Efficient Algorithms for Interface Timing Verification *

Ti-Yen Yen	Alex Ishii	Al Casavant	Wayne Wolf
Princeton University	NEC USA	NEC USA	Princeton University
Princeton, NJ 08544	Princeton, NJ 08540	Princeton, NJ 08540	Princeton, NJ 08544

Abstract

We present algorithms for computing separations between events that are constrained to obey relationships specified by an acyclic event graph. *The algorithms are useful for interface-timing verification, where separations are checked against timing requirments. The first algorithm computes separations when only* linear *and* max *constraints exist. The algorithm is conjectured to run in* $O(V^2 \log V + VE)$ *time. The second algorithm uses a branch-and-bound approach to compute separations when* min *constraints also exist. Experiments indicate the algorithms are efficient in practice.*

1 Introduction

The automation of interface timing verification is important for the design of reliable systems composed of several interconnected components. Due to manufacturing or environmental variation, the delay between two signal transitions of a component may not be a precise number, and is usually specified with a lower bound and an upper bound by the manufacturers. Exhaustive simulation, where only a particular delay value is checked at a time, is impractical for large systems. Therefore, analytical verification algorithms are necessary to ensure that all timing requirements will be satisfied for all possible delay values.

We will present an efficient interface timing verification algorithm based on the timing model used by Gahlinger [1], and by McMillan and Dill [2]. It uses two auxiliary graphs—compulsory constraints and slacks—to quickly check constraint satisfaction. Max-linear constraint systems are a subset of the more general min-max-linear form of the interface timing verification problem, while min-max-linear constraint satisfaction is known to be NP-complete. Max-linear constraint systems are general enough, however, to model a wide variety of interfaces. We have also used our efficient max-linear verification algorithm as the basis for a practical branch-and-bound algorithm for the general min-max-linear problem.

Our verification methods operate on a graph-based form of timing diagram; our present algorithm is restricted to acyclic timing diagrams. A timing diagram consists of a set of *signals* on which *events* occur, where an event is a change in the value of a signal. The interface specification includes *timing characteristics* and *timing requirements* which represent relationships between the times of events. Timing characteristics are represented by a set of *delay constraints*. A delay constraint c_{ij}, with a lower bound $lower[c_{ij}]$ and an upper bound $upper[c_{ij}]$, belongs to one of the three types:

- **linear**
  $\max_i(time[i] + lower[c_{ij}]) \leq time[j]$
  $\leq \min_i(time[i] + upper[c_{ij}])$
- **max**
  $\max_i(time[i] + lower[c_{ij}]) \leq time[j]$
  $\leq \max_i(time[i] + upper[c_{ij}])$
- **min**
  $\min_i(time[i] + lower[c_{ij}]) \leq time[j]$
  $\leq \min_i(time[i] + upper[c_{ij}])$

We model the timing diagram and its characteristics by an **event graph** whose nodes represent events and whose weighted, directed edges represent the delay constraints. Two events may have an uncertainty interval for their time separation. Our goal is to verify that all allowable values of the time separations satisfy the timing requirements and the given constraints are *consistent*—at least a solution exists. We need to find only the maximum separations, because a minimum separation value can be easily computed from a maximum separation value by $\min(time[i] - time[j]) = -\max(time[j] - time[i])$. The typical timing verification problem is created when two components, each with their own timing diagram specification, are plugged together to create a system; computing sep-

*This work was supported by NEC, USA.

arations ensures that each component meets the requirements of the other.

Vanbekbergen et al. [3] give a graphic interpretaion of the differences among the three types of delay constraints. Gahlinger [1] points out all the three types of constraints are necessary for modeling interface timing. It is important not to confuse the max and min constraints described here with the maximum or minimum constraints used in some other literature. Generally, those maximum or minimum constraints are the same as the upper bounds or lower bounds in the linear constraints. While *all* the lower bounds and *all* the upper bounds of linear constraints must be satisfied, only *one* upper bound of the max constraints entering a node has to be satisfied and only *one* lower bound of the min constraints entering a node has to be satisfied. This alternative or nonlinearity of satisfying bounds for max or min constraints makes the problem more complicated than the purely linear constraint cases such as layout compaction [4].

