Algorithms for Interface Timing Verification*

Kenneth L. McMillan
School of Computer Science
Carnegie Mellon University
Pitssburgh, PA 15213

David L. Dill
Computer Systems Laboratory
Stanford University
Stanford, CA 94305

Abstract

We examine algorithms for analyzing systems of inequalities with min/max *constraints that arise in interface timing specifications. A general form of the inequality is shown to be NP-complete but some interesting special cases can be solved efficiently. An branch-and-bound solution to the general case and applied to a previously published example.*

1 Introduction

The analysis of timing is crucial to the design and synthesis of high speed digital circuits. This is especially true in the case of interfaces between system components, where asynchronous timing enters the picture and failures can be difficult to reproduce. The interface timing verification problem is to determine whether the aggregated timing characteristics of two components that are connected meet specified timing requirements, such as setup and hold times.

We pose the problem of timing verification as one of finding the maximum achievable *separation* $s_{ij} = \max(t_j - t_i)$ between two events i and j under a system of timing constraints of the form:

$$t_j = \min_{i \in \text{preds}(i)} (t_i + \delta_{ij}) \tag{1}$$

or

$$t_j = \max_{i \in \text{preds}(i)} (t_i + \delta_{ij}). \tag{2}$$

where t_i is the time of event i and δ_{ij} is the *delay* from event i to event j. The delays are constrained to fall within fixed upper and lower bounds u_{ij} and l_{ij}. That is,

$$l_{ij} \leq \delta_{ij} \leq u_{ij}. \tag{3}$$

*This work was supported by the Semiconductor Research Corporation, Contract no. 91-DJ-205, and by the Stanford Center for Integrated Systems, Research Thrust in Synthesis and Verification of Multi-Module Systems. The first author was supported by and AT&T fellowship.

We also consider the addition of linear constraints of the form

$$t_j - t_i \leq s_{ij} \tag{4}$$

The complexity of optimizing the separation between every pair of events varies radically with the types of constraints which are allowed. For the case of acyclic systems of constraints using only the max function, we give an algorithm with complexity $O(n^3)$. If both the min and max functions are allowed, we show that the problem is NP-complete. For the case when max constraints and linear constraints are allowed, we give an algorithm that is exponential in the worst case, but can be feasible in practice. We do not know a polynomial-time algorithm, nor do we have a proof of intractability for this problem. Finally, for the case where all three kinds of constraints are allowed, we give an efficient branch and bound algorithm, and demonstrate it by analyzing the timing of an interface between a processor and a memory chip.

Other researchers have considered variations on the above problem. Borriello [1], for example, solves the problem with only linear constraints, using the shortest paths algorithm, as do Brzozowski, Gahlinger, and Mavvadat [2] and Mavaddat and Gahlinger [5, 3]. In his PhD thesis, Gahlinger also considers the problem with both min and max constraints, and provides a polynomial algorithm for bounding the separations in this case. The algorithm is unfortunately not correct in all cases, however Gahlinger does make the observation that min and max constraints are commonly required in constraining the onset and offset of valid data read from memory devices. This helps to motivate the solution of the problem, in spite of its inherent complexity (NP-complete).

Vanbekbergen has recently proposed an $O(n^2)$ algorithm for finding the separation between a particular pair of events, for systems with both *min* and *max* constraints. However, his algorithm is incorrect if there are *min* constraints (we prove the problem is NP-complete), and less efficient than our algorithm when there are only *max* constraints.

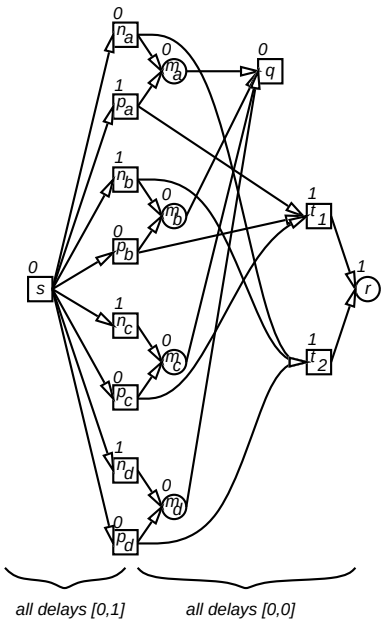

Figure 1: Example of reduction from 3SAT to min/max.

