Resource-Optimal Scheduling Using Priced Timed Automata

J.I. Rasmussen[1], K.G. Larsen[1], and K. Subramani*[2]

[1] Department of Computer Science, Aalborg University, Denmark, {illum,kgl}@cs.auc.dk

[2] Department of Computer Science and Electrical Engineering, West Virginia University, USA, ksmani@csee.wvu.edu

Abstract. In this paper, we show how the simple structure of the linear programs encountered during symbolic minimum-cost reachability analysis of priced timed automata can be exploited in order to substantially improve the performance of the current algorithm. The idea is rooted in duality of linear programs and we show that each encountered linear program can be reduced to the dual problem of an instance of the min-cost flow problem. Thus, we only need to solve instances of the much simpler min-cost flow problem during minimum-cost reachability analysis. Experimental results using UPPAAL show a 70-80 percent performance gain. As a main application area, we show how to solve energy-optimal task graph scheduling problems using the framework of priced timed automata.

1 Introduction

Recently, solving real-time planning and scheduling problems using verification tools such as KRONOS, [9], and UPPAAL, [16], has shown promising results, [1, 6].

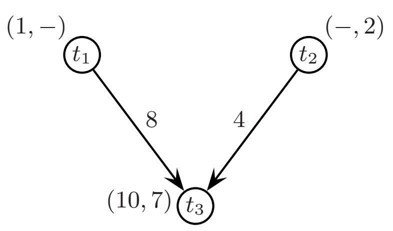
For addressing optimality constraints other than time (e.g. cost) priced timed automata (PTA) have been put forward, independently, as linearly priced timed automata in [15] and as weighted timed automata in [4]. The necessity for other optimality constraints is especially important within embedded systems where, for example, minimizing the overall energy consumption by embedded devices is imperative for their applicability.

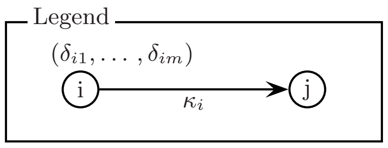
One such problem of minimizing energy consumption is that of energy-optimal task graph scheduling (TGS). This is the problem of scheduling a number of interdependent tasks onto a number heterogeneous processors that communicate through a single bus while minimizing the overall energy requirement and meeting an overall deadline. The interdependencies state that a task cannot execute until the results of all predecessors in the graph have been computed. Moreover, the results should be available in the sense that either each of the

* This research was conducted in part at Aalborg University, where the author was supported by a CISS Faculty Fellowship.

predecessors have been computed on the same processor, or on a different processor and the result has been broadcasted on the bus. An example task graph taken from [12] with three tasks is depicted in Figure 1. The task t_3 cannot start executing until the results of both tasks t_1 and t_2 are available. The available resources are two processors, p_1 and p_2, and a single bus with energy consumptions $\pi_1 = 4$, $\pi_2 = 3$, and $\pi_{bus} = 10$ per time unit when operating and 1 when idle.[1] The nodes in Figure 1 are annotated with their execution times on the processors, that is, t_1 can only execute on p_1, t_2 only on p_2, and t_3 can execute on both p_1 and p_2.

The energy-optimal schedule is achieved by letting t_1 and t_2 execute on p_1 and p_2, respectively, broadcast the result of t_2 on the bus, and execute t_3 on p_1, which consumes the total of 121 energy units. On the other hand, broadcasting the result of t_1 on the bus and executing t_3 on p_2 requires 141 energy units. Both schedules are depicted as Gantt charts in Figure 2.





$T = \{t_1, t_2, t_3\}$, $P = \{p_1, p_2\}$
$pre(t_1) = pre(t_2) = \emptyset$
$pre(t_3) = \{t_1, t_2\}$
$\pi_1 = 4, \pi_2 = 3$, $\pi_{bus} = 10$
$\tau_1 = 1, \tau_2 = 1$, $\tau_{bus} = 1$

Fig. 1. Example of an energy task graph with three tasks, two processors and a single bus.

Time-optimal task graph scheduling has received much attention in the research community as it is important in static, optimal scheduling of data independent algorithms onto digital signal processing architectures. For an overview of the proposed algorithms, an extensive comparison, evaluation, and benchmarks are provided in [13]. Recently, timed automata have in [1] been shown to be an efficient framework for modeling and solving TGS problems.

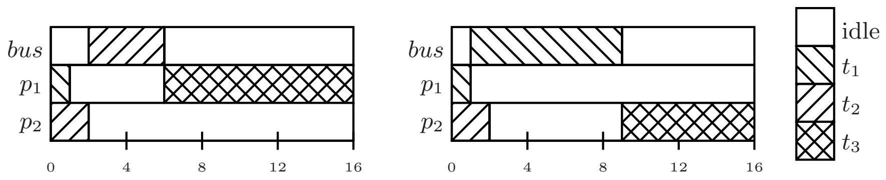


Fig. 2. a.) optimal schedule. b.) sub-optimal schedule

[1] We leave out units in this paper, but for real applications energy-consumption is, usually, measured in mW and execution times in ms.

For scheduling with cost extensions, timed automata are no longer sufficiently expressive. For cost-optimization problems, constraint programming, such as mixed integer linear programming (MILP), has been the preferred framework in the research community given the existence of efficient branch and bound algorithms for solving performance problems, [5,12]. Previously, the PTA approach to scheduling has shown to compete favorably with MILP for some problem instances, [15]. In this paper, we show how energy-optimal task graph problems can be modeled and solved using PTA.

The core of solving scheduling problems with PTA is symbolic minimum-cost reachability analysis. Within the reachability algorithm we frequently solve small, simple structured linear programs (LPs). As hinted in [15], reachability analysis might benefit significantly from exploiting the simple structure of these LPs since experimental results indicate that the current implementation in Uppaal, using the simplex method [8,11] spends 50-80 percent of the time during minimum-cost reachability, solving these LP problems.

In this paper, we show how to exploit the simple structure of the LPs in order to achieve an optimized algorithm. The idea comes from duality of linear programs and we show that each encountered LP can be reduced to the dual problem of an instance of the well-known min-cost flow problem, [2]. Thus, for each LP we can instead solve a min-cost flow problem using the much faster network simplex algorithm, [10]. Experimental results using a network simplex algorithm implementation [17] within Uppaal reduces the overall running-time of minimum-cost reachability analysis by 70-80 percent.

