Minimal DBM Subtraction

Alexandre David[1], John Håkansson[2], Kim G. Larsen[1], and Paul Pettersson[2]

[1] Department of Computer Science, Aalborg University, Denmark
{adavid,kgl}@cs.aau.dk.
[2] Department of Information Technology, Uppsala University, Sweden
{johnh,paupet}@it.uu.se.

Abstract. In this paper we present an algorithm to compute DBM subtractions that are minimal in the sense that the number of splits is guaranteed to be minimal and the resulting DBMs are disjoint to avoid any redundance. The subtraction is one of the few operations that result in a non-convex zone, and thus, requires splitting. It is of prime importance to reduce the number of generated DBMs in an exploration loop, e.g., for reachability, because the result is propagated and serves to compute further successors later.

1 Introduction

DBMs (difference bound matrices) [3, 10, 6] are efficient data structures to represent clock constraints in timed automata [1]. However, some operations require splitting, e.g. subtraction or some extrapolation algorithms [2] because DBMs can only represent convex zones. The resulting DBMs of a split are parts of successor states that will be used to compute further successors. Reducing splitting means to reduce the state explosion.

The new DBM library of the model-checker UPPAAL[1] supports subtractions and federations of DBMs to represent non-convex zones. There are other representations of zones that can deal with non-convex zones, such as CDDs [5] or CRDs [11]. In this paper we are concerned about how to solve the subtraction for the DBMs when the DBMs are already used in a model-checker. Depending on the operation, a given representation may be more or less efficient, we do not address this issue. Our contribution is the improvement on the standard subtraction algorithm to make it minimal at a reasonable cost (we do not increase the complexity of the subtraction).

We give the basic subtraction algorithm in Section 2 that we refine with a minimal number of splits in Section 3. We complete the algorithm with disjointness in Section 4 and we show with experiments in Section 5 that our algorithm improves the subtraction significantly.

2 DBM Subtraction

Notations. In this paper, a *zone* corresponds to the set of clock valuations that satisfy the constraints of a given DBM. Zones are convex by definition of DBMs.

[1] http://www.uppaal.com

A DBM C of dimension n is a set of constraints $x_i - x_j \sim b_{ij}$ where $\sim \in \{<, \leq\}$, $1 \leq i, j \leq n$, x_i and x_j the are clock valuations of the clocks i and j, and b_{ij} are the bounds of the constraints. This is indeed a conjunction of constraints: $C = \bigwedge_{1 \leq i,j \leq n}(x_i - x_j \sim b_{ij}) = \bigwedge_{1 \leq i,j \leq n} c_{ij}$ so we abuse the notation by writing $C = \bigwedge c_{ij}$ as a conjunction of constraints c_{ij} (that incorporate the inequality $\sim$), $C = \{c_{ij} | 1 \leq i, j \leq n\}$ as a set of constraints, or C may be used to refer to the zone itself (in appropriate context), i.e., the set of points satisfying the constraints of the DBM. The complement is defined as $\bar{\leq} = <$ and $\bar{<} = \leq$.

Definition 1 Operations on constraints For c_{ij} and z_{ij} constraints we define:

- $c_{ij} \leq z_{ij} \Leftrightarrow c_{ij} \Rightarrow z_{ij}$
- $c_{ij} < z_{ij} \Leftrightarrow c_{ij} \neq z_{ij} \wedge c_{ij} \leq z_{ij}$
- $\neg c_{ij} = \neg(x_i - x_j \sim b_{ij}) = x_j - x_i \bar{\sim} - b_{ij}$. Note that $\neg c_{ij}$ is a new c'_{ji} comparable with other constraints z_{ji}.
- $c_{ik} + c_{kj} = (x_i - x_k \sim b_{ik}) + (x_k - x_j \sim' b_{kj}) = x_i - x_j \sim'' b_{ik} + b_{kj}$ where $\sim'' = <$ if $\sim = <$ or $\sim' = <$, otherwise $\sim'' = \leq$.
- $c_{ik} - c_{jk} = c_{ik} + \neg c_{jk}$

A DBM is *canonical* if its constraints have been tightened by Floyd's shortest path algorithm [4]. We consider that DBMs are always canonical. We always use c_{ij}, respectively z_{ij}, to denote the constraints of the DBM C, respectively Z. A *facette* corresponding to a constraint c_{ij} (of a DBM) corresponds to the facette of the zone (polygon in dimension n) delimited by c_{ij}.

