Clock Allocation in Timed Automata and Graph Colouring

Neda Saeedloei*
Southern Illinois University, Carbondale
neda@cs.siu.edu

Feliks Kluźniak
Logic Blox
feliks.kluzniak@logicblox.com

ABSTRACT

We consider the problem of optimal clock allocation for a fairly general class of timed automata, TA_S, under the safe assumption that all locations are reachable. Techniques similar to those used in compiler technology allow us to construct an interference graph: the problem of clock allocation for timed automata in TA_S can be reduced to that of colouring this graph. We then describe a class of timed automata, $TA_{DS} \subsetneq TA_S$, for which optimal clock allocation can be computed in polynomial time, because the corresponding interference graphs are perfect. Finally, we discuss some of the difficulties in applying similar techniques to timed automata outside TA_S.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability;

KEYWORDS

Timed Automata, optimal clock allocation, graph colouring

ACM Reference Format:
Neda Saeedloei and Feliks Kluźniak. 2018. Clock Allocation in Timed Automata and Graph Colouring. In *HSCC '18: 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week), April 11–13, 2018, Porto, Portugal.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3178126.3178138

1 INTRODUCTION

Minimizing the number of clocks in timed automata is important, as it affects the complexity of the verification problem [1, 2].

As is well known, it is in general undecidable whether the number of clocks in a given timed automaton $\mathcal{A}$ can be reduced while preserving the language accepted by $\mathcal{A}$ [7]. The most effective known attempt at tackling the problem is based on constructing a timed automaton that is bisimilar to $\mathcal{A}$ [10]. The constructed automaton may have more locations, but has the minimal number of clocks in the entire class of timed automata that are bisimilar to $\mathcal{A}$. The growth in the number of locations can be exponential in the number of clocks in $\mathcal{A}$. The computational complexity of that algorithm is 2-EXPTIME.

Earlier work on synthesizing timed automata from scenarios [16] described a method that produces automata with some interesting properties (the class of such automata is called TA_{DS}). These properties allowed us to formulate a clock allocation algorithm whose results are, to all practical purposes, optimal [17]. The cost of the algorithm was quadratic in the size of the automaton's graph, which was surprising, since in general the problem of clock allocation in timed automata is hard.

The current paper addresses the questions that naturally arise from that work: (1) Why is it that clock allocation in TA_{DS} can be solved so efficiently? (2) Can our method be generalised to timed automata outside TA_{DS}?

Specifically, the contributions of the paper are as follows:

(1) We identify a more general class of timed automata $TA_S \supsetneq TA_{DS}$ (see Sec. 3.2). For any timed automaton within TA_S, there is at most one clock reset on any transition.
(2) We use techniques known from compiler technology and perform a liveness analysis of clocks for automata in TA_S.
(3) We show the specifics of how the problem of clock allocation in timed automata within TA_S relates to the graph colouring problem. The results of liveness analysis can be used to construct an interference graph (which is usually much smaller than the graph of the automaton). A colouring of the interference graph with a minimum number of colours is equivalent to finding an optimal clock allocation for the original timed automaton, i.e., the optimal number of clocks is the chromatic number of the graph. So, we are able to minimize the number of clocks *without changing the graph or the language of the original automaton.*
(4) We prove that for every automaton in $TA_{DS} \subsetneq TA_S$ (see Sec. 4) the interference graph is chordal (therefore perfect). Hence it can be coloured in a time that is linear in the size of this graph [14].[1] In other words, clock allocation for timed automata within TA_{DS} can be solved in polynomial time. This result answers the question of why our specialized clock allocation algorithm for TA_{DS} [17] was at all possible.
(5) We show that for timed automata outside TA_S there are two clock-renaming operations that do not affect the accepted language, but can affect the shape of the interference graph. Each of these operations can either decrease or increase the chromatic number of the graph, and we do not know whether there exists a polynomial algorithm that can determine a renaming which leads to a graph with the smallest chromatic number. For a timed automaton outside TA_S, the chromatic number of the interference graph is only an upper bound on the optimal number of clocks.

*The work was done while the first author was at the University of Minnesota, Duluth.

HSCC '18, April 11–13, 2018, Porto, Portugal

ACM ISBN 978-1-4503-5642-8/18/04…$15.00
https://doi.org/10.1145/3178126.3178138

[1] The cost of constructing the interference graph is quadratic in the number of edges in the graph of the original automaton.

Throughout the paper we make a general simplifying assumption: when we formulate our criteria for determining the minimal number of clocks we do not take into account the satisfiability of clock constraints. We assume that it is possible to take each transition out of a state, and that a state reachable from the initial node of the automaton's graph is actually reachable in the automaton. The simplification does not affect the correctness of the resulting clock allocation, but only the meaning of optimality. By taking the semantics of constraints into account one might, in some cases, further decrease the number of clocks (see Sec. 3.1 for a more detailed discussion).

The remainder of the paper is structured as follows. After presenting an overview of timed automata in Sec. 2, we present our general assumptions under which we study the clock allocation problem for timed automata, in Sec. 3.1. Then we introduce the class TA_S (Sec. 3.2) and present an algorithm for liveness analysis of clocks for timed automata in this class (Sec. 3.3). This is followed by our formal definition of the clock allocation problem (Sec. 3.4) and the relation between clock allocation and graph colouring (Sec. 3.5). In Sec. 4 we describe the class TA_{DS}; Sec. 4.1 contains a proof that interference graphs for timed automata within TA_{DS} are chordal, therefore can be coloured efficiently. Sec. 5 shows that stepping outside the class TA_S makes the clock allocation problem significantly more difficult. In Sec. 6 we show the stability of the clock allocation problem for automata within TA_{DS}. This is followed by a few practical examples of timed automata in TA_{DS} in Sec. 7. We present a comparison of our work with related work in Sec. 8 and conclude in Sec. 9.

2 TIMED AUTOMATA

We now present a very brief overview of timed automata [2].

For a set C of clock variables, $\Phi(C)$ is the set of *clock constraints*, whose form is $c \sim a$, where $\sim \in \{\leq, \geq, <, >, =\}$, $c \in C$, and a is a constant in the set of rational numbers, $\mathbb{Q}$.

A *timed automaton* is a tuple $\mathcal{A} = \langle E, Q, Q_0, Q_f, C, R\rangle$, where

- E is a finite alphabet;
- Q is the (*finite*) set of locations;
- $Q_0 \subseteq Q$ is the set of initial locations;
- $Q_f \subseteq Q$ is the set of final locations;
- C is a finite set of *clock variables*;[2]
- $R \subseteq Q \times Q \times E \times 2^C \times 2^{\Phi(C)}$ is the set of transitions of the form $(q, q', e, \lambda, \phi)$, where $\lambda \subseteq C$ is the set of clocks to be reset with this transition, and ϕ is a set of clock constraints.

A time sequence $\tau = \tau_1\tau_2\ldots$ is an infinite sequence of (time) values $\tau_i \in \mathbb{R}^{\geq 0}$, satisfying two requirements:

- *Monotonicity*: τ increases strictly monotonically, i.e., $\tau_i < \tau_{i+1}$ for all $i \geq 1$.
- *Progress*: For every $t \in \mathbb{R}^{\geq 0}$, there is some $i \geq 1$ such that $\tau_i > t$.