We note that since the feasible space of a system of constraints with min or max is not convex, well-known optimization techniques, such linear programming, max flow, or shortest path computations are not obviously applicable.

2 NP-completeness of *min/max* optimization.

The min/max constraint problem is easily seen to be NP-complete by reduction from 3-SAT. The graph of figure 1 depicts the min/max constraints corresponding to the 3-SAT formula $(a + b + c)(\overline{a} + \overline{b} + d)$. In the graph, the circles represent min events, the boxes represent max events, and the intervals on the edges represent the lower and upper bounds on the delay between events. The maximum separation s_{qr} equals 1 iff the formula is satisfiable (a choice of delays corresponds to a truth assignment). In the figure, numbers over the events indicate a solution of the timing constraints showing that the formula is satisfiable.

Consider a propositional formula f consisting of a conjunction of terms $f_1 \cdot f_2 \cdots f_n$, where each term f_i is a disjunction of three positive or negative literals $(x_i + y_i + z_i)$. Let the set of propositional variables be $a, b, c, \ldots$. We reduce the satisfiability of the formula f to a *min/max* problem. In this problem, there are events $p_a, p_b, p_c, \ldots$ and $n_a, n_b, n_c, \ldots$, corresponding to the positive and negative literals respectively, and subject to the following constraints, for all $v \in \{a, b, c, \ldots\}$:

$$t_{p_v} = t_s + \delta_{sp_v}, \text{ where } 0 \leq \delta_{sp_v} \leq 1$$

$$t_{n_v} = t_s + \delta_{sn_v}, \text{ where } 0 \leq \delta_{sn_v} \leq 1$$

The time of each event p_v or n_v ranges between 0 and 1, relative to the source event s. For each variable v, the min event m_v coincides with the earlier of p_v or n_v. Thus,

$$t_{m_v} = \min(t_{p_v}, t_{n_v})$$

There is also a max event q which coincides with the latest of the m_v events. Thus,

$$t_q = \max_{j \in \{a,b,c,\ldots\}} (t_{m_j}).$$

For each term f_i, there is a max event f_i satisfying the constraint

$$t_{f_i} = \max(t_{\phi(x_i)}, t_{\phi(y_i)}, t_{\phi(z_i)})$$

where $\phi(v) = p_v$ and $\phi(\neg v) = n_v$. Finally, there is a min event r such that

$$r = \min_{1 \leq j \leq n} (t_{f_j}).$$

The separation from q to r is 1 exactly when the formula f is satisfiable. To see this, first assume that β is a satisfying assignment of f. Then, if $\beta(v) = 1$, let $p_v = 1$ and $n_v = 0$, otherwise, let $p_v = 0$ and $n_v = 1$. It follows that $q = 0$ and $r = 1$. Second, assume that $q = 0$ and $r = 1$ (which must be the case if the separation $s_{qr} = 1$.) Since $q = 0$, at least one delay from each pair p_v, n_v must be zero. If $n_v = 0$, let $\beta(v) = 1$, else if $p_v = 0$, let $\beta(v) = 0$. β is a satisfying assignment of the formula f.

This reduction shows that the *min/max* constraint problem is NP-hard. The problem can be solved in NP time by the following procedure. For each min or max, guess nondeterministically which of the arguments is the actual minimum or maximum. This can be done in linear time, and reduces the problem to a system of linear constraints, which can be solved in polynomial time using a shortest paths algorithm. Therefore, the *min/max* constraint problem is NP-complete.

3 The acyclic max-only constraint problem.

Here we consider computing the separations for a system of constraints of the form of equations 3 and 2

only. Such a system can be represented as a graph in figure 1. In the graph, the boxes represent max events, and the intervals on the edges represent the lower and upper bounds on the delay between events. The graph can be regarded as a *interval-bounded* PERT chart (which we call an *IPERT chart*), in which activity times are unknown but bounded above and below by constants.

This *max*-only constraint problem can be solved efficiently, provided the IPERT chart is finite, acyclic, connected, and has a single source event (which we name 0). The set of predecessors of an event i is written preds(i). For describing the complexity of the algorithms, we denote the number of events (vertices) by n, and the number of edges by e. From the structure of the graph, we know that $n-1 < e < n(n-1)/2$.