Subtraction Given two DBMs C and Z, we want to subtract Z from C. The resulting zone S can be defined as the zone satisfying the constraints of C and $\neg Z$. The result $S = C \wedge \neg Z$, denoted $C - Z$, can be written as:

$$S = C \wedge \neg(\bigwedge_{1 \leq a,b \leq n} z_{ab}) = \bigvee_{1 \leq a,b \leq n} (C \wedge \neg z_{ab}) \quad \text{(De Morgan law)}$$

, which is a union of C constrained by each of the negated constraints of Z. Figure 1 illustrates the basic subtraction algorithm in two dimensions, i.e., with two clocks. The result $S = C - Z$ is represented by the union of the six smaller DBMs to the right in the figure. The number of zones in S is bounded by n^2, and creating each of these zones is $O(n^2)$, so the complexity of the algorithm is $O(n^4)$ (in both time and space).

Basic Algorithm. The subtraction is computed as follows: $C - Z = \bigvee_{i,j}(C \wedge \neg z_{ij})$.

Improved Algorithm The first observation from this basic algorithm is that some splits may be avoided by taking into account only the edges that belong to the minimal graph [7] of the DBM. Figure 2 shows the reduced subtraction by using only these constraints. Since the set of constraints Z_m obtained from the minimal graph is equivalent to Z (in terms of zone and constraints), we compute $C - Z = C - Z_m$. The complexity for computing the minimal graph is $O(n^3)$ but it is worth doing since it is more important to reduce the number of DBMs in the result.

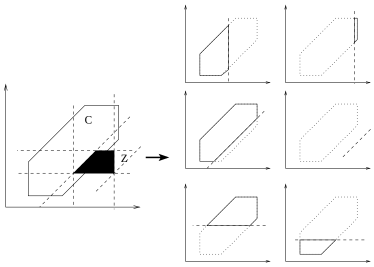

Fig. 1. Basic subtraction algorithm.

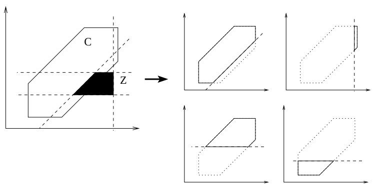

Fig. 2. Subtraction using the constraints part of the minimal graph.

3 Subtraction with Minimal Split

To further reduce the number of splits, let us consider the following four cases that arise on the constraints belonging to the minimal graph of the DBM to subtract:

1. The (negated) constraint reduces C to an empty zone: this corresponds to a false disjunct, and can be safely ignored.
2. The (negated) constraint has no effect on C: because C is convex, this means that the DBM Z to subtract is outside of C so we stop and the result is C.
3. The (negated) constraint is on a facet of Z that does not intersect C: we ignore it.
4. Otherwise we compute a split.

The third case is the contribution of our algorithm: the decision procedure is linear in the number of clocks and can rule out constraints that do not affect the result. The main argument for ruling out these constraints comes from the convexity of C and Z: if a facet of Z does not intersect C, then it has no effect on the subtraction. In fact it is more subtle because in some cases we can remove facettes that intersect C. As a remark, the first case is redundant with the third case but it is a simple test used to rule out simple cases. The complexity of the additional test is $O(n)$ per constraint, which is $O(n^3)$ in total, so we do not make the subtraction complexity worse.