A *timed word* over an alphabet E is a pair (σ, τ) where $\sigma = \sigma_1\sigma_2\ldots$ is an infinite word over E and τ is a time sequence.

A *clock interpretation* for a set C of clocks is a mapping from C to $\mathbb{R}^{\geq 0}_{\perp}$, where $\mathbb{R}^{\geq 0}_{\perp} = \mathbb{R}^{\geq 0} \cup \{\perp\}$. We say that a clock interpretation ν for C satisfies a set of clock constraints ϕ over C iff every clock constraint in ϕ evaluates to true after replacing each clock variable c with $\nu(c)$. All clock constraints that involve $\perp$, e.g., $\perp \leq 2$, evaluate to false.

For $\tau \in \mathbb{R}^{\geq 0}$, $\nu + \tau$ denotes the clock interpretation which maps every clock c to the value $\nu(c) + \tau$. We assume $\perp + \tau = \perp$.

For $Y \subseteq C$, $[Y \mapsto \tau]\nu$ denotes the clock interpretation for C which assigns τ to each $c \in Y$, and agrees with ν over the rest of the clocks.

A *run* ρ of a timed automaton over a timed word (σ, τ) is an infinite sequence of the form

$$\rho : \langle q_0, \nu_0\rangle \xrightarrow[\tau_1]{\sigma_1} \langle q_1, \nu_1\rangle \xrightarrow[\tau_2]{\sigma_2} \langle q_2, \nu_2\rangle \xrightarrow[\tau_3]{\sigma_3} \ldots$$

with $q_i \in Q$ and $\nu_i \in [C \to \mathbb{R}^{\geq 0}_{\perp}]$, for all $i \geq 0$, satisfying two requirements:

- $q_0 \in Q_0$, and $\nu_0(c) = \perp$ for all clocks $c \in C$;[3]
- for every $i \geq 1$ there is in R a transition $(q_{i-1}, q_i, \sigma_i, \lambda_i, \phi_i)$, such that $(\nu_{i-1} + \tau_i - \tau_{i-1})$ satisfies ϕ_i, and ν_i equals $[\lambda_i \mapsto 0](\nu_{i-1} + \tau_i - \tau_{i-1})$.

The set $inf(\rho)$ consists of $q \in Q$ such that $q = q_i$ for infinitely many $i \geq 0$ in the run ρ.

Different notions of acceptance have been proposed. For example, a run ρ of a timed *Büchi* automaton over a timed word (σ, τ) is an *accepting run* iff $inf(\rho) \cap Q_f \neq \emptyset$.

The language of $\mathcal{A}$, $L(\mathcal{A})$, is the set $\{(\sigma, \tau) \mid \mathcal{A}$ has an accepting run over $(\sigma, \tau)\}$.

3 FINDING AN OPTIMAL ALLOCATION OF CLOCKS

Our objective is to determine the minimum number of clocks required for a timed automaton $\mathcal{A}$, and to transform $\mathcal{A}$—*without changing its graph or its language*—to an automaton $\mathcal{A}'$ with an *optimal* (i.e., the smallest possible) number of clocks. (See Sec. 3.1 for a more precise formulation.)

Our clock allocation problem is reminiscent of the general register allocation problem [5, 15], but it is simpler: there is no limit on the number of clocks, and the formalism does not allow the value of a clock to be copied to another clock.

We will use some of the traditional terminology from the area of compiler construction, in particular the notions of *liveness* and *range*.

3.1 General assumptions

In the remainder of the paper we will assume that clocks are *well-defined*: if a clock occurs in a constraint on transition r, then the clock must be reset on every path from an initial location to r. (This can be checked by straightforward flow analysis. The cost is $O(|R|^2)$.)

Assume $\mathcal{A} = \langle E, Q, Q_0, Q_f, V, R\rangle$ is the original timed automaton, where $Q_f \neq \emptyset$.[4]

To keep the presentation simple and make a distinction between the clocks in the original and target timed automata, we assume

[2] We will follow the usual convention and use "clocks" instead of "clock variables".

[3] We adopt the convention of [3] (different from that in [2], where all clocks are assumed to be initially 0). This is in accordance with our assumption about well-defindness of clocks, mentioned at the begining of Sec. 3.1.

[4] One could argue that the optimal number of clocks of a timed automaton is zero when the accepted language is empty.

that the clocks in the original automaton belong to the set $V = \{t_0, t_1, t_2, ...\}$. The clock allocation found by our method will be a prescription for replacing each of these clocks with a clock that does not belong to V.

Let $N = \{j \mid t_j \sim a \in \phi$, where $(s, q, e, \lambda, \phi) \in R\}$. This is the set of *clock numbers*, i.e., of the indices of all the clocks that are referred to on transitions in R.

For a transition $r = (s, q, e, \lambda, \phi) \in R$, we use ϕ_r to denote the set of clock constraints on r.

Definition 1. *$clock\_ref : R \to 2^N$ maps transition r to the set $\{j \mid t_j \sim a \in \phi_r\}$. Intuitively, $clock\_ref(r)$ is the set of (indices of) clocks that are referred to in the constraints on r.*

Given a timed automaton $\mathcal{A}$, we abstract from the specific syntax of the constraints and consider only the identities of the clocks that are used in them. This concept is formalized next.

Definition 2. *Let $\mathcal{A} = \langle E, Q, Q_0, Q_f, V, R\rangle$. The syntactic abstraction of $\mathcal{A}$ is defined as $AbsSynt(\mathcal{A}) = \langle E, Q, Q_0, Q_f, V, R_c\rangle$, where $R_c = \{(s, q, e, \lambda, clock\_ref(r)) \mid r = (s, q, e, \lambda, \phi) \in R\}$.*

Let TA be a given set of timed automata. The function $AbsSynt$ induces a "natural" division of TA into equivalence classes[5] (of course, this has nothing to do with the equivalence of automata in the usual sense). We will use $C(\mathcal{A})$ to denote the equivalence class of $\mathcal{A} \in TA$, i.e., $C(\mathcal{A}) = \{a \in TA \mid AbsSynt(a) = AbsSynt(\mathcal{A})\}$.

Given a timed automaton $\mathcal{A} \in TA$, our method of clock allocation is carried out on $AbsSynt(\mathcal{A})$. It will be clear from the construction that the clock allocation is correct and optimal for the entire equivalence class of $\mathcal{A}$, in the following sense:

(1) *Correctness:* if we systematically replace the clocks (as prescribed by the computed allocation) in any timed automaton belonging to $C(\mathcal{A})$, then the resulting automaton will accept the same language as the original one.
(2) *Optimality:* any allocation with fewer clocks would be incorrect, i.e., there would exist at least one timed automaton $\mathcal{B} \in C(\mathcal{A})$, for which it would change the accepted language.