The next section reviews previous work on interface timing verification. Section 3 describes our algorithm for verification of max-linear constraint systems. Section 4 describes a branch-and-bound algorithm for solving min-max-linear constraint systems. Section 5 describes the results of experiments with our implementation of the verification algorithm. Due to space limitations, we omit the proofs of all the theorems in this paper.

2 Previous work

Brzozowski et al. [5] use shortest-paths algorithms to solve the problem with only linear constraints. Both Vanbekbergen et al. [3] and McMillian et al. [2] propose polynomial-time algorithms for the problem with only max constraints.

So far no polynomial-time algorithm has been published for any mixed-constraint problem. McMillan and Dill [2] prove that the problem with both max and min constraints is NP-complete. They solve the problem with all the three types of constraints by eliminating min constraints and dividing the problem into subproblems which contain only linear and max constraints. Since min constraints are few in most applications, the number of subproblems may not be very large in practice. Unfortunately, their algorithm for a subproblem with max and linear constraints has worst-case exponential running time which depends on delay values and might even diverge for some pathological cases. Walkup and Borriello propose an improved algorithm whose running time is independent of delay values for max-linear problem [6]. They conjecture that their algorithm runs in $O(V^6)$ in the worst case to determine all $|V|^2$ maximum event separations, or $O(V^5)$ for all $|V|$ maximum separations from a single node. The complexity of the min-max linear constraint graph problem is summarized in Figure 1.

Burks and Sakallah [7] apply mathematical programming techniques to solve the general min-max linear programming problem. Although their approach can be used to solve the interface timing problem, by concentrating on only the special case of the general min-max linear programming problem, we are able to develop a much more efficient graph-based algorithm.

3 The algorithm for max and linear constraints

We present an efficient algorithm to solve the problem with only max and linear constraints. Given an event graph $G = (V, E)$ and a node $s \in V$, the algorithm finds the maximum time separations $sepa[u] = \max(time[u] - time[s])$ for all the nodes $u \in V$. We may apply the algorithm several times for different s in the timing requirements, or $|V|$ times for all-pairs maximum time separations.

Although we are dealing with acyclic timing diagrams, the constraint graph can be cyclic when we treat the lower bound and the upper bound of a delay constraint as different edges. McMillan and Dill's algorithm [2] runs in time proportional to delay values when there is a *false* negative cycle, a negative cycle containing an upper bound of a max constraint which does not have to be satisfied. Unlike the problem with only linear constraints, where a negative cycle indicates an inconsistency, their algorithm might require an exponential number of cycle traversals before identifying a false cycle. Given this observation, we derive our algorithm in the following two steps.

- Ignore the upper bounds of the max constraints. The remaining bounds must be satisfied for all feasible solutions and are *compulsory*. Generate a constraint graph with only compulsory bounds. Use a longest-paths algorithm to obtain the smallest separation values that satisfy all compulsory bounds.
- Reintroduce the upper bounds of the max constraints and increase the separation values according to the *slacks* of the bounds. A procedure similar to a shortest-paths algorithm is used to calculate the amounts by which the separation values can increase. The key advantage of this

Constraint type	Complexity	Proposed by
linear only	$O(VE)$	Shortest-paths algorithms
max only	$O(V^2)$	Vanbekbergen et al.
max only	$O(E)$	McMillian & Dill
max + linear	$O(V^5)$ conjecture	Walkup & Borriello
max + linear	$O(V^2 \log V + VE)$ conjecture	This paper
min + max	NP-complete	McMillian & Dill
min + max + linear	NP-complete	McMillian & Dill

Figure 1: The complexity for deriving the maximum separations of all nodes from a single node for various types of constraints.

approach is that we can avoid suffering from false negative cycles because all the slacks are nonnegative.

- Iteratively relax the separations. Repeat the above step until no relaxation is possible.