Definition 2 (Redundant split) A split on a constraint z_{ij} is redundant iff $C - Z = C - (Z \backslash z_{ij})$, where $Z \backslash z_{ij}$ denotes the set of constraints of Z without z_{ij}.

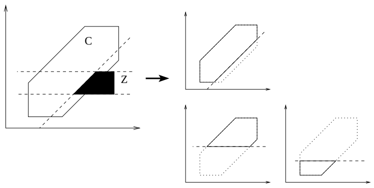

Fig. 3. Subtraction with minimal splitting.

We remove redundant facettes of Z with an intersection test with C, except for those that intersect C with a single point. Some of these facettes, as shown on Fig. 4, may be redundant and pass the intersection test. We define a predicate

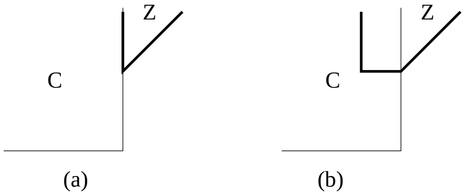

Fig. 4. Limit case for subtraction: (a) both facettes are needed, (b) the diagonal constraint is not needed.

to detect case (b): $limit(z_{ij}) = true$ if z_{ij} is non strict, it intersects C with one point, and $\exists k.\ (\leq 0) < z_{ik} + z_{kj}$, otherwise $limit(z_{ij}) = false$. In practice this is implemented in the intersection test by using the same k in the algorithm and by adjusting the comparisons. This adds nothing to the complexity. The test on $z_{ik} + z_{kj}$ is checking of it is possible to induce z_{ij} or not.

Minimal Split Algorithm. The algorithm is as follows:

1. If $\exists i, j.\ c_{ij} \leq \neg z_{ji}$ then $C - Z = \{C\}$.
2. Compute the minimal graph Z_m of Z.
3. $S = \{\}$
   $\forall z_{ij} \in Z_m.\ z_{ij} < c_{ij}$:
      if $limit(z_{ij})$ then skip z_{ij}
      $\forall k.\ k \neq i, k \neq j$:
         if $c_{ik} \leq \neg(z_{kj} - z_{ij})$ or $c_{kj} \leq \neg(z_{ik} - z_{ij})$
         then skip z_{ij}
         else $S = S \vee (C \wedge \neg z_{ij})$.
4. $C - Z = S$.

We use Z_M to denote the set of constraints z_{ij} for which the subtraction is computed. We abuse the notation with $C - Z_M$ denoting the subtraction $C - Z$ using only the constraints of $Z_M \subseteq Z$.

Lemma 1 (Soundness of minimal split) The algorithm computes the subtraction $C - Z$ correctly.

Proof:

1. The first condition detects $C \cap Z = \emptyset$ by comparing lower and upper bounds of C and Z. Since the represented zones are convex, we detect $C \cap Z = \emptyset$ and trivially the result is $S = \{C\}$ in this case.
2. Computing the minimal zone does not change the result since Z_m is equivalent to Z.
3. The loop on z_{ij} avoids the constraints z_{ij} s.t. $c_{ij} \leq z_{ij}$. Since C is not empty, $\neg c_{ji} \leq c_{ij} \leq z_{ij}$, thus $\neg z_{ij} \leq c_{ji}$. So constraining c_{ji} by $\neg z_{ij}$ results in $\neg z_{ij}$ (we tighten the constraint by taking the minimal one) and we have $c_{ij} + c_{ji} = c_{ij} - z_{ij} \leq z_{ij} - z_{ij} = (< 0)$. This tightening is too strong and can be safely ignored.
   The constraints z_{ij} skipped by $limit(z_{ij})$ are by definition redundant.
   The second loop on k projects the *facette* z_{ji} on z_{ij}, that is, Z' is the constrained zone Z such that $z_{ij} + z'_{ji} = (\leq 0)$. We compare the tightened constraints $z'_{jk} = z'_{ji} + z_{ik}$ and $z'_{ki} = z_{kj} + z'_{ji}$ against c_{kj} and c_{ik} respectively. If we have $c_{ik} \leq \neg z'_{ki}$ or $c_{kj} \leq \neg z'_{jk}$ then Z projected on z_{ij} does not intersect C, which is, the facette z_{ij} does not intersect C. Therefore $C - Z = C - (Z \backslash z_{ij})$, so we can safely ignore z_{ij}.
   The remaining constraints are used for the subtraction and compute $C - Z$ as shown previously. □