The reason we perform this abstraction is that in our analysis we do not want to concern ourselves with "pathological" cases in which the required number of clocks is strongly affected by some particular form of the constraints. We give three examples of such cases:

- An automaton with a clock c, such that all the clock constraints on c are of the form $c \geq 0$, which is always true. Clearly, in an optimal clock allocation clock c can be safely removed.
- An automaton that contains transitions whose clock constraints are always false, e.g., a constraint of the form $c > 2 \wedge c < 2$. Such a transition can be safely removed from the automaton, which might lead to further reductions.
- An automaton similar to that of Fig. 1, which is timed bisimilar (therefore, timed-language equivalent) to the automaton of Fig. 2 (assuming both l_1 and l_2 are accepting locations). Obviously, the optimal number of clocks for this automaton is zero.

[5]Let $f : X \to Y$ be a total function. For any $D \subseteq X$, $\{(a, b) \in D^2 \mid f(a) = f(b)\}$ is an equivalence relation.

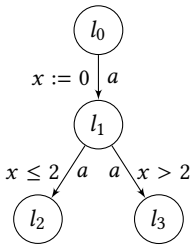

Figure 1: A timed automaton

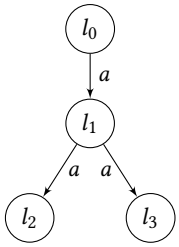

Figure 2: An automaton timed bisimilar to the one in Fig. 1

Opportunities for optimisation arising out of the fact that the constraints have some particular form should of course be pursued whenever it is practical to do so. Such optimisations are, however, outside the scope of the current paper. It would be best to perform them *before* clock allocation.

3.2 The class TA_S

Definition 3. *A timed automaton belongs to the class TA_S if and only if for any transition $r = (s, q, e, \lambda, \phi)$, $|\lambda| \leq 1$ (i.e., r has at most one clock reset).*

In the remaining of Sec. 3 we limit our attention to timed automata in TA_S.[6] Sec. 5 briefly explores the problems associated with stepping outside this class.

3.3 Liveness analysis of clocks

Let r be a transition in a timed automaton, and let, say, $t_0 < 2$ and $t_1 = 1$ be clock constraints on r. We say that the clocks t_0 and t_1 are *in conflict on r*, i.e., they cannot, in general, both be replaced by the same clock. The purpose of this subsection is to pin down this intuition.

We begin with liveness analysis, which is performed by Algorithm 1. The algorithm determines, in effect, the "ranges" of all the clocks in the original timed automaton. Each such range is a set of transitions. Informally, a range for a particular reset of clock t_j captures the notion "this value of t_j will be needed only on these transitions".

Before presenting the algorithm, we introduce some auxiliary definitions.

[6]We focus on traditional timed automata and do not consider, for example, the notions of committed and urgent locations, introduced in UPPAAL [4]. If committed locations are allowed, then it is possible to replace any timed automaton with an equivalent one in TA_S.

For a transition $r = (s, q, e, \lambda, \phi) \in R$, we define $source(r) = s$. We will use λ_r and ϕ_r to denote the sets of clocks to be reset on r and the clock constraints on r, respectively.

For a location $s \in Q$, we use $out(s)$ to denote the set of transitions that originate in s, i.e., for which s is the source.

Let $p = r_1 ... r_k$ be a path in the graph of automaton $\mathcal{A}$. We define $transitions(p)$ to be the set $\{r_1, ..., r_k\}$.

We also define the following functions:

- $born : R \to 2^N$ maps transition r to the set $\{j \mid t_j \in \lambda_r$ and there exists a path $rr_1 ... r_k$, $k \geq 1$, such that $j \in clock\_ref(r_k)$ and $t_j \notin \lambda_{r_i}$ for $1 \leq i < k\}$. Intuitively, $born$(r) identifies a clock that is reset on r whose value can be used on some transition reachable from r.
- $origins : N \to 2^R$ maps clock number j to the set $\{r \mid j \in born(r)\}$. Intuitively, $origins(j)$ is the set of transitions at which clock t_j is born, i.e., reset and later used.
- $reachable\_from : R \to 2^R$ maps transition r to the set of transitions from which it can be reached by some non-empty path.
- $active : R \to 2^N$ maps transition r to the set $\{j \mid$ there is a path $rr_1 ... r_k$, $k \geq 1$, such that $j \in clock\_ref(r_k)$ and $t_j \notin \lambda_{r_i}$ for $1 \leq i < k\}$. Intuitively, $active(r)$ identifies clocks that are "alive" on r (i.e., their values may be subsequently used). Notice that $born(r) \subseteq active(r)$.
- $last\_ref : R \to 2^N$ maps transition r to $clock\_ref(r) \setminus active(r)$.
- $needed : R \to 2^N$ maps transition r to $active(r) \cup last\_ref(r)$.

The graph may contain cycles, so r_k need not be different from r in the definitions of *born* and *active*. Note that a constraint on r cannot refer to a reset on r if r is not a part of a cycle: the reset takes place after the constraints have been used in determining whether the transition can be taken.

We should explain the distinction between *active* and *needed*. Let $j \in last\_ref(r)$, i.e., the value of clock t_j will not be used in any transition after r, unless the clock is first reset. Let c be the (new) clock that will replace the old clock t_j in the target automaton. c is needed on r because it occurs there in a constraint, but once the constraint has been used to determine whether the transition can be chosen, c can be immediately reassigned, i.e., reset on r and used to replace another (old) clock. Had j been present in $active(r)$, c could not be reassigned on r.

Note that in this example $j \in active(r')$, for every r' that is a direct predecessor of r, because t_j must be reset on every path to r.

In Fig. 3, 0 is in both $active(r_2)$ and $clock\_ref(r_2)$; 0 is also in $clock\_ref(r_4)$, but not in $active(r_4)$, so $0 \in last\_ref(r_4)$.

We say that clock t_j *is needed* (or *is active*) on r if $j \in needed(r)$ (or $j \in active(r)$).

Definition 4. *A path $p = r_0 ... r_n$ is a* path for clock t_j *iff $j \in born(r_0)$ and $j \in needed(r_i)$ for $0 \leq i \leq n$.*

Definition 5. *$range : N \times R \to 2^R$ maps (j, r) to $\{r' \mid r' \in transitions(p)$, where p is a path for clock t_j that starts at $r\}$. Intuitively, $range(j, r)$, where $r \in origins(j)$, is the set of all transitions on which the value from the reset of t_j on r must be available.*

Algorithm 1: Building the ranges for clocks

Input : A timed automaton $\mathcal{A} = \langle E, Q, Q_0, Q_f, V, R \rangle$.
Output: The *range* function for $\mathcal{A}$ and auxiliary functions.

```
foreach j ∈ N do
    origins(j) := ∅;
foreach transition r = (s, q, e, λ, φ) ∈ R do
    born(r) := ∅;
    active(r) := ∅;
    reachable_from(r) := ∅;
    foreach t_j ∈ λ do
        range(j, r) := ∅;
repeat
    foreach transition r = (s, q, e, λ, φ) ∈ R do
        foreach r_o ∈ out(q) do
            active(r) := active(r) ∪
              (active(r_o) \ born(r_o)) ∪ clock_ref(r_o);
            reachable_from(r_o) :=
              reachable_from(r_o) ∪ reachable_from(r) ∪ {r};
        foreach t_j ∈ λ do
            if j ∈ active(r) then
                born(r) := born(r) ∪ {j};
                origins(j) := origins(j) ∪ {r};
                range(j, r) := range(j, r) ∪ {r};
until there were no changes;
foreach r ∈ R do
    foreach j ∈ active(r) ∪ clock_ref(r) do
        foreach r' ∈ origins(j) do
            if r' ∈ reachable_from(r) then
                range(j, r') := range(j, r') ∪ {r};
```

Given a timed automaton $\mathcal{A}$, Algorithm 1 constructs all the ranges for all the clocks in $\mathcal{A}$. It is a standard flow-analysis algorithm whose complexity is quadratic in the number of edges.[7]

In the automaton of Fig. 3 the ranges for clock t_1 obtained by Algorithm 1 are $range(1, r_2) = \{r_2, r_4, r_6\}$ and $range(1, r_3) = \{r_3, r_5, r_6\}$.