The rest of this paper is organized as follows. Section 2 gives a short introduction to the theory of PTA. The model for expressing energy-optimal TGS problems as PTA is covered in Section 3. Section 4 provides symbolic semantics for PTA and gives a branch and bound algorithm for performing symbolic minimum-cost reachability analysis. In Section 5 we show how the LP encountered during minimum-cost reachability analysis are reduced to dual min-cost flow problems. Section 6 provides experimental results. Finally, we conclude the paper and reflect on future work in Section 7.

2 Priced Timed Automata

Priced timed automata, [15,7,6,4], are extensions of timed automata, [3], with prices added to edges and locations. The interpretation of the price label is, that we associate a fixed price with taking transitions and a fixed price rate per time unit while delaying in locations. Intuitively, PTA are timed automata where each finite trace has an overall accumulated price.

Formally, let $\mathbb{C}$ be a set of real-valued clocks with power set $2^{\mathbb{C}}$. Then $\mathcal{B}(\mathbb{C})$ is the set of formulae obtained as conjunctions of atomic constraints of the form $x_i \bowtie n$ and $x_i - x_j \bowtie m$, where $x_i, x_j \in \mathbb{C}$, $n \in \mathbb{N}$, $m \in \mathbb{Z}$ and $\bowtie \in \{<, \leq, =, \geq, >\}$. We refer to the elements of $\mathcal{B}(\mathbb{C})$ as clock constraints.

Definition 1 (Priced Timed Automata). *A priced timed automaton over clocks* $\mathbb{C}$ *and actions Act is a 5-tuple* $(L, l_0, E, I, \mathcal{P})$ *where L is a finite set of* *locations, l_0 is the initial location, $E \subseteq L \times \mathcal{B}(\mathbb{C}) \times Act \times 2^{\mathbb{C}} \times L$ is the set of edges, $I : L \to \mathcal{B}(\mathbb{C})$ assigns invariants to locations, and $\mathcal{P} : (L \cup E) \to \mathbb{N}$ assigns prices to edges and locations. When $(l, g, a, r, l') \in E$ we write $l \xrightarrow{g,a,r} l'$.*

Actual clock values are represented as functions from $\mathbb{C}$ to the set of non-negative reals, $\mathbb{R}_{\geq 0}$, called clock valuations. The set of all clock valuations is denoted by $\mathbb{R}^{\mathbb{C}}$, and single clock valuations are ranged over by u, u' etc.

For a clock valuation, $u \in \mathbb{R}^{\mathbb{C}}$, and a clock constraint, $g \in \mathcal{B}(\mathbb{C})$, we write $u \in g$ when u satisfies all the constraints of g. For $d \in \mathbb{R}_{\geq 0}$, we define the operation $u + d$ to be the clock valuation that assigns $u(x) + d$ to all clocks, and the operation $u[r \to 0]$ to be the clock valuation that agrees with u for all clocks in $\mathbb{C} \backslash r$ and assigns zero to all clocks in r. Furthermore, u_0 is defined to be the clock valuation that assigns zero to all clocks.

The semantics of a PTA $\mathcal{A} = (L, l_0, E, I, \mathcal{P})$ is given in terms of a labeled transition system with state set $L \times \mathbb{R}^{\mathbb{C}}$, initial state (l_0, u_0), and label set $(E \cup \{\delta\}) \times \mathbb{R}_{\geq 0}$, with the transition relation defined as:

- $(l, u) \xrightarrow{\delta,p} (l, u + d)$ if $\forall\, 0 \leq d' \leq d \,.\, u + d' \in I(l)$, and $p = d \cdot \mathcal{P}(l)$,
- $(l, u) \xrightarrow{e,p} (l', u')$ if $e = (l, g, a, r, l') \in E, u \in g, u' = u[r \to 0]$, and $p = \mathcal{P}(e)$.

In other words, δ-transitions correspond to delays, and e-transition correspond to edges of the automaton. The associated value, p, gives the price of the action. When performing minimum-cost reachability analysis of PTA, we are interested in finite executions of the form $\alpha = (l_0, u_0) \xrightarrow{a_1,p_1} (l_1, u_1) \xrightarrow{a_2,p_2} \cdots \xrightarrow{a_n,p_n} (l_n, p_n)$, where l_n is some goal location. The cost of the execution α is the sum of all prices on the path, $\sum_{i=1}^{n} p_i$. The minimum cost of reaching a location l is defined to be the infimum cost of all finite executions ending in a state of the form (l, u).

3 Energy-Optimal Task Graph Scheduling Using PTA

In this section we formalize the model of energy-optimal task graph scheduling and show how to translate such an instance into a PTA.

Definition 2 (Energy Task Graph). *An energy task graph is a tuple $(T, P, pre, \delta, \kappa, \pi, \tau, d)$ where $T = \{t_1, \ldots, t_n\}$ is a set of tasks, $P = \{p_1, \ldots, p_m\}$ is a set of processors, $pre : T \to 2^T$ determines the set of predecessors of every task, $\delta : T \times P \hookrightarrow \mathbb{N}$ is the execution time for tasks on processors, $\kappa : T \to \mathbb{N}$ is the bus transfer time for every task, $\pi : P \cup \{bus\} \to \mathbb{N}$ is the energy consumption rate per time unit when processing/transferring for the processors and bus, $\tau : P \cup \{bus\} \to \mathbb{N}$ is the energy consumption rate when being idle, and d is the deadline.*

We use the shorthand notations δ_{ij}, κ_i, $\{\pi, \tau\}_i$, and $\{\pi, \tau\}_{bus}$ for $\delta(t_i, p_j)$, $\kappa(t_i)$, $\{\pi, \tau\}(p_i)$, and $\{\pi, \tau\}(bus)$, respectively.