Lemma 2 (Facette intersection) In our algorithm, the facette z_{ij} intersects C iff $\forall k.\ k \neq i, k \neq j : \neg(z_{kj} - z_{ij}) < c_{ik}$ and $\neg(z_{ik} - z_{ij}) < c_{kj}$.

Proof:

1. If $\exists k$ s.t. one of the conditions fails, then the facette z_{ij} does not intersect C. This holds by convexity of the projected zone Z on z_{ij} (even if we haven't prooved that the updated constraints z'_{jk} and z'_{ki} are the tightest).
2. If the facette z_{ij} does not intersect C then one condition fails. The constraints z'_{jk} and z'_{ki} are the tightest constraints that result from the projection of Z on z_{ij} by definition of the tightening algorithm [10] that we can rewrite in our case when we tighten z_{ji} (Fig. 5). Since z_{ik} and z_{kj} are already the tightest for Z, they will not be tightened for the projection, so $z'_{ik} = z_{ik}$ and $z'_{kj} = z_{kj}$. By convexity, if there is no intersection, then one condition among $\neg z'_{ik} < c_{ki}$, $\neg z'_{ki} < c_{ik}$, $\neg z'_{jk} < c_{kj}$, or $\neg z'_{kj} < c_{jk}$ must fail. The algorithm only checks the conditions for the tightest z'_{ki} and z'_{jk}. We have already tested in the first step that all $c_{ij} > \neg z_{ji}$ so we know that the conditions on z'_{ik} and z'_{kj} already hold. The other constraints of the projected DBM do not matter since it is a projection on z_{ij}. □

Lemma 3 (No redundant split) The algorithm computes the subtraction $C - Z$ with no redundant split: $\{z_{ij} \in Z_M | C - Z_M = C - (Z_M \backslash z_{ij})\} = \emptyset$.

```
for k in 1..n
   if z_ji + z_ik < z_jk then z_jk := z_ji + z_ik
for k in 1..n
   if z_kj + z_ji < z_ki then z_ki := z_kj + z_ji
   for l in 1..n
      if z_ki + z_il < z_kl then z_kl := z_ki + z_il
```

Fig. 5. Specialization of Floyd's algorithm when tightening z_{ji}.

Proof:

- The result is empty: $S = \{\}$. The algorithm does not constrain C at all and there is 0 split.
- Dimension n of C (and Z) verify $n \leq 2$: there is no k to consider in the algorithm and the previous conditions only remove trivial cases.
- The result is not empty and $n > 2$. Suppose there exists one redundant split, i.e., $\exists z_{ij} \in Z_M.\ C - Z_M = C - (Z_M \backslash z_{ij})$. We pick $p \in C \cap Z$ on the facette z_{ij} (on the inside if z_{ij} is strict), which is possible by construction since this facette intersects C (lemma 2). We pick a point p' s.t. p' satisfies $Z \backslash z_{ij}$, $\neg z_{ij}$, and C. In dimension n, picking a p' near p such that $\overrightarrow{p' - p}$ is orthogonal to the facette z_{ij} will do. Since $\neg z_{ij} < c_{ji}$ (first step of the algorithm) and $c_{ij} < z_{ij}$ (condition on the z_{ij} in the loop of the second step), this is possible. For this p' we have $p' \in Z \backslash z_{ij}$, which contradicts the hypothesis $p' \in C - Z = C - Z_M = C - (Z_M \backslash z_{ij}) = C - (Z \backslash z_{ij})$. □

Property 1 (Completeness of minimal split) The algorithm computes the subtraction with DBMs with the minimal possible number of splits.