We formalize the first step of the algorithm as follows.

Definition 1 (Compulsory constraint graph) Given an event graph $G = (V, E)$ and a source node s, the corresponding *compulsory constraint graph* $G_c = (V, E_c)$ is a weighted directed graph, where E_c contains the following edges. For each linear or max constraint $c_{ij} \in E$, $e_{ij} \in E_c$ and has weight $lower[c_{ij}]$. For each linear constraint $c_{ij} \in E$, $e_{ji} \in E_c$ and has weight $-upper[c_{ij}]$. For each node i, $e_{si} \in E_c$ and has weight `-MAXINT`, where `MAXINT` is a number larger than any sum of the finite bounds, but obeys the arithmetic rules of finite numbers, ■

The $O(VE)$ single-source longest-paths algorithm [8] can be applied to G_c. If there is a positive cycle in G_c, the problem must be inconsistent and the algorithm can exit. The compulsory constraint graph for an example modified from [2] is shown in Figure 2.

The second step proceeds with an auxiliary graph defined below.

Definition 2 (Slack graph) Given an event graph $G = (V, E)$, a source node s, and a separation value $sepa[i]$ for each node $i \in V$, the *slack graph* $G_s = (V, E_s)$ is a weighted directed graph, where E_s is defined as follows. For each constraint $c_{ij} \in E$, $e_{ji} \in E_s$ and has weight $sepa[j] - sepa[i] - lower[c_{ij}]$. Add $e_{ij} \in E_s$ with weight $upper[c_{ij}] + sepa[i] - sepa[j]$, if the weight is nonegative. Mark e_{ij} as **max-optional** if c_{ij} is a max constraint, otherwise it is *compulsory*. If a node u is a *max event*, an event with some max constraints entering it, and no max-optional edge enters u, add a max-optional edge e_{su} with weight zero. ■

Note the weights in G_s are the slacks of the bounds in the constraints. We ignore the unsatisfied upper bounds of max constraints in G_s, so all the weights of the edges must be nonnegative. It can be easily shown that we cannot increase $sepa[j]$ by $weight[e_{ij}]$ more than the amount by which we increase $sepa[i]$, unless either some linear constraint between i and j is violated, or some other max constraint c_{kj} is satisfied. The amount the separation of a node can be increased by considering all the slacks into it is defined as follows.

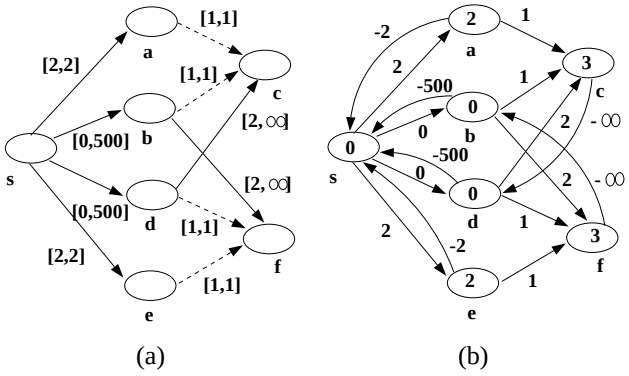

Figure 2: (a) An event graph. The solid lines are linear constraints while the dotted lines are max constraints. (b) The corresponding compulsory constraint graph. The node values are the longest-path weights, which are the initial values of the time separations.

Definition 3 (Shortest slack) Suppose the *shortest slack* from s to i is $\delta[i]$ for each node i in G_s. Then $\delta[i] = \min(\delta_l[i], \delta_m[i])$, where

$$\delta_l[i] = \min_{e_{ji} \text{ is compulsory}} (\delta[j] + weight[e_{ji}])$$

$$\delta_m[i] = \max_{e_{ji} \text{ is max-optional}} (\delta[j] + weight[e_{ji}])$$

If no compulsory e_{ji} exists, $\delta_l[i] = \infty$. If no max-optional e_{ji} exists, $\delta_m[i] = \infty$. ■

