4^1}) = \{s_{T_4^1}, s_{T_4}, \langle \mathit{inc}, \mathit{count}, 1\rangle\}$*;*
- $\mathit{pre}(a_{T_5^1}) = \neg d_{T_5} \wedge d_{T_2} \wedge \neg s_{T_5} \wedge \langle \mathit{count} + 1 \leq 2\rangle$ *and* $\mathit{eff}(a_{T_5^1}) = \{s_{T_5^1}, s_{T_5}, \langle \mathit{inc}, \mathit{count}, 1\rangle\}$*;*
- $\mathit{pre}(a_{T_6^1}) = \neg d_{T_6} \wedge d_{T_5} \wedge d_{T_3} \wedge d_{T_4} \wedge \neg s_{T_5} \wedge \langle \mathit{count} + 1 \leq 2\rangle$ *and* $\mathit{eff}(a_{T_5^1}) = \{s_{T_5^1}, s_{T_5}, \langle \mathit{inc}, \mathit{count}, 1\rangle\}$.

$$I = \bigcup_{i=1}^{|\mathcal{T}|=6} \{\langle \mathit{time}_{T_i} := 0\rangle\} \cup \{\langle \mathit{time} := 0\rangle, \langle \mathit{count} := 0\rangle\}$$

$$P = \bigcup_{i=1}^{|\mathcal{T}|=6} \{\langle s_{T_i}, \{\mathit{time}_{T_i}, 1\}\rangle \cup \{\rho_{\mathit{time}} = \langle \top, \langle \mathit{time}, 1\rangle\rangle\}\}$$

$$E = \{\varepsilon_{T_2^1}, \varepsilon_{T_2^2}, \varepsilon_{T_2^3}\} \cup \bigcup_{i \in \{1,3,4,5,6\}} \{\varepsilon_{T_i^1}\}$$

$$\varepsilon_{T_2^1} = \langle \langle t_{T_2} = 19\rangle \wedge s_{T_2^1}, \{\neg s_{T_2}, \neg s_{T_2^1}, d_{T_2}, \langle \mathit{dec}, \mathit{count}, 1\rangle\}\rangle$$

$$\varepsilon_{T_2^2} = \langle \langle t_{T_2} = 25\rangle \wedge s_{T_2^1}, \{\neg s_{T_2}, \neg s_{T_2^1}, d_{T_2}, \langle \mathit{dec}, \mathit{count}, 1\rangle\}\rangle$$

$$\varepsilon_{T_2^3} = \langle \langle t_{T_2} = 30\rangle \wedge s_{T_2^1}, \{\neg s_{T_2}, \neg s_{T_2^1}, d_{T_2}, \langle \mathit{dec}, \mathit{count}, 1\rangle\}\rangle$$

$$\varepsilon_{T_i^1} = \langle \langle t_{T_2} = 25\rangle \wedge s_{T_i^1}, \{\neg s_{T_i}, \neg s_{T_i^1}, d_{T_i}, \langle \mathit{dec}, \mathit{count}, 1\rangle\}\rangle$$

$$G = d_{sink(\mathcal{G})} \wedge \langle \mathit{time} \leq \Delta\rangle = d_{T_6} \wedge \langle \mathit{time} \leq 100\rangle$$

Note that the $\varepsilon_{T_i^1}$ *definition is provided for each* $i \in \{1,3,4,5,6\}$.

Handling QoS in $\Pi_{\text{QOS-ARTMS}}$

In this section, we explore some of the challenges posed by the resulting planning problem and how to effectively address it.

The PDDL+ problem $\Pi_{\text{QOS-ARTMS}}$ obtained through the proposed translation is unaware of the notion of quality of the computed solution, i.e., $QoS(\mathcal{A})$.

There are several planning systems that can reason over the PDDL+ problem $\Pi_{\text{QOS-ARTMS}}$, such as UPMURPHI (Penna et al. 2009), DINO (Piotrowski et al. 2016), SMTPLAN(Cashmore, Magazzeni, and Zehtabi 2020) and ENHSP (Scala, Haslum, and Thiébaux 2016). However, these planning engines exploit heuristics designed to minimise the makespan of the plan. Then, if we used these planning systems off-the-shelf for solving $\Pi_{\text{QOS-ARTMS}}$, we would compute plans neglecting the notion of QoS.

We have two possibilities for incorporating the quality of a solution within the PDDL+ formulation. Firstly, we can extend the PDDL+ formalism to take account the action costs under the form of a cost function that associate to each action $a \in A$ a real non-negative number, i.e., $c : A \to \mathbb{R}_0^+$. However, this choice poses two problems. The original problem $\mathcal{A}$ is formulated in terms of QoS, and so it has to be reformulated in terms of costs. Given a task $T_i \in \mathcal{T}$, each variant in $\{T_i^1, ..., T_i^{k_i}\}$ has a term which contributes to QoS, i.e., $\{O_i^1, ..., O_i^{k_i}\}$. Such terms can be reformulated as costs by using the following simple mapping: for each $j \in [1..k_i]$, then $c_i^j = \frac{\max\{O_i^1, ..., O_i^{k_i}\}}{O_i^j}$.