An IPERT chart can be thought of as summarizing a family of PERT charts, each of which can be generated by choosing an an actual activity time within the bounds associated with each edge. (i.e., choosing a value for each δ_{ij} in the constraint system). Each PERT chart is called a *delay choice*, and it determines the time of occurrence of each event in the graph. The sets of event times resulting from delay choices are exactly the solutions to the original constraint system.

Vanbekbergen has recently proposed a simple, recursive algorithm for computing separations [7], which is claimed to find the maximum separation between a single pair of events in $O(n^2)$ steps. In fact, this algorithm can exhibit $O(n^3)$ behavior on some graphs. We had previously discovered an (unpublished) $O(n^3)$ algorithm for computing the separations for *all* event pairs. The algorithm presented here combines ideas from both algorithms to compute the separations s_{ij} between a fixed i and all other j in $O(e)$ time[1], and can compute separations between *all* pairs of events in $O(n \cdot e)$ time.

The main algorithm is a recursive depth-first search procedure that calls a second procedure for finding shortest paths. To understand it, it is helpful to define a generalized notion of critical paths: given a particular choice of event times, a path in the graph is a *critical path from i to j* if the sum of the delays along the path is equal to $t_j - t_i$. It is easy to see that for every transition i, there is at least one critical path from 0 to i. However, there is a critical path from i to j iff there is a critical path from 0 to j on which i appears.

The following observation explains the relevance of shortest paths to the algorithm. Suppose there is a path from event j to i and that the sum of the l-bounds along the path is c. Then, for every delay choice, the total delay of that path cannot be less than c. Hence, $\min(t_i - t_j) \geq c$, and therefore $s_{ij} = \max(t_j - t_i) \leq -c$.

The separations algorithm needs to know the minimum value of $-c$ above for all paths from j to i. Given events i and j, the following algorithm computes the shortest path from j to i in the PERT chart, using the negated l-bounds of the edges (succs(i) is defined to be $\{j \mid i \in \text{preds}(j)\}$). The algorithm is just the standard algorithm for solving the single-source shortest-paths problem on an acyclic graph; in this case, the graph is the reverse of the PERT chart.

$P(i,j)$:
  if p_{ij} *has not already been defined*
  then
    if $i = j$
    then $p_{ii} := 0$;
    else $p_{ij} := \min_{k \in \text{succs}(j)}(P(i,k) - l_{jk})$;
    endif
  endif
  return p_{ij};

The recursive function $S(i,j)$ for computing the exact separations is based on the additional observation that we can maximize $t_j - t_i$ by maximizing $t_k - t_i$ for some predecessor k of j and then maximizing the delay along the edge from k to j. We can also minimize the delays on all edges that are on some path from j to i, since this act cannot reduce t_i and cannot increase t_j. There are two cases to consider: either there is no critical path from j to i, in which case $t_j - t_i = t_k - t_i + u_{kj} = s_{ik} + u_{kj}$ (we have maximized $t_k - t_i$ so it is s_{ik}); or there is a critical path from j to i, along which all delays have been set to their l-bounds, so $t_j - t_i = P(i,j)$. In fact, the correct value for the separation is the minimum of these two values.

$S(i,j)$:
  if s_{ij} *has not already been defined*
  then
    if $i = j$
    then $s_{ii} := 0$
    else $s_{ij} :=$
      $\min(\max_{k \in \text{preds}(i)}(S(i,k) + u_{kj}), P(i,j))$;
    endif
  endif
  return s_{ij};

This theorem asserts the correctness of the algorithm:

Theorem 1 *For all i and j and for every real value c satisfying $-s_{ji} \leq c \leq s_{ij}$, there exists a solution to the constraints such that that $t_j - t_i = c$.*

[1] We are ignoring the cost of initializing matrices.

The proof that every delay choice yields a separation between $-s_{ji}$ and s_{ij} follows the reasoning above. The proof that it covers *every* point between the bounds consists of proving that the two separations at the end-points can be achieved, then invoking the intermediate value theorem.

The first argument to both S and P remains unchanged in all recursive calls. For each i, the control structure of S is a recursive depth-first traversal of the vertices preceding j in the graph, so S will call itself recursively at most e times. P calls itself at most e times for each value of i, and S calls P at most e times, so the total complexity of S is $O(e)$. The complete set of pairwise separations can be computed in time $O(n \cdot e)$ by calling $S(i, j)$ for all i and for those j that have no successors in the graph.