Definition 3 (Feasible Schedule). *A feasible schedule, $\mathcal{S}$, for an energy task graph $(T, P, pre, \delta, \kappa, \pi, \tau, d)$ is a function $\mathcal{S} : T \to P \times \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \cup \{\infty\}$, such that:*

1. $\forall t_i \in T \,.\, \mathcal{S}(t_i) = (p_k, s, c) \Rightarrow \delta_{ik}$ *is defined and* $c \geq s + \delta_{ik}$
2. $\forall t_i, t_j \in T$ *with* $\mathcal{S}(t_i) = (p_k, s, c), \mathcal{S}(t_j) = (p_l, s', c')$.
   a) $t_j \in pre(t_i) \wedge c = \infty \Rightarrow p_k = p_l$
   b) $t_j \in pre(t_i) \Rightarrow s \geq \begin{cases} s' + \delta_{jl} + \kappa_j : p_k \neq p_l \\ s' + \delta_{jl} : p_k = p_l \end{cases}$
   c) $p_k = p_l \wedge s' \geq s \Rightarrow s' \geq s + \delta_{ik}$
   d) $c' \geq c \Rightarrow c' \geq c + \kappa_i$
3. $len(\mathcal{S}) <= d$

where $len(\mathcal{S}) = max\{s + \delta_{ik}, c + \kappa_i \,|\, (t_i, p_k, s, c) \in \mathcal{S}\}$.

We will often use the notation $(t_i, p_k, s, c) \in \mathcal{S}$ when $\mathcal{S}(t_i) = (p_k, s, c)$. Given a feasible schedule, $\mathcal{S}$, the intuitive understanding of an element (t_i, p_k, s, c) is that the execution of task t_i started on processor p_k at time s and was broadcasted on the bus at time c. If $c = \infty$ the result was never broadcasted. The interpretation of the constraints are (1) tasks can only execute on allowed processors and the result cannot be broadcasted until execution has terminated. (2a) when a task depends on the result of another task, these are either executed on the same machine or the result has been broadcasted. (2b) no task can begin executing until the results of all dependent tasks are available. (2cd) each processor/bus can only execute/transfer one task/result at any time and such operations cannot be preempted. (3) the schedule should meet the deadline. The processing time, $proc(p_k)$, of a processor $p_k \in P$ is defined as $\sum_{\{i \,|\, (t_i, p_k, s, c) \in \mathcal{S}\}} \delta_{ik}$ and similarly for the bus. The idle time, $idle(p_k)$, is then given as $len(\mathcal{S}) - proc(p_k)$. Now, we can define the cost of schedule as:

$$\mathsf{Cost}(\mathcal{S}) = \sum_{p_k \in P} \big(\pi_k \cdot proc(p_k) + \tau_k \cdot idle(p_k)\big) + \pi_{bus} \cdot proc(bus) + \tau_{bus} \cdot idle(bus) \quad (1)$$

A feasible schedule, $\mathcal{S}^*$, is optimal if for all feasible schedules, $\mathcal{S}$, $\mathsf{Cost}(\mathcal{S}^*) \leq \mathsf{Cost}(\mathcal{S})$.

Example 1. The optimal schedule (Figure 2a) for the energy task graph in Figure 1 corresponds to $\mathcal{S} = \{(t_1, p_1, 0, \infty), (t_2, p_2, 0, 2), (t_3, p_1, 6, \infty)\}$.

In the following we describe how to take any energy-optimal task graph problem instance and convert it into a network of priced timed automata. Following Uppaal syntax, we express a timed automaton as a composition of automata synchronizing on binary channels and using shared variables that can be compared in guards and updated on transitions. Consequently, each state in the global state space is associated with a location vector, variable assignment, and clock valuation. Here, we indicate by the binary variables $\mathsf{fin}[t_i]$ whether t_i has finished execution, by $\mathsf{act}[p_j]$ whether p_j (or bus) is being used, and by $\mathsf{res}[p_j][t_i]$ whether the result of t_i is available at p_j. The integer variable $\mathsf{d}[p_j]$ expresses the time p_j (or bus) is occupied.

For a given energy task graph $(T, P, pre, \delta, \kappa, \pi, d)$ we construct a PTA for each task, processor, and the bus.

Definition 4 (Processor/Bus Automaton). *Given a processor $p_k \in P$ of an energy task graph $(T, P, pre, \delta, \kappa, \pi, d)$, the automaton for p_k has a local clock c, action $\{p_k\}$ and is defined as $(L, l_0, E, I, \mathcal{P})$ where $L = \{s_0, s_1\}$, $l_0 = s_0$, $I(s_0) = \emptyset$, $I(s_1) = c \leq \mathsf{d}[p_k]$, $\mathcal{P}(s_0) = \tau_k$, $\mathcal{P}(s_1) = \pi_k$, and $e = \{(s_0, \emptyset, \{p_k\}, \{c := 0, \mathsf{act}[p_k] := 1\}, s_1), (s_1, c = \mathsf{d}[p_k], \{p_k\}, \{\mathsf{act}[p_k] := 0\}, s_0)\}$.*

The PTA for the bus is similar to the one for the processor and will not be defined explicitly. For simplicity we assign values to variables together with resetting clocks.

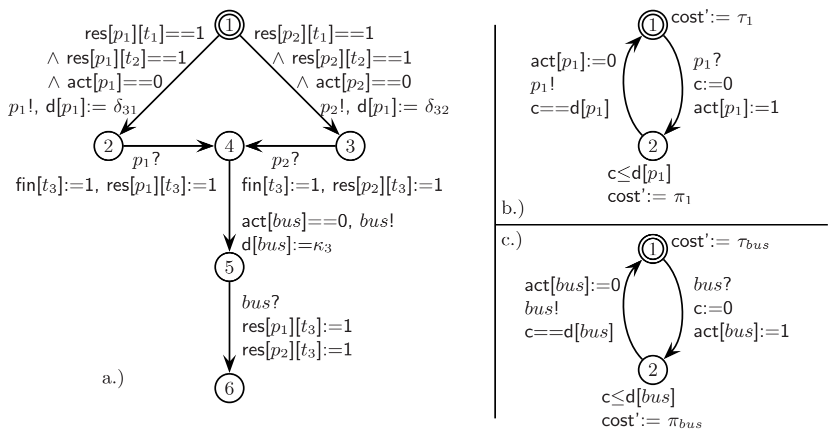


Fig. 3. Priced timed automata models for a.) task t_3 b.) processor p_1 c.) the bus.