Proof: Z_m contains the minimal number of constraints necessary to describe Z. By definition of the minimal graph, any sub-set Z' of Z giving the same zone contains Z_m. $Z_M \subseteq Z_m$ contains no redundant constraint to compute the subtraction $C - Z = \bigvee_{z_{ij} \in Z_M} (C \wedge \neg z_{ij})$. Every term $C \wedge \neg z_{ij}$ in the union is necessary (or the split would be redundant by definition). Therefore the split is minimal. □

4 Minimal Subtraction

Having the minimal number of DBMs as the result from a subtraction is not enough: if the successor states are operands of subsequent subtractions then there should be no overlapping between them, otherwise these subtractions may have redundant points. The bad thing could be that future generated DBMs may not be simply comparable with respect to inclusion checking, which means, that even if some of them are redundant, they will be kept. We propose a simple procedure to reduce the size of the resulting DBMs to make them disjoint. The ordering of the splitting procedure affects the result but it is still guaranteed to be minimal and disjoint. The copy operation for each zone in $C - Z$ is still dominating, so the subtraction is still $O(n^4)$.

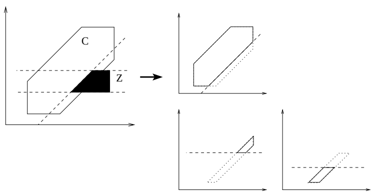

Fig. 6. Subtraction with minimal splitting and disjoint result.

Algorithm. Applying the algorithm of Section 3 gives the set $Z_M \subseteq Z$ of constraints z_{ij} to use in the subtraction. We compute $C - Z$ as follows:

1. Compute Z_M.
2. $S = false$ (or $\{\}$ if seen as a set)
   $\forall z_{ij} \in Z_M$:
      $S = S \vee (C \wedge \neg z_{ij})$
      $C = C \wedge z_{ij}$.
3. $C - Z = S$

Lemma 4 (Soundness of disjointness) The algorithm still computes the same subtraction $C - Z$.

Proof: C is constrained at every step by z_{ij} but the points we remove from C already belong to the result S. □

Lemma 5 (Completeness of disjointness) S is a union of disjoint DBMs: $\forall s_1, s_2 \in S.\ s_1 \neq s_2 : s_1 \cap s_2 = \emptyset$.

Proof: $\forall s_1, s_2 \in S.\ s_1 \neq s_2 \Rightarrow \exists z_{ij} \in M'_Z.\ s_1 \wedge \neg z_{ij} = s_1$ and $s_2 \wedge z_{ij} = s_2$ (or $s_2 \wedge \neg z_{ij} = s_2$ and $s_1 \wedge z_{ij} = s_1$) by construction of S. Therefore $s_1 \cap s_2 = \emptyset$. □

Corollary 1 (Minimal subtraction) The algorithm computes a minimal subtraction in the sense that the number of DBM splits is minimal and the resulting DBMs contain no redundant point.

5 Experiments

In these experiments we show that the extra effort spent on computing a minimal subtraction is valuable for a model-checker. Although a subtraction is approximately twice as expensive to compute, the overall performance and memory consumption is improved and we reduce the problem of propagation of splits as mentioned previously.