The theorem and proof above do not apply directly to open intervals, but they can be generalized easily. This also allows the use of infinite upper bounds for constraining edge delays to be finite but unbounded.

Applications

To check interface timing, whenever there is a timing requirement of the form $t_j - t_i \leq c_{ij}$, we compute s_{ij} and check that $s_{ij} \leq c_{ij}$. If it is true, the timing requirement will always be met, otherwise, there is a choice of delays that can violate it.

Another application suggested by Hung, Meng, Vanbekbergen, and Myers [4, 6, 7], is the identification of *redundant max* constraints, which can be removed without affecting the possible timings of events. An edge from i to j is redundant iff the minimum separation $-s_{ij}$ is greater than the maximum delay that can be placed on the edge, u_{ij}. A problem that remains to be solved is an efficient algorithm for finding all redundant constraints in a *timed signal transition graph* (STG), which is can be thought of as an infinite PERT chart that has been "rolled up" into a finite graph. Efficient safe approximations which find a subset of the redundant edges can be computed by unrolling the STG a finite number of times and then analyzing it with the method described above [6].

4 Generalized *max*-only constraint problem.

We now consider the addition constraints of the form of inequation 4 ($t_j - t_i \leq s_{ij}$) to the *max*-only constraint problem (note that s_{ii} must be 0). We have not been able to find a polynomial-time algorithm for finding maximum separations with these constraints.

A system of constraints of of the form of equations 2, 3, and 4 is said to be *tight* if the following inequalities hold

1. for all i, j, k, $s_{ij} \leq s_{ik} + s_{kj}$ and
2. for all $i, j \in \text{preds}(i)$, $s_{ji} \leq -l_{ij}$ and
3. for all i, j, $s_{ij} \leq \max_{k \in \text{preds}(j)}(s_{ik} + u_{kj})$.

If the system is tight, then all of the maximum separations s_{ij} are achievable (the proof is omitted because of lack of space).

A system of constraints which is not tight can be tightened by the following procedure.

step 1: for all i, j, let $s_{ij} = \min(s_{ij}, -l_{ji})$
repeat
  step 2: for all i, j, k, let $s_{ij} = \min(s_{ij}, s_{ik} + s_{kj})$
  step 3: for all i, j,
    let $s_{ij} = \min(s_{ij}, \max_{k \in \text{preds}(j)}(s_{ik} + u_{kj}))$
until
  cond 1: for some i, $s_{ii} < 0$, or
  cond 2: no change.

We can easily verify that steps 1, 2 and 3 leave the feasible space unchanged. If the procedure terminates under condition 1, the system is infeasible. If it terminates under condition 2, the system is tight, and hence all the maximum separations are achievable.

There is an upper bound on the running time of $O(n^3 \Sigma_{ij} s_{ij})$, because the procedure terminates in one iteration of the loop under condition 2 whenever $s_{ij} + s_{ji} < 0$. Every iteration of the loop decreases at least one separation by at least 1, since otherwise the procedure would terminate under condition 1. Hence the maximum number of iterations is the sum of all the initial separations $\Sigma_{ij} s_{ij}$. Since each iteration of the loop is $O(n^3)$, the procedure is bounded by $O(n^3 \Sigma s_{ij})$. The procedure is thus pseudo-polynomial. However, there are examples for which the algorithm requires a number of steps proportional to largest constant appearing in any of the interval edge labels.

5 Branch and bound algorithm.

Here we propose a heuristic, branch and bound solution to the *min/max* constraint problem. We can eliminate each *min* constraint by splitting the problem into several subproblems, which are generated by adding the assumption that one of the arguments to

the *min* is less than the others. By doing this recursively, we can eliminate all the *min* constraints, leaving an instance of the generalized *max*-only problem. The actual separations are the maxima of the reported separations for all subproblems.

A system with the min constraint

$$t_j = \min_{i \in \text{preds}(j)} (t_i + \delta_{ij})$$

yields $|\text{preds}(j)|$ subproblems. In each subproblem, we choose some $k \in \text{preds}(j)$ and add the assumption that $t_k + \delta_{kj}$ is the minimum operand. This assumption can be written in the appropriate form by replacing the min constraint with the equation $t_j = t_k + \delta_{kj}$ and the inequalities $t_k \leq t_i + u_{kj}$ for all $i \in S_j$ (these are all trivial max constraints).