Definition 5 (Task Automaton). *Given a task $t_i \in T$ of an energy task graph $(T, P, pre, \delta, \kappa, \pi, d)$, the automaton for t_i with $Act = \{p_k \mid \delta_{ik}$ is defined$\} \cup \{bus\}$ and is defined as $(L, l_0, E, I, \mathcal{P})$ where $L = \{s_0, done, bcing, bced\} \cup Act$, $l_0 = s_0$, $I(l) = \emptyset$ and $\mathcal{P}(l) = 0$ for all $l \in L$. E has the following transition for handling broadcast $(done, \mathsf{act}[bus] = 0, \{bus\}, \{\mathsf{d}[bus] := \kappa_i\}, bcing)$ and $(bcing, \emptyset, \{bus\}, \{\mathsf{res}[p_k][t_i] := 1 \mid p_k \in P\}, bced)$ and for each $p_k \in Act$ we have the following two transitions:*

- $(s_0, \mathsf{act}[p_k] = 0 \land \bigwedge_{t_j \in pre(t_i)} \mathsf{res}[p_k][t_j] = 1, \{p_k\}, \{\mathsf{d}[p_k] := \delta_{ik}\}, p_k)$
- $(p_k, \emptyset, \{p_k\}, \{\mathsf{fin}[t_i] := 1, \mathsf{res}[p_k][t_i] := 1\}, done)$

Now, the PTA for a energy task graph is the parallel composition of the bus automata, all task automata, and all processor automata. Furthermore, the cost of the optimal schedule is the minimum cost of reaching a location where $\bigwedge_{t_i \in T} \mathsf{fin}[t_i] = 1$.

Example 2. Figure 3 depicts PTA models for task t_3, processor p_1, and the bus of the energy task graph of Figure 1. The two outgoing transitions from the initial state (double circle state) of Figure 3a indicates that task t_3 can execute on both machines. The guard on transition (1,2) states that the result of both

t_1 and t_2 should reside at p_1, and that p_1 should be inactive. In such case, the automaton can synchronize with p_1 and set the occupation time of p_1 to δ_{31}. When p_1 has finished execution and is ready to synchronize, the automaton will set the finish flag for t_3, update res such that the result of t_3 resides at p_1 and proceed to state 4. From here the result can be broadcasted in a similar way by synchronizing with the bus. The result of this is that res is updated such that the result of t_3 resides at all processors.

Processor p_1 of Figure 3b has price (cost) τ_1 in the initial state (1) and π_1 is the processing state (2). When p_1 synchronizes with a task, it resets the local clock and sets its processing flag. The processor will remain in the processing state until the clock reaches the value of the occupation time variable for p_1. At this point, p_1 will synchronize with the occupying task and set the processing flag to false.

4 Symbolic Minimum-Cost Reachability Analysis

The semantics of PTA is an infinite state transition system. In order to effectively handle infinite state systems we require symbolic techniques, which operate on sets of states simultaneously.

Usually, reachability analysis is performed on symbolic states of the form (l, Z) where l is a location and $Z \in \mathcal{B}(\mathbb{C})$ is a zone. Semantically, (l, Z) is the collection of states (l, u) with $u \in Z$. We need two operations on a zone, Z, delay, $Z^{\uparrow}$, and reset with respect to a set of clocks r, $\{r\}Z$. The operations are defined as $Z^{\uparrow} = \{u + d \mid u \in Z, d \in \mathbb{R}_{\geq 0}\}$ and $\{r\}Z = \{u[r \mapsto 0] \mid u \in Z\}$.

Given a symbolic state (l, Z), we define the delay successor to be $post_{\delta}(l, Z) = (l, (Z \wedge I(l))^{\uparrow} \wedge I(l))$ and for any edge, $e = (l, g, a, r, l') \in E$, the successor with respect to e is defined as $post_e(l, Z) = (l', \{r\}(Z \wedge g))$. Let $Post(l, Z) = \{post_{\delta}(l', Z') \mid (l', Z') = post_e(l, Z), e \in E\}$ be the set of successors of (l, Z) by following an edge in the automaton and delaying in that state. The symbolic semantics of a timed automaton (L, l_0, E, I, P) can then be given as a transition system with state set of the form (l, Z), initial state $post_{\delta}(l_0, u_0)$, and transition relation $(l, Z) \rightarrow (l', Z')$ if and only if $(l', Z') \in Post(l, Z)$.

In the core of cost-optimal reachability analysis for PTAs, we apply a symbolic semantics on a priced extension of zones, [7]. For this purpose we define the offset, $\Delta_Z \in Z$, of a zone Z as the unique clock valuation, in the closure of the zone, with $\Delta_Z(x_i) \leq u(x_i)$ for all $x_i \in \mathbb{C}$ and $u \in Z$.

Definition 6 (Priced Zone). *A priced zone, $\mathcal{Z}$, is a tuple (Z, c, r), where Z is a zone, $c \in \mathbb{N}$ is the price of the offset, Δ_Z, and $r : \mathbb{C} \rightarrow \mathbb{Z}$ assigns a cost rate, $r(x)$, to each clock, $x \in \mathbb{C}$. For any $u \in Z$, the cost of u in $\mathcal{Z}$, $\mathsf{Cost}(u, \mathcal{Z})$, is defined as $k + \sum_{x \in \mathbb{C}} r(x) \cdot u(x)$ where $k = c - \sum_{x \in \mathbb{C}} r(x) \cdot \Delta_Z(x)$.*

For a priced zone $\mathcal{Z} = (Z, c, r)$ and a clock valuation u, we write $u \in \mathcal{Z}$ when $u \in Z$.

Example 3. Figure 4 depicts a priced zone $\mathcal{Z} = (Z, 8, r)$ over clocks $\{x_1, x_2\}$, with $r(x_1) = 3$ and $r(x_2) = -2$, $\Delta_Z(x_1) = \Delta_Z(x_2) = 1$, cost function $3x_1 - 2x_2 + 7$, and constraints $x_1 - x_2 \leq 1, 1 \leq x_2 \leq 3, x_1 \geq 1$.