Experiment 1 We benchmark a particular expensive function of our DBM library [2]: *e*xpensiveReduce that takes as an input a federation (a.k.a. union of zones), computes cross-subtractions between DBMs to detect and remove redundant DBMs of the federation w.r.t. to the union of zones (if $D_i \subseteq \bigcup_j D_j$). With n being the dimension of the DBMs and m the length of the federation, the space complexity is $O(dim^2 * (dim^2)^m)$ and the time complexity $O(dim^4 * (dim^2)^m)$. The term $(dim^2)^m$ is the worst number of splits needed. This function should of course never be used, except for benchmarking the subtraction. The interesting point is that its behaviour mimics what could happen in a model-checker where the result of a subtraction is used for further subtractions. The experiment is carried out on a Pentium 4 running Linux 2.6.8 with 512M of RAM and 512M of swap. The benchmark iterates 1000 times and tries to generate bad DBMs randomly. We provide 3 runs with different random seeds (0, 111, 333222111). Table 1 shows the result (mx DBMs is the length of the federation). We give the memory in megabytes (M) and time in seconds (s) needed. The different columns correspond to the minimal subtraction (Min+Disj), the disjoint subtraction alone (Disj), the minimal split alone (Min), and none of these two improvements (None). The minimal graph is always used to have a sensible basic implementation to compare with.

The disjointness (Disj) is the dominant factor in reducing the space explosion. The combination with the minimal split (Min+Disj) improves even more, which is particularly clear for “longer” federations. The seemingly incoherent results are due to the randomness of the benchmark. The explosion of the case where no optimisation is done illustrates the bad complexity of the operation.

Experiment 2 We have implemented a timed game reachability engine (part of another project), whose purpose is to find winning strategies. The binaries we used are available on-demand, as well as the test example. This engine implements a special reachability algorithm and one critical operation is the *p*re operation in the backward phase. In this version, it is using 3 subtractions for its definition, the arguments being federations. The test example is the production cell [8, 9]. Table 2 shows the results of this experiments: the column *Pre* shows how many times the *pre* operation was called in the search. The other columns are correspond to the settings of the previous experiment. The same machine is used.

The results are in contrast with the previous experiment: this time the minimal split (Min) is more important. This shows that the performance depends in practice on the configuration of the federations. Another surprising result is the bad behaviour of the disjointness alone (Disj) where the combination (Min+Disj) gives the best results. The explanation comes from the *inclusion checking* of the zones: in this case zones are disjoint but they are needlessly split in small parts, which is very bad for the inclusion checking (this removes included zones in the search). The presented combined optimisations give a significant gain in time in

[2] distributed for free from the UPPAAL web page *http://www.uppaal.com*

DBM 7x7	Min+Disj	Disj	Min	None
1x DBM	2.5M, 0.1s	2.5M, 0.1s	2.6M, 0.1s	2.7M, 0.1s
	2.5M, 0.1s	2.5M, 0.1s	2.6M, 0.1s	2.7M, 0.1s
	2.5M, 0.1s	2.5M, 0.1s	2.6M, 0.1s	2.7M, 0.1s
2x DBMs	2.6M, 0.1s	2.6M, 0.1s	2.5M, 0.1s	3.8M, 0.1s
	2.7M, 0.1s	2.5M, 0.1s	3.3M, 0.2s	3.8M, 0.2s
	2.5M, 0.1s	2.6M, 0.1s	3.1M, 0.2s	5.0M, 0.2s
3x DBMs	2.8M, 0.2s	3.0M, 0.2s	8.3M, 0.4s	14M, 0.5s
	2.6M, 0.2s	2.6M, 0.2s	3.8M, 0.3s	10M, 0.4s
	2.5M, 0.s2	2.6M, 0.2s	6.8M, 0.3s	10M, 0.4s
4x DBMs	2.7M, 0.3s	2.8M, 0.3s	586M, 4.5s	-
	2.7M, 0.3s	2.8M, 0.3s	104M, 1.5s	353M, 3.1s
	2.7M, 0.3s	2.9M, 0.3s	44M, 0.8s	167M, 1.5s
5x DBMs	2.7M, 0.5s	2.8M, 0.5s	58M, 1.8s	238M, 4.4s
	2.6M, 0.5s	2.7M, 0.5s	27M, 1.4s	103M, 2.6s
	2.7M, 0.5s	2.9M, 0.5s	342M, 3.4s	-
6x DBMs	3.0M, 0.6s	3.1M, 0.6s	794M, 7.2s	-
	2.7M, 0.6s	2.8M, 0.6s	416M, 6.6s	-
	3.1M, 0.6s	3.3M, 0.6s	-	-
7x DBMs	2.9M, 0.8s	3.0M, 0.9s	-	-
	3.0M, 0.8s	3.3M, 0.9s	-	-
	3.0M, 0.8s	3.1M, 0.9s	-	-
14x DBMs	3.8M, 2.9s	4.3M, 3.2s	-	-
	3.0M, 2.9s	3.7M, 3.2s	-	-
	3.9M, 2.9s	4.7M, 3.3s	-	-
28x DBMs	7.0M, 13.9s	9.7M, 17.6s	-	-
	4.9M, 13.6s	7.1M, 17.6s	-	-
	6.3M, 13.7s	9.7M, 17.6s	-	-
56x DBMs	13.0M, 92.7s	25.3M, 147s	-	-
	11.7M, 92.5s	22.3M, 147s	-	-
	11.3M, 98.7s	29.7M, 157s	-	-