Definition 6. *$Ranges : N \to 2^{2^R}$ maps j to $\{range(j, r) \mid r \in origins(j)\}$. Intuitively, $Ranges(j)$ is the set of all the ranges for clock t_j.*

Definition 7. *Given a clock t_j, we define $rel_j = \{(a, b) \in Ranges(j) \times Ranges(j) \mid a \cap b \neq \emptyset\}$.*

The relation rel_j is reflexive and symmetric; rel_j^* is its transitive closure.

[7] Since the number of transitions, locations and original clocks is known, all sets can be represented by bit vectors. So in practice set operations can be considered to take constant time. The constant does grow with the size of the problem (albeit 64 times less quickly), so strictly speaking the complexity is cubic rather than quadratic.

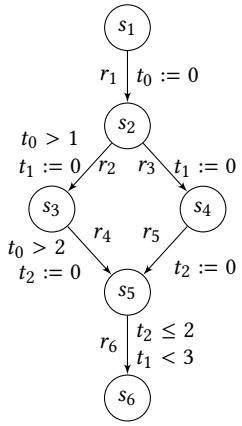

Figure 3: A timed automaton (in TA_S)

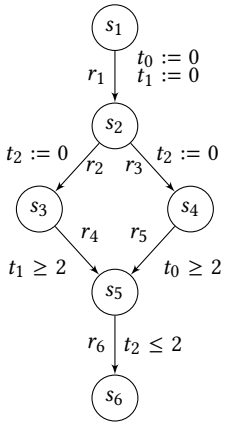

Figure 4: A timed automaton (not in TA_S)

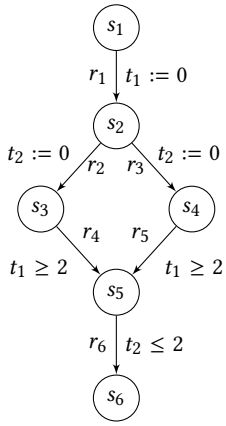

Figure 5: An automaton equivalent to that in Fig. 4

Definition 8. *Let $j \in N$ and $a \in Ranges(j)$. We define* $Rel(j,a) = \{b \in Ranges(j) \mid a\ rel_j^*\ b\}$.

One can think of $Rel(j,a)$ as the set of ranges to which range a for clock t_j is directly or indirectly "related". The ranges are related, because they share some transitions, so on all of these ranges t_j must be represented by the same clock. Naturally, $b \in Rel(j,a)$ implies $Rel(j,a) = Rel(j,b)$.

Definition 9. *Let $j \in N$ and $a \in Ranges(j)$. The set* $f_j = \{r \in \bigcup_{S \in Rel(j,a)} S \mid j \in active(r)\}$ *is a* family *for t_j.*

We say that family f_j *originates* at location q, if there is a transition $r \in out(q)$ such that $r \in f_j \cap origins(j)$. Note that a family may originate at more than one location.

In Fig. 3 $range(1, r_2)$ and $range(1, r_3)$ overlap on r_6, so there is one family for t_1, viz. $\{r_2, r_3, r_4, r_5\}$. The other families are $\{r_1, r_2\}$ for t_0 and $\{r_4, r_5\}$ for t_2.

There may, of course, be a number of different families for the same clock. Two families for the same clock must be disjoint (otherwise they would be the same family).

We use $\mathcal{F}_\mathcal{A}$ to denote the set of all families in $\mathcal{A}$ (for all clocks).

Definition 10. *We define conflict $\subset \mathcal{F}_\mathcal{A} \times \mathcal{F}_\mathcal{A}$ to be the set* $\{(f,g) \mid f \cap g \neq \emptyset \wedge f \neq g\}$.

We say that two families f and g in $\mathcal{F}_\mathcal{A}$ conflict, if f *conflict* g. If $r \in f \cap g$, then we say f and g *conflict on* r. If f is a family for t_j and g is a family for t_k, then we also say that the clocks t_j and t_k conflict on r. Notice that a clock cannot conflict with itself on any transition.

From the discussion of *active* vs. *needed* (p. 4) we see that:

- If $i \neq j$, $j \in born(r)$ and $i \in last\_ref(r)$, then t_j and t_i do not conflict on r.
- If there is a transition r such that, for some $j \neq k$, $j \in last\_ref(r) \wedge k \in last\_ref(r)$, then there must be families f for t_j and g for t_k such that $f \cap g \neq \emptyset$ (i.e., t_j and t_k will conflict on some other transition). This is a direct consequence of the assumption that the clocks are well-defined.

This observation explains why it is sufficient to limit a family only to those transitions on which the relevant clock is active.

3.4 Clock allocation

After the families are generated, we can use this information to allocate new clocks. The general idea is that, once a clock is reset on one of the initial transitions of a family for the clock, it should never be reset again within the transitions of that family; however, the clock can be reused outside the family.

We will now formally define clock allocations and their interesting properties.

Let $\mathcal{A}$ be a timed automaton with the set $\mathcal{F}_\mathcal{A}$ of families. We assume the existence of a set C of clock variables, disjoint from V.

Definition 11. *A* clock allocation *for $\mathcal{A}$ is a relation alloc $\subseteq \mathcal{F}_\mathcal{A} \times C$.*

Inclusion of (f,c) in *alloc*, where f is a family for clock t_j, represents the intention to replace each occurrence of t_j with c on all transitions of f. In particular, c will be reset on all those transitions in $origins(j)$ that belong to f.

Given an allocation *alloc*, we will say that c is *associated with* f, or *allocated to* f, if $(f,c) \in alloc$.

Definition 12. *The clock allocation alloc for $\mathcal{A}$ is* complete *iff, for every family $f \in \mathcal{F}_\mathcal{A}$, there is a clock $c \in C$ such that $(f,c) \in alloc$.*

Definition 13. *A clock allocation alloc for $\mathcal{A}$ is* incorrect *iff there exist two conflicting families f and g in $\mathcal{F}_\mathcal{A}$, such that $(f,c) \in alloc \wedge (g,c) \in alloc$, for some $c \in C$. alloc is* correct *iff it is not incorrect.*

In the automaton of Fig. 3, associating the same clock c with the families for t_1 and t_2 would create an incorrect clock allocation, because the two families conflict. It would, however, be correct to associate the same clock with the families for t_0 and t_2, since they do not conflict.