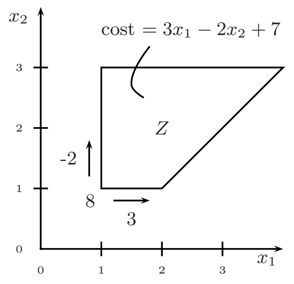


Fig. 4. A priced zone.

```
COST := $\infty$; PASSED := $\emptyset$
WAITING := $\{post_\delta(l_0, (u_0, 0, r_0))\}$
while WAITING $\neq \emptyset$
    pick $(l, \mathcal{Z})$ from WAITING
    if $\forall (l', \mathcal{Z}') \in$ PASSED: $\neg((l', \mathcal{Z}') \sqsubseteq (l, \mathcal{Z}))$ then
        add $(l, \mathcal{Z})$ to PASSED
        if $l \in G$ and $mincost(\mathcal{Z}) <$ COST then
            COST := $mincost(\mathcal{Z})$
            continue
        if $mincost(\mathcal{Z}) + remain(l, \mathcal{Z}) <$ COST then
            add $Post(l, \mathcal{Z})$ to WAITING
return COST
```

Fig. 5. Branch and bound algorithm.

Obviously, priced symbolic states should be pairs $(l, \mathcal{Z})$ where l is a location and $\mathcal{Z}$ is priced zone. Furthermore, we want the a-successor $(a \in \{e, \delta\})$ of a priced symbolic state $(l, \mathcal{Z})$ to be a state $(l', \mathcal{Z}')$ such that whenever $u \in \mathcal{Z}$ and $(l, u) \xrightarrow{a,p} (l', u')$ then $u' \in \mathcal{Z}'$ and $\mathsf{Cost}(u', \mathcal{Z}') = inf\{\mathsf{Cost}(u, \mathcal{Z}) + p \mid (l, u) \xrightarrow{a,p} (l', u')\}$.

Unfortunately, under these conditions, priced symbolic states are not directly closed under the $post_\delta$, $post_e$, and $Post$ operations. However, by applying the method outlined in [15], the above successor criteria can be met in a way such that both $post_\delta$ and $post_e$ on priced zones result in a finite union of priced zones. $Post$ on priced zones is then defined in the obvious way, $Post(l, \mathcal{Z}) = \{post_\delta(l', \mathcal{Z}') \mid (l', \mathcal{Z}') \in post_e(l, \mathcal{Z}), e \in E\}$.

The symbolic semantics for PTA can be stated similarly to the symbolic semantics for timed automata with $Post$ determining the transition relation and $(l_0, \mathcal{Z}_0) = post_\delta(l_0, (u_0, 0, r_0))$ as the initial state where r_0 has rate zero for all clocks.

Let $mincost(\mathcal{Z}) = inf\{\mathsf{Cost}(u, \mathcal{Z}) \mid u \in \mathcal{Z}\}$ denote the infimum cost over all clock valuations in $\mathcal{Z}$. The cheapest way of reaching some goal location l_g is then given as the minimum cost of all $mincost(\mathcal{Z})$ where there exists a finite path in the priced symbolic state space from the initial state to a state $(l_g, \mathcal{Z})$.

Now, we are ready to provide an algorithm for symbolic minimum-cost reachability analysis. The algorithm searches the symbolic state space based on a branch and bound approach and returns the minimum cost of reaching a location in a set of goal locations, G. The algorithm is depicted in Figure 5.

The algorithm uses a list of explored states, PASSED, and a list of states waiting to be explored, WAITING. Initially, the waiting list contains the initial state $(l_0, \mathcal{Z}_0)$. For every iteration of the while loop, we remove one element,

$(l, \mathcal{Z})$, from the list in some order. If none of the states in the passed list have the same location and are larger and cheaper ($\sqsubseteq$, defined below), we add $(l, \mathcal{Z})$ to the passed list. Then, if l is a goal locations and the minimum cost of the priced zone is the smallest encountered, we update the cheapest cost and skip to the next iteration[2]. We assume $remain(l, \mathcal{Z})$ to provide lower bound estimates on the cost of reaching a goal location from $(l, \mathcal{Z})$. We only compute the successors of $(l, \mathcal{Z})$ if there is a chance that we reach a goal location with a lower cost than previously encountered.

The algorithm terminates when there are no more states to explore and Cost will hold the lowest cost of reaching a goal location. Termination of the algorithm depends on the symbolic state space being well-quasi ordered under $\sqsubseteq$, [15]. Formally, the notion of 'bigger and cheaper' between priced symbolic states is defined as follows, $(l', \mathcal{Z}') \sqsubseteq (l, \mathcal{Z})$ if and only if $l = l'$, $Z \subseteq Z'$, and for all $u \in Z$, $\mathsf{Cost}(u, \mathcal{Z}') \leq \mathsf{Cost}(u, \mathcal{Z})$ where Z and Z' are the zone parts of $\mathcal{Z}$ and $\mathcal{Z}'$, respectively.

Given two priced zones $\mathcal{Z} = (Z, c, r)$ and $\mathcal{Z}' = (Z', c', r')$ where $Z \subseteq Z'$ we can decide whether $(l, \mathcal{Z}') \sqsubseteq (l, \mathcal{Z})$ by computing $mincost(\mathcal{Z}'')$ over a priced zone $\mathcal{Z}''$ with zone part Z and $\mathsf{Cost}(u, \mathcal{Z}'') = \mathsf{Cost}(u, \mathcal{Z}) - \mathsf{Cost}(u, \mathcal{Z}')$ for all $u \in Z$. If the result is larger than zero we know that $(l, \mathcal{Z}') \sqsubseteq (l, \mathcal{Z})$.

Thus, solving $mincost(\mathcal{Z})$ becomes a central aspect of the algorithm for symbolic minimum-cost reachability and the solution corresponds to the following linear program over clock variables.

Definition 7 (*mincost* LP). *Given a priced zone $\mathcal{Z} = (Z, c, r)$, with all strong inequalities relaxed, over clock variables $\mathbb{C}$, the mincost LP is defined as:*

$$\begin{array}{ll} \textbf{Minimize:} & \textbf{subject to:} \\ k + \sum_{x \in \mathbb{C}} r(x) \cdot x & Z \text{ and } \forall\, x \in \mathbb{C} \,.\, x \in \mathbb{R}_{\geq 0}, \end{array} \qquad (2)$$

Solutions to the *mincost* LP are given in terms of clock valuations and the objective function value of a given solution, u, is denoted by $z(u)$.