Because all the compulsory bounds must be satis-

```
ShortestSlack(G_s, s)
{
  for each node i {
    enqueue(Q, i);
    d[i] = ∞;
    m[i] = 0;
    n[i] = no. of max-optional edges entering i;
  }
  d[s] = 0;
  while (Q ≠ 0) {
    Find u in Q with a minimum d[u];
    for each edge e_ui {
      /* Relax the edges */
      t = d[u] + weight[e_ui];
      if (e_ui is max-optional) {
        n[i] − −;
        if (m[i] < t)
          m[i] = t;
        if (n[i] == 0 and d[i] > m[i])
            d[i] = m[i];
      }
      else if (d[i] > t)
        d[i] = t;
    }
    dequeue(Q, u);
  }
}
```

Figure 3: The procedure `ShortestSlack` for computing the shortest slack from a source node s for each node in a slack graph G_s.

```
MaxSeparation(G, s)
{
  Construct the compulsory constraint graph G_c.
  if (!LongestPath(G_c, s) )
    return INCONSISTENT;
  for each node u {
    sepa[u] = the longest path weight of u;
  }
  /* Iterative relaxation */
  do {
    Construct the slack graph G_s.
    ShortestSlack(G_s, s);
    change = NO;
    for each node u {
      if (δ[u] == ∞)
        sepa[u] = MAXINT;
      else if ((δ[u] > 0) {
        sepa[u]+ = δ[u];
        change = YES;
      }
    }
  } while (change == YES);
  for each node u
    if (sepa[u] ≥ MAXINT) sepa[u] = ∞;
  if (all constraints are satisfied) return CONSISTENT;
  else return INCONSISTENT;
}
```

Figure 4: The algorithm `MaxSeparation` for finding the maximum separations from a source node s for an event graph G, where only max constraints and linear constraints exist.

fied, δ_l is computed by the min function of the slacks. Only one upper bound corresponding the max-optional edges needs to be satisfied, δ_m is computed by the max function to increase the separations as much as possible. We can increase $sepa[i]$ by $\delta[i]$ without violating any already satisfied constraints. The procedure `ShortestSlack`(G_s) shown in Figure 3 computes $\delta[i]$ for each node i. It is based on Dijkstra's shortest-paths algorithm [8]. During execution, for each node i, let $d[i]$ be a *shortest slack estimate*, $n[i]$ be the number of max-optional edges entering i, and $m[i]$ be the temporary value for $\delta_m[i]$. If we implement the priority queue Q with a Fibonacci heap, the complexity of `ShortestSlack` is $O(V \log V + E)$. The following theorem shows the correctness of this procedure.

Theorem 1 After we run the procedure `ShortestSlack`(G_s, s), at termination $d[u] = \delta[u]$ for all nodes u in G_s. ■

Though the shortest slack $\delta[u]$ in Definition 3 may not be unique, it can be shown `ShortestSlack` computes shortest slacks. Moreover, the shortest slacks computed by `ShortestSlack` can be used by `MaxSeparation`(G, s), shown in Figure 4, to find the maximum time separations from a single source in an event graph with only max and linear constraints. Algorithm `MaxSeparation` increases the time separation $sepa[u]$ by $\delta[u]$, updates the slack graph, and repeats the steps until $\delta[u] = 0$ or $\delta[u] = \infty$ for all $u \in V$. The iteration steps for the example in Figure 2 are shown in Figure 5.

We can prove that the algorithm produces the correct result.

Theorem 2 (Correctness) If the algorithm `MaxSeparation`(G, s) returns `CONSISTENT`, $sepa[u]$ is the maximum separation from the source node s to node u for all nodes u in G. If it returns `INCONSISTENT`, the given constraints in G are inconsistent. ■

The complexity of constructing a slack graph is $O(E)$, that of `ShortestSlack` is $O(V \log V + E)$, and that of increasing the separations is $O(V)$. The key to know

the worst-case running time of the algorithm is to show a bound on the iterations for the **do-while** loop.