Table 1. Results with the expensiveReduce function.

Plates	Pre	Min+Disj	Disj	Min	None
4	20515	10M, 1.1s	10M, 1.5s	10M, 1.1s	10M, 1.2s
5	88678	43M, 7.4s	46M, 17.7s	43M, 8.0s	43M, 16.4s
6	454679	249M, 108.4s	400M, 18772s	249M, 110.7s	249M, 157.6s

Table 2. Results with the timed games prototype.

a realistic model-checker *in spite of* the algorithmic overhead of their implementations.

6 Conclusion

We have shown that it is more important to have a good result for the subtraction operation on DBMs, in the sense that the result has the fewest possible number of splits and contains no redundancy. Even if computing such a result involves an overhead, the overall performance is improved because these results are propagated in common reachability algorithms. We have presented and implemented such an efficient algorithm, efficient in the sense that the resulting complexity is the same as the original complexity $O(n^4)$, which means it is optimal.

References

1. Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In *Proc. of Int. Colloquium on Algorithms, Languages, and Programming*, volume 443 of *LNCS*, pages 322–335, 1990.
2. Johan Bengtsson. *Clocks, DBMs and States in Timed Systems*. PhD thesis, Uppsala University, 2002.
3. David L. Dill. Timing assumptions and verification of finite-state concurrent systems. volume 407 of *LNCS*, pages 197–212. Springer, 1989.
4. Robert W. Floyd. Acm algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.
5. Kim G. Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Clock difference diagrams. *Nordic Journal of Computing*, 6(3):271–298, 1999.
6. Kim G. Larsen, Paul Pettersson, and Wang Yi. Model-checking for real-time systems. In *Proc. of Fundamentals of Computation Theory*, number 965 in Lecture Notes in Computer Science, pages 62–88, August 1995.
7. Fredrik Larsson, Kim G. Larsen, Paul Pettersson, and Wang Yi. Efficient verification of real-time systems: Compact data structures and state-space reduction. In *Proc. of the 18*th *IEEE Real-Time Systems Symposium*, pages 14–24. IEEE Computer Society Press, December 1997.
8. Claus Lewerentz and Thomas Lindner. "production cell": A comparative study in formal specification and verification. In *KORSO – Methods, Languages, and Tools for the Construction of Correct Software*, volume 1009 of *LNCS*, pages 388–416. Springer-Verlag, 1995.
9. Helmut Melcher and Klaus Winkelmann. Controller synthesis for the "production cell" case study. In *Proceedings of the second workshop on Formal methods in software practice*, pages 24–36. ACM Press, 1998.
10. Tomas Gerhard Rokicki. *Representing and Modeling Digital Circuits*. PhD thesis, Stanford University, 1993.
11. Farn Wang. RED: Model-checker for timed automata with clock-restriction diagram. In Paul Pettersson and Sergio Yovine, editors, *Workshop on Real-Time Tools, Aalborg University Denmark*, number 2001-014 in Technical Report. Uppsala University, 2001.