The *mincost* LP, when first described in [15], was solved using the simplex algorithm[3], [11]. As noted in [15], the *mincost* LP's occurring through minimum-cost reachability of practical problems are often very small, since the number of variables equals the number of clocks. Furthermore, the constraints of the LP's can be stated solely as clock difference constraints when using the difference bound matrix representation (DBM) outlined in, [14][4]. The simplex package applied, on the other hand, is tailored towards large, general LP's with up to 30000 variables and 50000 constraints, [8]. Experimental results show that, in the current implementation, 50-80 percent of the time spent during minimum-cost

[2] We are not concerned with the successors given the monotonicity of cost evolution.

[3] The implementation used the `lp_solve` package by Michel Berkelaar, `ftp://ftp.es.ele.tue.nl/pub/lp_solve`.

[4] By introducing a clock, x_0, whose value is always zero, any constraint of the form $x_i \bowtie n$ can be written as $x_i - x_0 \bowtie n$, $\bowtie \in \{\leq, \geq, =, <, >\}$.

reachability is used for solving *mincost* LP's. Thus, we may benefit significantly by exploiting the simple structure and/or the small size of the *mincost* LP's.

5 Minimum Cost Flow and Duality

In this section we describe a problem that is closely related to the *mincost* LP, namely the minimum cost flow problem (or min-cost flow). At the end of the section we show how the two problems relate through duality of linear programs.

Definition 8 (Min-Cost Flow Problem). *Let $G = \langle \mathcal{N}, \mathcal{A} \rangle$ be a directed graph with node set $\mathcal{N}$ and arc set $\mathcal{A} \subseteq \mathcal{N} \times \mathcal{N}$. With each node $i \in \mathcal{N}$ we associate a value $b_i \in \mathbb{N}$, such that $\sum_{i \in \mathcal{N}} b_i = 0$, with each arc $(i,j) \in \mathcal{A}$ we associate a cost c_{ij} and a variable y_{ij}. The min-cost flow problem is defined as:*

Minimize: $\displaystyle\sum_{(i,j)\in\mathcal{A}} c_{ij} y_{ij}$

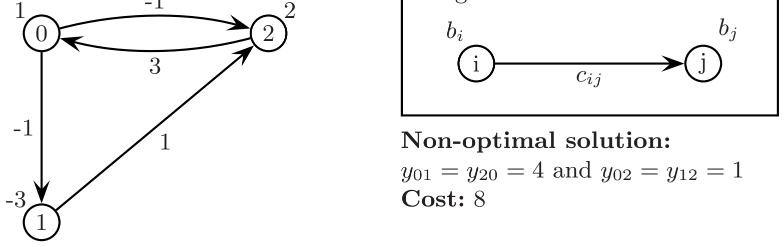
subject to:

$$\forall\, i \in \mathcal{N}\,.\; \sum_{\{j:(i,j)\in\mathcal{A}\}} y_{ij} - \sum_{\{j:(j,i)\in\mathcal{A}\}} y_{ji} = b_i, \qquad (3)$$

$$\forall\, (i,j) \in \mathcal{A}\,.\, y_{ij} \in \mathbb{R}_{\geq 0}. \qquad (4)$$

We call a node $i \in \mathcal{N}$ a supply node, demand node, or transshipment node depending on whether $b_i > 0$, $b_i < 0$, or $b_i = 0$, respectively. The intuitive interpretation of the min-cost flow problem is to find the cheapest assignment of flow to arcs, such that for each node the outflow minus the inflow equals the supply/demand of that node.

Example 4. Figure 6 depicts a min-cost flow problem with nodes $\{0,1,2\}$ and arcs $\{(0,1),(0,2),(1,2),(2,0)\}$. The supply/demand of the nodes are $b_0 = 1, b_2 = 2$, and $b_1 = -3$. The arc costs are $c_{01} = c_{02} = -1, c_{12} = 1$, and $c_{20} = 3$. One (non-optimal) solution that satisfies Equation 3 is to assign 4 units flow to arcs $(0,1)$ and $(2,0)$, and 1 to arcs $(0,2)$ and $(1,2)$. The cost of this solution is 8.



Non-optimal solution:
$y_{01} = y_{20} = 4$ and $y_{02} = y_{12} = 1$
Cost: 8

Fig. 6. Example min-cost flow problem. Node 1 is a demand node and nodes 0 and 2 are supply nodes.

For the solution of min-cost flow problems, a special and considerably faster adaptation of the general simplex algorithm, the network simplex algorithm, [10], has been proposed. Despite the worst-case exponential running-time of the network simplex algorithm it is often, in practice, faster than its polynomial time counterparts, [2].

Now, we show that instead of solving *mincost* LP's as general linear programs, we can instead solve related instances of the min-cost flow problem. The relation comes through duality of linear programming. For every linear program, called the primal, there is a closely related linear program, called the dual, and such primal/dual pairs share a number of properties. The property we exploit in this paper is the so-called strong duality theorem, [11,2]:

Theorem 1 (Strong Duality Theorem). *For every pair of primal/dual problems, if either problem has a bounded optimal solution, then so does the other and the optimal objective function values are identical.*

In other words, to obtain the optimal objective function value of a linear program, we can instead solve the dual problem since the optimal objective function values agree.

Definition 9 (Min-Cost Flow Dual). *Given a min-cost flow problem as stated in Definition 8, the dual problem is defined, over variables* x_i *where* $i \in \mathcal{N}$*, as:*

Maximize:	**subject to:**	
$\sum_{i \in \mathcal{N}} b_i x_i$	$\forall (i,j) \in \mathcal{A} \,.\, x_i - x_j \leq c_{ij}$	(5)
	$\forall i \in \mathcal{N} \,.\, x_i \in \mathbb{R}$	(6)

Obviously, the min-cost flow dual resembles the *mincost* LP using the DBM representation of zones. The differences are that the min-cost flow dual is a maximization problem, the decision variables range, unrestrictedly, over the reals, and the cost rates of all clocks must sum to zero. However, we can accommodate for these discrepancies by rewriting the *mincost* LP.