Theorem 3 The algorithm must terminate in finite steps. ■

Conjecture 4 (Complexity) The number of iterations for the **do-while** loop in the algorithm `MaxSeparation` is bounded by $|V| + 1$. ■

Unfortunately, we have not been able to prove this conjecture rigorously. If this conjecture is true, the algorithm for finding the maximum separations from a single source when only max and linear constraints exist has an asymptotic upper bound of $O(V^2 \log V + VE)$.

4 The algorithm for all types of constraints

The fact that the problem with all min, max and linear constraints is NP-complete [2] justifies a branch and bound approach. In particular, we make the following modifications to algorithm `MaxSeparation`. Without loss of generality, we assume no event has both max and min constraints entering it. An event with min constraints entering it is a *min event*. The major modifications are as follows:

- Add edges for the upper bounds of min constraints to the compulsory constraint graph. Add compulsory edges for upper bounds of min constraints and *min-optional* edges for lower bounds of min constraints to the slack graph.
- In the `ShortestSlack` procedure, for each dequeued min event, if the relaxation of any min-optional edge does not reduce the shortest slack estimate, ignore the relaxation of all the min optional edges out of the event. Otherwise, choose one min-optional edge each time, save the current status of the graph, delete all the other min-optional edges, do a recursive call to start a new subproblem, and restore the status of the graph after the subproblem is finished.
- The maximum separation of each event is the maximum of the separations given by by all the subproblem.

Our algorithm for each subproblem, where only max and linear constraints exist, is efficient. In the worst case, if U is the set of all min events, we will have $\Pi_{u \in U}$ `indegree`$[u]$ subproblems, where `indegree`$[u]$ in the number of min constraints entering u. We avoid

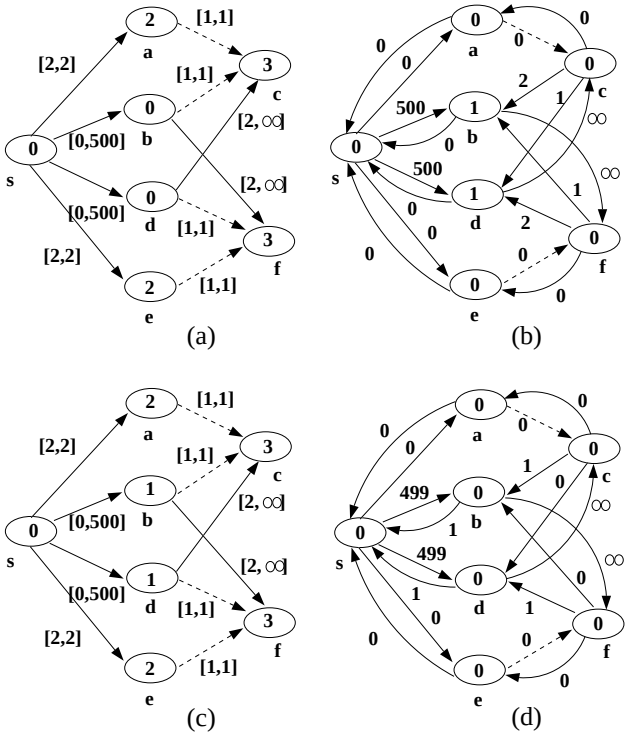

Figure 5: (a) The event graph whose node values are initial time separations. (b) The slack graph in the first iteration. The dotted lines are max-optional edges. The node values are the shortest slacks. (c) The event graph whose node values are the time separations after each initial values is increased by the shortest slack. (d) The slack graph in the second iteration. Because all the shortest slacks are zeros, and all constraints are satisfied, we have obtained the maximum separations in (c).

generating a subproblem when it is possible. If the lower bound of a min constraints entering an event has been satisfied because of some other constraints, the alternative of choosing a min constraint disappears. When we generate a new subproblem, we continue it with the current status instead of running the procedure from the beginning. These approaches make the algorithm for the problem with all three types of constraints more efficient than dividing the problem into subproblems at first and solving each subproblems independently.