Definition 14. *We define the number of clocks used in allocation alloc by:*

$cost(alloc) = |\{c \in C \mid \exists_{f \in \mathcal{F}_\mathcal{A}} (f, c) \in alloc\}|$.

Definition 15. *Let alloc be a complete correct clock allocation for $\mathcal{A}$. alloc is* optimal *if there is no complete correct allocation alloc′ for $\mathcal{A}$ such that $cost(alloc') < cost(alloc)$.*

Definition 16. *Two families, f and g in $\mathcal{F}_\mathcal{A}$, belong to the same* tribe*, if f conflict* g, where conflict* is the reflexive transitive closure of conflict.*

All the three families of the automaton in Fig. 3 belong to the same tribe.

Observe that members of different tribes do not share transitions.

Definition 17. *Two families, f and g in $\mathcal{F}_\mathcal{A}$, belong to the same* cluster *iff $f \cap g \neq \emptyset$. A cluster F is a maximal set of such families, i.e., every family outside F is disjoint from at least one of the families in F.*

The members of a cluster must be families for different clocks, since two families for the same clock cannot overlap (otherwise they would be one family). A family can belong to more than one cluster, but only to one tribe. All the members of a cluster belong to the same tribe.

Since each pair of families in a cluster shares some common transition, we immediately see the following:

Observation 1. *A complete and correct allocation must associate every family in a cluster with a different clock.*

Observation 2. *Let alloc be a complete correct allocation for $\mathcal{A}$. Then cost(alloc) cannot be smaller than the size of the largest cluster in $\mathcal{F}_\mathcal{A}$.*

In the automaton of Fig. 3 there are two clusters, each of size two: $\{\{r_1, r_2\}, \{r_2, r_3, r_4, r_5\}\}$ (i.e., families for t_0 and t_1) and $\{\{r_4, r_5\}, \{r_2, r_3, r_4, r_5\}\}$ (i.e., families for t_2 and t_1). Obviously, a correct complete allocation for the automaton requires at least two clocks.

3.5 Clock allocation and graph colouring

Given a timed automaton $\mathcal{A} \in TA_S$, with its set of families $\mathcal{F}_\mathcal{A} = \{f_1, f_2, \ldots, f_n\}$, we construct its *interference graph*, $\mathcal{I}_\mathcal{A}$.

$\mathcal{I}_\mathcal{A}$ is an undirected graph, constructed as follows. For every family f_i in $\mathcal{F}_\mathcal{A}$, we create a node v_{f_i} in $\mathcal{I}_\mathcal{A}$. For every two different families f_i and f_j in $\mathcal{F}_\mathcal{A}$, we create an edge between v_{f_i} and v_{f_j} if and only if f_i and f_j conflict.

It is now clear that the problem of finding an optimal clock allocation in $\mathcal{A}$ is equivalent to the problem of colouring the nodes of $\mathcal{I}_\mathcal{A}$ with a minimum number of colours. Two nodes cannot be coloured with the same colour if they are connected by an edge; two families cannot be allocated the same clock if they conflict. It is enough to establish a one-to-one correspondence between colours and clocks.

Assume f_j, a family for clock t_j, is allocated a new clock c. Then, in the final timed automaton, all occurences of t_j are replaced by c on all transitions of f_j on which j is needed. The resulting timed automaton is clearly timed bisimilar to the original automaton.

A cluster in $\mathcal{F}_\mathcal{A}$ corresponds to a clique in $\mathcal{I}_\mathcal{A}$: the size of a maximal cluster is equal to the size of a maximal clique, and it is a lower bound[8] on the minimum number of clocks and on the chromatic number, which are equal.

It is worth noting that in TA_S the number of families cannot exceed the number of transitions, which puts a limit on the size of the interference graph. In our experience the interference graphs tend to be quite small in comparison with the graphs of the original timed automata.

If there is more than one tribe, then the interference graph is disconnected. In this case the number of clocks required for a timed automaton $\mathcal{A} \in TA_S$ is the maximum over the chromatic numbers of the disconnected subgraphs of $\mathcal{I}_\mathcal{A}$.

4 A CLASS OF TIMED AUTOMATA: TA_{DS}

In this section we introduce a class of timed automata, $TA_{DS} \subsetneq TA_S$, which has an important property: for timed automata that belong to this class, the interference graphs are *perfect* (see Sec. 4.1). The graph colouring and maximum clique problems for perfect graphs can be solved in polynomial time [9].

The class TA_{DS} was first encountered in the context of formal model synthesis of timed systems [16]. The class is interesting primarily because it is amenable to efficient clock allocation, but it is not without some practical value. Although the issue is tangential to the main topic of this paper, we discuss it briefly in Sec. 7.

Intuitively, for a timed automaton in this class, we want to be sure that a clock in every constraint is not only well-defined, but also that its value measures the amount of time that has elapsed since leaving a particular location. Checking that a given timed automaton belongs to TA_{DS} can be peformed easily by a polynomial time algorithm.

We now characterize TA_{DS}.

Let $\mathcal{A}$ be a timed automaton with a set of locations Q and a unique initial location. If s and q are locations in Q, then s *dominates* q if and only if all paths from the initial location to q pass through s [12]. We denote the *dominance relation* on Q by $\succeq$: $s \succeq q$ iff s dominates q (we also say that q is dominated by s). Note that $\succeq$ is a partial order on Q. We write $s \succ q$ to denote that $s \succeq q$ and $s \neq q$.

We extend the definition of dominated locations to *dominated transitions*: a transition r is *dominated* by location s iff $s \succeq source(r)$.

We assume that the definition of a timed automaton is extended with an injective partial labelling function: $L : Q \to N_L$, where $N_L \subset \{0, 1, 2, \ldots\}$ is the set of labels (values of L) used for $\mathcal{A}$.

Definition 18. *A timed automaton belongs to the class TA_{DS} if and only if it satisfies the following three restrictions:*

1. *It has a unique initial location, q^0. Every location can be reached from q^0.*
2. *Clock t_j is reset only on transitions in $out(s)$, where $L(s) = j$; moreover, t_j is reset on all the transitions in $out(s)$.*[9]
3. *A clock constraint on a transition r can contain an occurrence of t_j* only *if $j \in N_L$ and $L^{-1}(j) \succ source(r)$.*

Restriction 3, which we call *the dominance assumption*, means that if $t_j \sim a$ is a clock constraint on a transition $r \in out(s)$, then the

[8] For instance, in any odd cycle graph with 5 or more vertices the maximal clique number is 2 and the chromatic number is 3.

[9] If a transition leads only to paths on which t_j is not used, the reset (and the clock that is reset) will be eliminated by our algorithms.

value of t_j represents the amount of time that has elapsed since leaving a location q, where $q \succ s$ and $L(q) = j$.

The automaton of Fig. 5 belongs to TA_{DS}, while the one in Fig. 3 does not.

Observe that for any automaton in TA_{DS} there is at most one reset per transition, therefore $TA_{DS} \subsetneq TA_S$.