For this purpose we derive a linear program that is identical to the *mincost* LP in the sense, that for every solution to the *mincost* LP there is a solution in the new LP, which achieves the same objective function value, and vice versa.

Definition 10 (Relaxed *mincost* LP). *Given a priced zone* $\mathcal{Z} = (Z, c, r)$ *over a set of clock variables* $\mathbb{C} \cup \{x_0\}$*, the relaxed mincost LP is defined as*[5]

Minimize:	**subject to:**	
$k + \sum_{x \in \mathbb{C}} r(x) \cdot (x - x_0)$	Z^* and $\forall x \in \mathbb{C} \cup \{x_0\} \,.\, x \in \mathbb{R}$	(7)

[5] Z^* denotes the representation of a zone, Z, using only difference constraints by introducing a clock, x_0. [14]. Furthermore, all strong inequalities have been relaxed.

Solutions to the relaxed *mincost* LP are given in terms of an assignment function $v : \mathbb{C} \cup \{x_0\} \rightarrow \mathbb{R}$. The objective function value of a given solution, v, is denoted by $z(v)$. The relaxed *mincost* LP has a number of interesting properties. The proofs of these properties are straight forward and can be found in, [18]

Property 1. If u is a solution to the *mincost* LP, then v is a solution to the relaxed *mincost* LP, where $v(x) = u(x)$ for all $x \in \mathbb{C}$ and $v(x_0) = 0$. Furthermore, $z(u) = z(v)$.

Property 2. If v is a solution to the relaxed *mincost* LP, then u is a solution to the *mincost* LP, where $u(x) = v(x) - v(x_0)$ for all $x \in \mathbb{C}$. Furthermore, $z(v) = z(u)$.

As a consequence of Properties 1 and 2 the optimal objective function values of the *mincost* LP and the relaxed *mincost* LP agree. In other words, when determining the lowest price of a priced zone, we can choose to solve either LP.

In order for the relaxed *mincost* LP to be a min-cost flow dual, we need to change it into a maximization problem. However, this is trivial as minimizing a function corresponds to maximizing the negated function. That is, we can negate the objective function of the relaxed *mincost* LP, solve it as a maximization problem, and negate the result.

Finally, we need to verify that the sum of all supply/demand in the primal min-cost flow problem sum to zero. This fact follows immediately from the objective function of the relaxed *mincost* LP, as the factor of x_0 is the negated sum of all cost rates of $x_i \in \mathbb{C}$.

We conclude that instead of solving the *mincost* LP, we can solve the primal min-cost flow problem of the maximization version of the relaxed *mincost* LP. This technique requires us to negate the result received from the min-cost flow problem and add the constant k in order to obtain the correct solution.

Theorem 2. *Given a priced zone $\mathcal{Z} = (Z, c, r)$, the corresponding min-cost flow problem is obtained as: For each clock $x_i \in \mathbb{C}$ create a node x_i and set $b_{x_i} = -r(x_i)$, create a node x_0 for the zero clock with $b_{x_0} = \sum_{x_i \in \mathbb{C}} r(x_i)$, and for every constraint $x_i - x_j \leq m$ in Z^* make an arc from node x_i to x_j with cost $c_{x_i x_j} = m$. The solution to the mincost LP is the negated solution to the min-cost flow problem plus k.*

Example 5. To illustrate the above technique we show that the min-cost flow problem of Figure 6 is, in fact, the primal problem of the relaxed *mincost* LP over the priced zone in Figure 4. For each of the clock variables x_0, x_1, and x_2 we have the nodes $0, 1$, 2, respectively. The constraints of Z^* are $x_0 - x_1 \leq -1, x_0 - x_2 \leq -1, x_2 - x_0 \leq 3$, and $x_1 - x_2 \leq 1$, each of which corresponds to an arc with appropriate cost in the graph. The supply/demands of the nodes are given as $1, -3$, and 2, respectively, which obviously equals the negated cost rates and sum to zero.

Furthermore, the optimal solution to the min-cost flow graph is $x_{20} = 2, x_{01} = 3$, and $x_{02} = x_{12} = 0$ and has cost 3. By negating the result and adding the constant $k = 7$, we obtain the optimal solution (i.e. 4) to the *mincost* LP given in Example 3.

Note that when solving the *mincost* LP, the number of variables equals the number of clocks, and when solving the dual min-cost flow problem, the number of variables equals the number of clock constraints. Thus, we are interested in reducing the number of constraints. For this purpose, we use the algorithm provided in [14] that computes a zone representation with a minimal set of constraints.

6 Experimental Results

In this section we provide the experimental results obtained for energy-optimal task graph scheduling using PTA.

For conducting the experiments we have taken the Standard Task Graph Set[6] of [19] and added communication costs, energy consumptions, and restricting task execution to a subset of the processors. The results are summarized in Table 1. The missing entries indicate that we were unable to create a problem instance that could be solved within the available memory. Simplex and Net-Simplex refer to the running-times using the standard simplex algorithm [8] and network simplex algorithm [17], respectively. The performance gain is given as Speedup and Threshold shows the time needed to compute a feasible schedule within 10 percent of the optimal.

The results in Table 1 show that the running-time of symbolic minimum-cost reachability can be improved by 70-80 percent[7] by solving the min-cost flow dual problem instead of the *mincost* LP. The results are similar when performing the same experiments on the examples reported in [15].

Given the computational complexity of (energy-optimal) task graph scheduling the, key to solving large problem instances is to guide the search towards approximate solutions as fast as possible, [13]. Unfortunately, we have been unable to find energy task graphs with known optimal solutions to use for experimenting with heuristic-based guiding techniques. However, using a random-depth first search order, Table 1 shows that even for most problems, we can obtain a solution within a 10 percent margin using only a fraction of the time need for obtaining the optimum.

Since we were unable find comparable results using other algorithms, it is hard to conclude on the competitiveness energy-optimal TGS using PTA. However, the results in [15] show that the PTA approach to cost-optimal scheduling is competitive with MILP. Furthermore, [1] provides results showing that we can efficiently find approximate solutions to TGS with timed automata using guiding heuristics. For these reasons, we are positive that using more involved

[6] The STG set is available at `http://www.kasahara.elec.waseda.ac.jp/schedule/`.