5 Experimental results

The first example for Intel 8086 ROM read cycle is from [1], and used by [2]. As pointed out in [2],

Example	#events	#min	#max	#linear	#require	satisfied?	CPU time
8086 + 2716	13	2	2	14	6	NO	0.04s
80386 + 2147H-3	25	2	9	21	11	NO	0.19s
80386 + 2147H-2	25	2	9	21	11	YES	0.19s

Figure 6: The problem size and the result. The columns show the number of events, min constraints, max constraints, linear constraints, timing requirements respectively. Then the result and the execution time given by our algorithm are shown.

because the modeling of the address latch is incorrect, the result shows the timing is not satisfied.

The second and third examples are cache read hit bus cycles for an Intel 80386 system. The components involved are the Intel 80386 CPU, the Intel 82385 cache controller [9], 2147H-3 (55 ns) Static RAM [10], an address latch 74FCT373T and a data buffer 74FBT245A [11]. Suppose the clock is 20MHz. The algorithm shows the timing is not satisfied. However, if we replace 2147H-3 with 2147H-2 (45 ns), all the timing requirements are satisfied.

The problem size and the execution time of our algorithm, measured on a Sun Sparc ELC Workstation, for each example is shown in Figure 4.

6 Conclusion

This paper has presented an algorithm for computing event-separations in constraint systems consisting of max and linear constraints. The algorithm achieves its efficiency by using new techniques for eliminating false negative constraint cycles. We have also proposed a new branch-and-bound algorithm, embedded in the algorithm for the max-linear problem, to solve the problem with min, max, and linear constraints. We believe that the elimination of false negative constraint cycles, intrinsic to the algorithm for the max-linear problem, will allow the branch-and-bound algorithm to be very efficient for typical problems.

We are working to use these algorithms to develop a complete verification methodology for hardware interfaces, particularly systems which include microprocessors. We hope to extend these algorithms to deal with cyclic timing diagrams, a problem which has been explored by Amon, et.al [12]. In addition, we hope that similar constraint-graph-based techniques can be used to analyze the complex system timing that occurs when interfaces "switch modes" and must simultaneously meet the constraints of multiple timing diagrams, a problem partially addressed by Daga and Birmingham [13] with the use of annotated state transition graphs.

References

[1] Tony Gahlinger. *Coherence and satisfiability of waveform timing specification.* PhD thesis, University of Waterloo, 1990.

[2] K. McMillan and D. Dill. Algorithms for interface timing verification. In *Proceedings, IEEE International Conference on Computer Design*, 1992.

[3] P. Vanbekbergen, G. Goossens, and H. De Man. Specification and analysis of timing constraints in signal transition graphs. In *Proceedings, the European Conference on Design Automation*, 1992.

[4] W. H. Wolf and A. E. Dunlop. Symbolic layout and compaction. In B. T. Preas and M. J. Lorenzetti, editors, *Physical Design Automation of VLSI Systems.* Benjamin-Cummings, 1988.

[5] J. A. Brzozowski, T. Gahlinger, and F. Mavaddat. Consistency and satisfiability of waveform timing specification. *Networks*, January 1991.

[6] E. A. Walkup and G. Borriello. Interface timing verification with application to synthesis. In *Proceedings, Design Automation Conference*, 1994.

[7] T. Burks and K. Sakallah. Min-max linear programming and the timing analysis of digital circuits. In *Proceedings, IEEE International Conference on Computer-Aided Design*, 1993.

[8] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* McGraw-Hill, 1990.

[9] Intel Corporation. *Microprocessors*, 1990.

[10] Intel Corporation. *Memory*, 1990.

[11] Integrated Device Technology, Inc. *Logic Data Book*, 1990.

[12] T. Amon, H. Hulgaard, S. M. Burns, and G. Borriello. An algorithm for exact bounds on the time separation of events in concurrent systems. In *Proceedings, IEEE International Conference on Computer Design*, 1993.

[13] A. J. Daga and W. P. Birmingham. VITCh: A methodology for the timing verification of board-level circuits. In *Proceedings, TAU Workshop*, 1993.