4.1 The interference graphs for timed automata in TA_{DS}

In this section we show that, for any $\mathcal{A} \in TA_{DS}$, the interference graph $\mathcal{I}_\mathcal{A}$ is *chordal*, therefore perfect. In a chordal graph all cycles of four or more vertices have a chord: an edge that is not part of the cycle but connects two vertices of the cycle.

As is well known, the clique number of a perfect graph is exactly equal to its chromatic number [9]. Moreover, for a chordal graph the colouring problem can be solved by an algorithm that is linear in the size of the graph [14]. So the consequence of the theorem proven in this section is that the problem of clock allocation for timed automata in TA_{DS} can be solved in polynomial time.

Given $\mathcal{A} \in TA_{DS}$ with its set of families $\mathcal{F}_\mathcal{A}$, its interference graph $\mathcal{I}_\mathcal{A}$, and families f and g in $\mathcal{F}_\mathcal{A}$, we use $f - g$ to denote that there is an edge between v_f and v_g in $\mathcal{I}_\mathcal{A}$, i.e., families f and g conflict. The relation $-$ is irreflexive and symmetric. For a family f, s_f is the (unique) location in $\mathcal{A}$ at which f originates.

Observation 3. *Let $s_1 \succeq s_3$, $s_2 \succeq s_3$, and $s_1 \neq s_2$. Then either $s_1 \succ s_2$ or $s_2 \succ s_1$.*

Proof. This is a direct consequence of the definition of dominance. □

Observation 4. *For any transition r in a family f, $s_f \succeq source(r)$.*

Proof. This is a direct consequence of our assumption about where a clock can be reset, the dominance assumption and the definition of families. □

Observation 5. *Let $f - g$. Then $s_f \neq s_g$.*

Proof. If $s_f = s_g$, then f and g are two families for the same clock (because of our assumption about where a clock can be reset). But then $f \cap g = \emptyset$. □

Observation 6. *Let $f - g$. Then there exists a transition r such that $r \in f$, $r \in g$, $s_f \succeq source(r)$ and $s_g \succeq source(r)$.*

Proof. $f \cap g \neq \emptyset$. Let $r \in f \cap g$ and use Observation 4. □

Observation 7. *Let $f - g$. Then $s_f \succ s_g$ or $s_g \succ s_f$.*

Proof. Observations 6, 5 and 3. □

Observation 8. *Let r be a transition in a family f. Then the transitions on all paths from s_f to r belong to f.*

Proof. The dominance assumption, our definitions of "active" and "family". □

Observation 9. *Let $f - g$, $g - h$ and $s_f \succ s_h \succ s_g$. Then $f - h$.*

Proof. There must be a path p from s_f to some transition in g such that all transitions of p belong to f. Since $s_f \succ s_h \succ s_g$, p must pass through s_h, then through s_g. Let p_{hg} be that subpath of p that begins at s_h and ends at s_g. Since $s_h \neq s_g$, p_{hg} is not empty. But $g - h$, so by Observations 6 and 8 the transitions of p_{hg} must belong to h. □

Observation 10. *Let $f - g$, $g - h$ and $s_h \succ s_f \succ s_g$. Then $f - h$.*

Proof. Observation 9 and the symmetry of $-$. □

Observation 11. *Let f and f' be two families for the same clock (both originating at location s_f). If $f - g$ and $g - f'$, then $s_g \succ s_f$.*

Proof. By Observation 7, $s_f \succ s_g$ or $s_g \succ s_f$. Assume $s_f \succ s_g$. But then by Observations 6 and 8 both f and f' would overlap on the paths from s_f to s_g, so they would be the same family. □

We informally say that $\mathcal{I}_\mathcal{A}$ includes a "valley", if there are families f, g and h, such that $f - g$, $g - h$, $s_f \succ s_g$ and $s_h \succ s_g$.

Lemma 1. *Let $f - g$, $g - h$, $s_f \succ s_g$ and $s_h \succ s_g$. Then $f - h$. ("A valley must be bridged.")*

Proof. By Observation 11, $s_f \neq s_h$. Then, by Observation 3, $s_f \succ s_h$ or $s_h \succ s_f$. So we must have either $s_f \succ s_h \succ s_g$ or $s_h \succ s_f \succ s_g$. Therefore $f - h$, by Observations 9 and 10. □

Theorem 1. *$\mathcal{I}_\mathcal{A}$ is chordal.*

Proof. Assume $f_0 - f_1 - f_2 - \ldots - f_{n-1} - f_0$, for $n \geq 4$, is a circuit in $\mathcal{I}_\mathcal{A}$, i.e., there is no edge between any two non-consecutive nodes.

Consider the sequence $s_{f_0}, s_{f_1}, \ldots, s_{f_{n-1}}, s_{f_0}, s_{f_1}$. By Observation 7, neighbouring elements of this sequence are comparable and not equal. But the sequence cannot monotonically increase or decrease. So there is a valley, and the valley must be bridged.

Formally, there must be a contiguous subsequence x, y, z such that $x \succ y$ and $z \succ y$. Let i be such that $x = s_{f_i}$, $y = s_{f_{(i+1) mod\ n}}$ and $z = s_{f_{(i+2) mod\ n}}$. Then, by Lemma 1, $f_i - f_{(i+2) mod\ n}$. □

5 SPLITTING AND CONFLATION OF CLOCKS

In this section we explain why we limited our attention to TA_S while discussing clock allocation, its optimality and its relation to graph colouring. We show how allowing more than one clock reset on a transition can affect the optimality of an allocation and make the problem computationally expensive.

The shape of the interference graph for an automaton depends on the ranges and families of the clocks in the automaton. However, the ranges and families can be potentially affected by two operations: *splitting of clocks* and *clock conflation*.

Consider the automaton of Fig. 4, which does not belong to TA_S. In this automaton there are three families, each of which conflicts with the other two. So the chromatic number of the interference graph, shown in Fig. 7(A), is three. However, it is possible to represent both t_0 and t_1 with one clock, say t_1, that is reset on r_1 (see Fig. 5). The chromatic number of the interference graph of this newly obtained automaton, shown in Fig. 7(B), is two: indeed, two clocks are sufficient.

In general, two clocks t_i and t_j can be so *conflated* if they are reset on exactly the same transitions. However, conflating any two such clocks does not always result in a decrease of the chromatic number. For instance, conflating t_0 and t_1 in the automaton of Fig. 6 will increase the chromatic number of the interference graph from two to three: the original interference graph is a square, similar to that of Fig. 7(C), after replacing t_0 and t_1 by one clock, the interference graph changes to a triangle, similar to that of Fig. 7(A).

In the timed automaton of Fig. 8 the interference graph is exactly like the graph of Fig. 7(A), so the chromatic number is three. However, since the family for clock t_0 forks into two disjoint paths at location s_2, clock t_0 can be split (thus moving the automaton outside TA_S): two clocks t_0' and t_0'' will be introduced and reset on r_1, clock t_0 will be replaced by t_0' on r_4 and by t_0'' on r_5. In the new interference graph, shown in Fig. 7(C), the chromatic number decreases to two, and so does the number of required clocks.