[7] We disregard the results where both algorithms terminate within one second, since there is an overhead at startup where the input is passed and converted before execution begins.

Table 1. Experimental results for 19 energy task graph instances. The results were obtained on a PentiumM 1.2GHz with 512MB RAM. Done indicates that the algorithm found the optimal and terminated within 1 sec.

Processors	Tasks	5	7	9	10	11	12
2	Simplex	0.438s	2.734s	24.889s	127.318s	15.345s	844.804s
	NetSimplex	0.205s	0.592s	4.565s	27.890s	3.041s	181.997s
	Speedup (%)	**53.2**	**78.3**	**81.7**	**78.1**	**80.1**	**78.5**
	Threshold	done	done	2.730s	0.694s	1.297s	9.663s
3	Simplex	0.466s	10.850s	27.039s	155.823s	197.141s	
	NetSimplex	0.159s	2.618s	5.235s	36.403s	56.270s	
	Speedup (%)	**65.9**	**75.9**	**80.6**	**76.6**	**71.5**	
	Threshold	done	0.392s	5.768s	1.121s	0.141s	
4	Simplex	1.827s	15.049s	31.080s	450.859s	302.062s	
	NetSimplex	0.426s	3.583s	7.006s	106.804s	87.685s	
	Speedup (%)	**76.7**	**76.2**	**77.5**	**76.3**	**71.0**	
	Threshold	done	0.593s	0.779s	4.616s	17.85s	
5	Simplex	4.122s	20.603s	44.882s			
	NetSimplex	0.896s	5.104s	10.690s			
	Speedup (%)	**78.3**	**75.2**	**76.2**			
	Threshold	done	0.476s	5.099s			

guiding heuristics together with good *remain* estimates, PTA will be a competitive approach to solving energy-optimal TGS problems.

7 Conclusions and Future Work

In this paper we have shown how to exploit the structure of the LPs solved during symbolic minimum-cost reachability through a reduction to the min-cost flow problem, thus providing an answer to the question put forward in [15]. The current implementation in Uppaal uses a simplex algorithm to solve these LPs. Experimental results show that solving the related min-cost flow instances with a network simplex algorithm instead, reduces the overall running-time of the reachability algorithm by 70-80 percent. In particular, we have shown how to solve energy-optimal TGS problems with PTA, and through initial experimental results we believe that PTA is a competitive approach to solving such problems. Additional experiments on e.g. the aircraft landing problem, [15], indicate that the performance gain generalizes to all PTA minimum-cost reachability problems.

To improve the competitiveness of PTA in energy-optimal TGS, we need to develop more elaborate guiding heuristics and remaining estimates in order to efficiently guide the search in the direction of approximate solutions.

An interesting alternative, which can potentially provide considerable speedup, is to identify abstractions that are exact w.r.t. optimal reachability, e.g. a suitable adaption of the domination points techniques used in [1].

Another promising approach is be to develop a compositional technique for finding either approximate or exact optimal solutions. The idea is to exploit the (in)dependency structure often found in large scale real problems.

References

1. Yasmina Abdeddaim, Abdelkarim Kerbaa, and Oded Maler. Task graph scheduling using timed automata. *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, 2003.
2. Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows - Theory, Algorithms, and Applications*. Prentice Hall, 1993.
3. R. Alur and D. Dill. Automata for modelling real-time systems. In *Proc. of Int. Colloquium on Algorithms, Languages and Programming*, number 443, pages 322–335, July 1990.
4. Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. *Lecture Notes in Computer Science*, 2034:pp. 49–??, 2001.
5. J. E. Beasley, M. Krishnamoorthy, Y. M. Sharaiha, and D. Abramson. Scheduling aircraft landings - the static case. *Transportation Science*, 34(2):pp. 180–197, 2000.
6. Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Larsen, Paul Petterson, and Judi Romijn. Efficient guiding towards cost-optimality in UPPAAL. *Lecture Notes in Computer Science*, 2031:pp. 174+, 2001.
7. Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Larsen, Paul Pettersson, Judi Romijn, and Frits Vaandrager. Minimum-cost reachability for priced timed automata. *Lecture Notes in Computer Science*, 2034:pp. 147+, 2001.
8. Michel Berkelaar. `http://www.cs.sunysb.edu/~algorith/implement/lpsolve/implement.shtml`, Oct. 2003.
9. M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A model-checking tool for real-time systems. In A. J. Hu and M. Y. Vardi, editors, *Proc. 10th International Conference on Computer Aided Verification, Vancouver, Canada*, volume 1427, pages 546–550. Springer-Verlag, 1998.
10. W. H. Cunningham. A network simplex method. *Mathematical Programming*, 11:pp. 105–106, 1976.
11. George B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, New Jersey, 1963.
12. Flavius Gruian and Krzysztof Kuchcinski. Low-energy directed architecture selection and task scheduling. *Proceedings of the 25th EuroMICRO Conference*, 1:pp. 296–302, 1999.
13. Yu-Kwong Kwok and Ishfaq Ahmad. Benchmarking and comparison of the task graph scheduling algorithms. *Journal of Parallel and Distributed Computing*, 59(3):pp. 381–422, 1999.
14. K. Larsen, F. Larsson, P. Pettersson, and W. Yi. Efficient verification of real-time systems: Compact data structure and state space reduction. In *Proc. Real-Time Systems Symposium*, pages pp. 14–24, 1997.
15. Kim Larsen, Gerd Behrmann, Ed Brinksma, Ansgar Fehnker, Thomas Hune, Paul Pettersson, and Judi Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata. *Lecture Notes in Computer Science*, 2102:pp. 493+, 2001.

16. Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.
17. Andreas Löbel. `http://www.zib.de/Optimization/Software/Mcf/`, Oct. 2003.
18. Jacob Illum Rasmussen. Priced timed automata and duality - available at `http://www.cs.auc.dk/~illum/pubs/ptaduality.html`, 2003.
19. T. Tobita, M. Kouda, and H. Kasahara. Performance evaluation of minimum execution time multiprocessor scheduling algorithms using standard task graph set. *Proc. of PDPTA'00*, pages 745–751, 2000.