Now the PDDL+ problem can be extended by adding a cost function defined as follows:

$$c(a) = \begin{cases} c_i^j \text{ if } a = a_{T_i^j} \\ 0 \text{ otherwise.} \end{cases}$$

Now we can associate to each valid plan $\pi_t = \langle \pi, \langle t_s, t_e\rangle\rangle$ for $\Pi_{\text{QOS-ARTMS}}$ a cost function, i.e., $c(\pi_t) = \sum_{i=1}^{n=|\pi|} c(a_i)$, for which finding the optimal plan which minimises the cost function is equivalent to finding the best scheduling for $\mathcal{A}$ which maximises the QoS. This choice, however, involves the engineering of the PDDL+ planning engines in order to make them sensitive to costs, as in the past, the heuristics designed for classical planning, originally conceived for unit-cost planning tasks, have been generalised for handling the non-unit-cost case (Keyder and Geffner 2008).

PDDL2.1 provides explicit handling of customised metrics to sort the plans according to a qualitative criterion (Fox and Long 2003). Percassi, Scala, and Vallati 2021 recently proposed two translations for reformulating a PDDL+ problem into a discretised PDDL2.1 task. These translations, combined with the possibility of exploiting customised plan metrics, make these schemata suitable for addressing the QOS-ARTMS, doubly reformulated, especially since the QOS-ARTMS task is natively formulated in a discrete fashion. For making the resulting PDDL2.1 problem QoS-aware, it suffices just to add a numeric variable $cost$ and increase it according to the definition of $c(a)$ given above.

A simpler and alternative way could be to create handcrafted PDDL+ heuristics, encapsulating the knowledge about the QoS within them, to address the QOS-ARTMS problem as a forward search problem.

The exploration of how to effectively address the resulting planning problem is left to future works.

Conclusions

The QoS-Aware Approximate Real-time tasks execution on Multiprocessor Systems problem is a relevant and challenging scheduling problem. This problem involves different versions of the tasks to be executed which differently impact the quality of the resulting application. The scheduling must also be subject to constraints concerning the precedence over the tasks, the limited number of resources and a deadline within which to complete all the tasks.

In this article, we have proposed a novel translation that allows one to cast a given QOS-ARTMS problem into a PDDL+ planning task. The problem thus obtained can be addressed by applying a further reformulation present in the literature (Percassi, Scala, and Vallati 2021) to discretise a PDDL+ problem into a numeric one. The models obtained are almost ready to be used, but still require some dedicated effort to be able to deal with them properly. For the future, we see several avenues for extending this work, e.g., testing existing planning engines on the reformulated tasks and designing heuristics specifically targeted to address a QOS-ARTMS problem. Alternatively, since discrete PDDL+ is equivalent to PDDL2.1, it is possible to directly reformulate the QOS-ARTMS problem in PDDL2.1. We intend to investigate which planning formalism is best suited to model this kind of problem.

References

Barták, R. 1999. On the boundary of planning and scheduling: a study. In *Proc. of UK PlanSig 1999*, 28–39.

Cao, K.; Xu, G.; Zhou, J.; Wei, T.; Chen, M.; and Hu, S. 2018. QoS-Adaptive Approximate Real-Time Computation for Mobility-Aware IoT Lifetime Optimization. *IEEE Transactions on CAD*, 38(10): 1799–1810.

Cashmore, M.; Magazzeni, D.; and Zehtabi, P. 2020. Planning for Hybrid Systems via Satisfiability Modulo Theories. *JAIR*, 67: 235–283.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *JAIR*, 20: 61–124.

Fox, M.; and Long, D. 2006. Modelling Mixed Discrete-Continuous Domains for Planning. *JAIR*, 27: 235–297.

Gaudreault, J.; Frayret, J.-M.; Rousseau, A.; and D'Amours, S. 2011. Combined planning and scheduling in a divergent production system with co-production: A case study in the lumber industry. *COR*, 38(9): 1238–1250.

Gerevini, A.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *AIJ*, 173(5-6): 619–668.

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

Hoffmann, J. 2003. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *JAIR*, 20: 291–341.

Keyder, E.; and Geffner, H. 2008. Heuristics for Planning with Action Costs Revisited. In *Proc. of ECAI 2008*, volume 178, 588–592. IOS Press.

Penna, G. D.; Magazzeni, D.; Mercorio, F.; and Intrigila, B. 2009. UPMurphi: A Tool for Universal Planning on PDDL+ Problems. In *Proc. of ICAPS 2009*.

Percassi, F.; Scala, E.; and Vallati, M. 2021. Translations from Discretised PDDL+ to Numeric Planning. In *Proc. of ICAPS 2021*, 252–261.

Piotrowski, W. M.; Fox, M.; Long, D.; Magazzeni, D.; and Mercorio, F. 2016. Heuristic Planning for PDDL+ Domains. In Kambhampati, S., ed., *Proc. of IJCAI 2016*, 3213–3219.

Roy, K.; and Raghunathan, A. 2015. Approximate Computing: An Energy-Efficient Computing Technique for Error Resilient Applications. In *Proc. of ISVLSI 2015*, 473–475.

Saha, S.; Chakraborty, S.; Zhai, X.; Ehsan, S.; and McDonald-Maier, K. 2022. ACCURATE: Accuracy Maximization for Real-Time Multi-core systems with Energy Efficient Way-sharing Caches. *IEEE Transactions on CAD*.

Scala, E.; Haslum, P.; and Thiébaux, S. 2016. Heuristics for Numeric Planning via Subgoaling. In *Proc. of IJCAI 2016*, 3228–3234.

Shin, J.; and Davis, E. 2005. Processes and continuous change in a SAT-based planner. *AIJ*, 166(1-2): 194–253.

Stavrinides, G. L.; and Karatza, H. D. 2010. Scheduling multiple task graphs with end-to-end deadlines in distributed real-time systems utilizing imprecise computations. *J. Syst. Softw.*, 83(6): 1004–1014.