In general, a clock t_j can be *split* into two clocks if its range forks into (at least) two disjoint paths. Then t_j can be replaced by t_j' on one branch of the fork and by t_j'' on the other branch of the fork. Just as conflation, clock splitting does not always decrease the chromatic number. For instance, splitting t_1 in Fig. 5 will increase the chromatic number: the interference graph (see Fig. 7(B)) changes to a triangle.

Splitting and conflation of clocks, which can be applied before liveness analysis, are *clock renaming* (or simply *renaming*) *operations*.

As is shown by these examples, it is, in general, possible to use renaming operations in order to transform a timed automaton $\mathcal{A}$ to a provably equivalent automaton $\mathcal{B}$ whose interference graph has a different chromatic number than that of $\mathcal{A}$. We are not aware of a polynomial algorithm that would determine the sequence of renaming operations that minimises the chromatic number of the associated interference graph: we conjecture that in the general case this could require a complete search.

So, for automata outside TA_S we can make only the following observation (while preserving the assumptions in Sec. 3.1):

> *For a timed automaton $\mathcal{A}$ outside TA_S, the chromatic number of $\mathcal{I}_\mathcal{A}$ is an upper bound on the number of clocks required for $\mathcal{A}$.*

As already mentioned (see Fig. 8), clock splitting may be possible for an automaton within TA_S. This may result in a smaller chromatic number, but would also bring us outside TA_S. In Sec. 6, we show that splitting a clock in an automaton within TA_{DS} cannot result in such an improvement.

It is worth noting that allowing clock conflations would invalidate our criterion for the correctness of an allocation. If f_i and f_j are conflicting families for two clocks t_i and t_j that can be conflated, then it is possible to allocate the same clock to both f_i and f_j (see Definition 13).

6 THE STABILITY OF THE CLOCK ALLOCATION PROBLEM IN TA_{DS}

Theorem 2. *For a timed automaton $\mathcal{A} \in TA_{DS}$, the chromatic number of $\mathcal{I}_\mathcal{A}$ cannot be decreased by renaming operations.*

Proof. Conflatiion of clocks is not possible within TA_{DS}, as there is at most one reset per transition. We show that splitting clocks cannot decrease the chromatic number of $\mathcal{I}_\mathcal{A}$.

Assume F is the largest cluster in $\mathcal{F}_\mathcal{A}$. Suppose f_j is a family for some clock t_j, such that $f_j \in F$. Suppose further that t_j is split into two clocks t_j' and t_j''. Let f_j' and f_j'' be the two new families that are formed as a result of this split. Observe that f_j' and f_j'' must share an initial path, and then perhaps branch away from each other. Assume that by splitting t_j, the size of cluster F decreases. This can only happen if there are two other families, $g \in F$ and $h \in F$, such that: (1) g and h both conflicted with f_j, (2) g now conflicts with f_j', but not with f_j'', and (3) h now conflicts with f_j'' but not with f_j'. But if so, then there must exist a location x at which f_j' branches away from f_j'', such that $x \succ s_g$ and $x \succ s_h$. Therefore g cannot conflict with h, which contradicts the assumption that g and h are families in F. □

7 A FEW PRACTICAL EXAMPLES OF TIMED AUTOMATA IN TA_{DS}

The pivotal role of the dominance assumption makes the class TA_{DS} somewhat restricted, but it can still be used to model interesting properties of real-time systems, e.g., safety properties. In particular, *bounded-response* and *bounded-invariance*, which are both safety properties under the assumption that time progresses [11], can be easily expressed in TA_{DS}. Bounded-response properties assert that "something good" will happen within a specified amount of time. Bounded-invariance properties assert that "nothing bad" will happen for a certain amount of time.

Safety properties can be reduced to reachability by using *monitor* automata [13]. For instance, Fig. 9 shows a variant of a monitor automaton [13], which checks the property that "a is always followed by b within at most 10 time units". The idea is to check whether the "bad" state, in which the desired safety property is violated, is reachable. Observe that the label $any - but - a$ on the loop in state s_0 matches any input symbol except a. The label on the loop in state s_1 can be interpreted similarly.

Fig. 10 shows an automaton that checks a bounded-invariance property that "b will never occur within 10 time units after a's occurrence". (Though very similar to the automaton in Fig. 9, it expresses something quite different.)

Periodic behaviors can also be specified quite easily within TA_{DS}. For instance, the automaton shown at the top of Fig. 11 [2] can be simulated by an automaton in TA_{DS} (shown at the bottom of Fig. 11), by introducing a new location and a silent transition.

All these automata clearly belong to TA_{DS}. A more sophisticated example is a model of an Automatic Teller Machine (ATM) [16].

8 COMPARISON WITH RELATED WORK

The problem of reducing the number of clocks in timed automata has been addressed by constructing bisimilar timed automata [6, 10]. (Both these papers make use of standard liveness analysis, but in a way that is quite different from ours.)

The approach used by Daws and Yovine [6] combines two methods for reducing the number of clocks. The first method is based on identifying the set of clocks active in each location, and applying

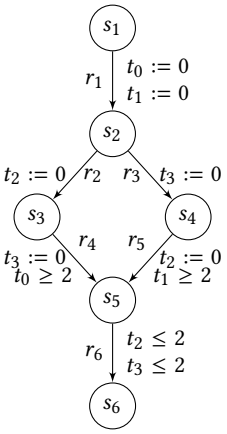

Figure 6: An automaton outside TA_S: clock conflation would be harmful

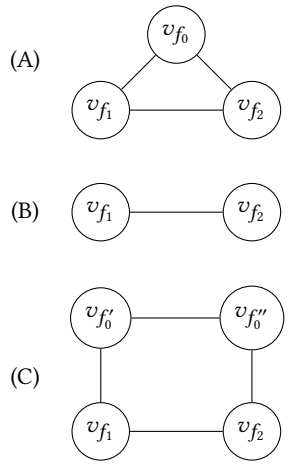

Figure 7: Two interference graphs

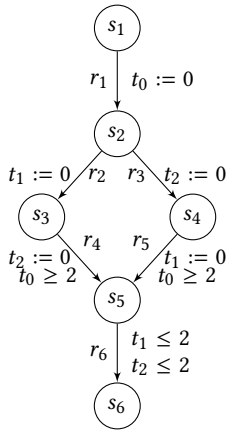

Figure 8: An automaton inside TA_S: clock splitting would be beneficial

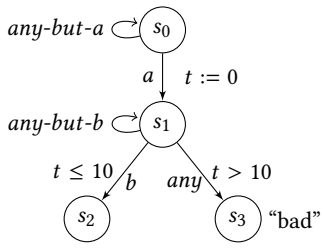

Figure 9: bounded response

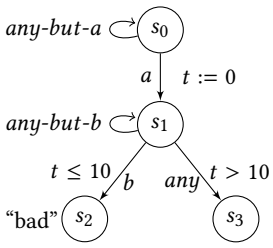

Figure 10: bounded invariance

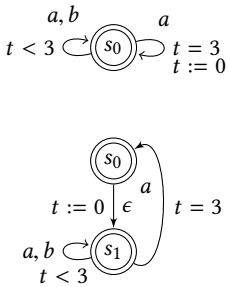

Figure 11: Periodic behavior

a clock renaming to this set locally, at each location, to obtain a bisimilar timed automaton. The second method is based on the notion of equality between clocks: if two clocks are equal in a location, one is deleted. The authors allow a clock to be assigned to another clock, which is an extension of the traditional formalism of timed automata. Their method will not always yield the optimal number of clocks, as argued by Guha et al. [10].

Guha et al. [10] propose a method that, given a timed automaton $\mathcal{A}$, constructs another automaton that has the minimal number of clocks in the entire class of timed automata that are bisimilar to $\mathcal{A}$. Their method uses zone graphs for identifying redundant transitions and implied constraints that can be eliminated. It also uses a technique of "splitting locations" for reducing the number of clocks. As a result of this the number of locations in the constructed automaton may become exponential in the number of clocks in $\mathcal{A}$. The algorithms are rather complicated, and the computational complexity of the method is 2-EXPTIME.

To the best of our knowledge, our work is the first attempt to detail how the problem of clock allocation in timed automata is directly related to the graph colouring problem. The last of the 4 stages in Guha et al. [10] uses graph colouring of a "clock graph". The construction of clock graph in [10] uses the semantics and a zone graph: this graph is different from our interference graph.

For a given automaton in TA_S that satisfies the assumptions in Sec. 3.1, our clock allocation method computes the minimum number of clocks that are required[10], and optimally allocates clocks *without changing the shape of the graph or the language of the automaton*. For an automaton outside TA_S our method computes only an upper bound on the required number of clocks.

We have identified a class of timed automata, $TA_{DS} \subsetneq TA_S$, for which the problem can be solved in polynomial time.

The cost of our liveness analysis, performed by Algorithm 1, is quadratic in the number of edges. For TA_{DS}, the interference graph is chordal, so colouring can be performed in $O(|V|+|E|)$ time [8, 14].

9 CONCLUSIONS

We propose a novel approach to the problem of optimal clock allocation in timed automata. Our method is based on liveness analysis of clocks and utilizes well-studied results in graph theory, in particular the graph colouring and maximum clique problems.

We show how the problem of clock allocation in timed automata relates to the graph colouring problem, and how an upper bound on the required number of clocks in a timed automaton can be determined by computing the chromatic number of its interference

[10] In the sense explained in Sec. 3.1.

graph. If we limit our attention to class TA_S, in which there is at most one reset per transition, then this upper bound is tight.

We identify two operations, clock splitting and conflation, which are possible in an arbitrary timed automaton, but not for automata in TA_S. These operations make the problem of optimal clock allocation difficult to solve in practice, as each of them can increase or decrease the chromatic number of the interference graph.

We identify an interesting class of timed automata, $TA_{DS} \subsetneq TA_S$, which has two important properties. First, the aforementioned clock operations cannot decrease the chromatic number of the interference graph generated from a member of TA_{DS}. Second, such an interference graph is chordal, so the cost of colouring is linear in the size of this graph.

The efficacy of our method is based on the safe simplifying assumption that all locations in the automaton are always reachable. Since this might sometimes not be the case, our clock allocation might not always be strictly optimal according to more general criteria, but we believe this caveat has little practical importance.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous referee of an earlier version of the paper for detailed and helpful comments.

REFERENCES

[1] Rajeev Alur, Costas Courcoubetis, and David Dill. 1993. Model-Checking in Dense Real-time. *INFORMATION AND COMPUTATION* 104 (1993), 2–34.

[2] Rajeev Alur and David L. Dill. 1994. A theory of timed automata. *Theor. Comput. Sci.* 126, Article 2 (April 1994), 53 pages.

[3] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. 1999. Event-Clock Automata: A Determinizable Class of Timed Automata. *Theor. Comput. Sci.* 211, 1-2 (1999), 253–273.

[4] Gerd Behrmann, Alexandre David, and Kim G. Larsen. 2004. A Tutorial on Uppaal. In *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004 (LNCS)*, Marco Bernardo and Flavio Corradini (Eds.). Springer–Verlag, 200–236.

[5] G. J. Chaitin. 1982. Register Allocation & Spilling via Graph Coloring. *SIGPLAN Not.* 17, Article 6 (June 1982), 4 pages. https://doi.org/10.1145/872726.806984

[6] Conrado Daws and Sergio Yovine. 1996. Reducing the number of clock variables of timed automata. In *Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS '96), December 4-6, 1996, Washington, DC, USA*. IEEE Computer Society, 73–81.

[7] Olivier Finkel. 2006. Undecidable Problems About Timed Automata. In *FORMATS'06 (Lecture Notes in Computer Science, Volume 4202.)*, Eugene Asarin and Patricia Bouyer (Eds.). Springer, France, 187–199. https://hal.archives-ouvertes.fr/hal-00121529

[8] Fanica Gavril. 1972. Algorithms for Minimum Coloring, Maximum Clique, Minimum Covering by Cliques, and Maximum Independent Set of a Chordal Graph. *SIAM J. Comput.* 1, 2 (1972), 180–187.

[9] Martin Grötschel, László Lovász, and Alexander Schrijver. 1988. *Geometric Algorithms and Combinatorial Optimization*. Algorithms and Combinatorics, Vol. 2. Springer.

[10] Shibashis Guha, Chinmay Narayan, and S. Arun-Kumar. 2014. Reducing Clocks in Timed Automata while Preserving Bisimulation. In *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*. Springer, 527–543.

[11] Thomas A. Henzinger. 1992. Sooner is Safer Than Later. *Inf. Process. Lett.* 43, 3 (1992), 135–141. https://doi.org/10.1016/0020-0190(92)90005-G

[12] Thomas Lengauer and Robert Endre Tarjan. 1979. A Fast Algorithm for Finding Dominators in a Flowgraph. *ACM Trans. Program. Lang. Syst.* 1, Article 1 (Jan. 1979), 21 pages.

[13] Gabriela Nicolescu and Pieter J. Mosterman. 2009. *Model-Based Design for Embedded Systems* (1st ed.). CRC Press, Inc., Boca Raton, FL, USA.

[14] Fernando Magno Quintão Pereira and Jens Palsberg. 2005. Register Allocation Via Coloring of Chordal Graphs. In *Programming Languages and Systems, Third Asian Symposium, APLAS 2005, Tsukuba, Japan, November 2-5, 2005, Proceedings*. Springer, 315–329.

[15] Massimiliano Poletto and Vivek Sarkar. 1999. Linear Scan Register Allocation. *ACM Trans. Program. Lang. Syst.* 21, Article 5 (Sept. 1999), 19 pages.

[16] Neda Saeedloei and Feliks Kluzniak. 2017. From Scenarios to Timed Automata. In *Formal Methods: Foundations and Applications - 20th Brazilian Symposium, SBMF 2017, Recife, Brazil, November 29 - December 1, 2017, Proceedings*. 33–51. https://doi.org/10.1007/978-3-319-70848-5_4

[17] Neda Saeedloei and Feliks Kluźniak. 2017. Optimal Clock Allocation for a Class of Timed Automata. http://www2.cs.siu.edu/~neda/report2.pdf. (2017). Technical Report.