Clearly, every solution to a subproblem is also a solution to the original problem, and every solution to the original problem is a solution to some subproblem, so this method is correct. It also terminates, since each recursive step eliminates one of the finitely-many min constraints.

A reasonable heuristic for which min constraint to eliminate, would be to choose the one that generates the fewest cases. There is no need to consider a case if the resulting system of linear inequalities is infeasible, a condition that can be detected in $O(n^3)$ time, which can be taken into account by the heuristic.

Example

Here, we take an example from Gahlinger [3], of an Intel 8086 CPU, conected via an address decoder and an address latch to an Intel 2716 EPROM (erasable programmable read only memory). In this example, the beginning of valid data asserted by the EPROM is a max event, since it depends on both the assertion of the address and the assertion of the read strobe signal. Conversely, the end of valid data asserted by the EPROM is a min event, since removing either address or read strobe will cause the data to become invalid. Unfortunately, because of a misunderstanding of the function of the address latch in the circuit, it was modeled as a simple bounded delay. In fact, the function of this latch is to hold the address stable during the entire memory cycle. As a result of this error, it was found in [3] that the timing requirements are violated, since the address does not remain asserted long enough. Nevertheless, we will use this model as an example, since it is the only basis of comparison for the two methods. In this case, we find that the exact separations produced by the algorithm of section 5 are the same as those produced by Gahlinger's program.

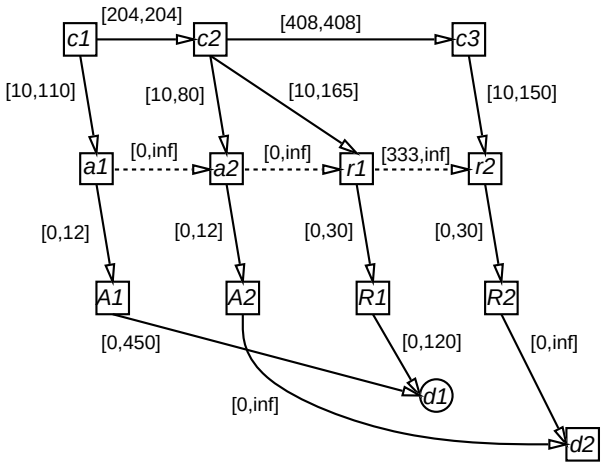

Figure 2: Timing constraints for 8086/2716 read cycle

Figure 2 depicts the timing constraints, for a read cycle. The events $c1, c2, c3$ are clock transitions, $a1$ and $a2$ are the beginning and end of valid address of the data/address bus, $A1$ and $A2$ are the beginning and end of valid address at the address latch outputs, $r1$ and $r2$ are the beginning and end of the read strobe, $R1$ and $R2$ are the beginning and end of the read strobe output by the address decoder, and finally $d1$ and $d2$ are the beginning and end of valid data on the data/address bus. Note that since there is only one early firing event $d2$, the heuristic for choosing which *min* constraint to eliminate is not an issue in this example. There are some infinite bounds in this example, which happen not to cause a problem with the convergence of the generalized *max*-only constraint algorithm,

Table 1 shows the required separations, the actual separations computed by the branch and bound algorithm, and the results of Gahlinger's program. The results of the two methods are the same.

j	i	$[-s_{ji}, s_{ij}]$ required	actual	Gahlinger
$A1$	$A2$	$[0, \infty]$	$[92, 286]$	$[92, 286]$
$a2$	$d1$	$[0, \infty]$	$[0, 358]$	$[0, 358]$
$R1$	$R2$	$[0, \infty]$	$[303, 578]$	$[303, 578]$
$d1$	$c3$	$[30, \infty]$	$[40, 398]$	$[40, 398]$
$d1$	$d2$	$[0, \infty]$	$[0, \infty]$	$[0, \infty]$
$c3$	$d2$	$[10, \infty]$	$[-398, \infty]$	$[-398, \infty]$

Table 1: Required and computed separations

References

[1] Gaetano Borriello. A new interface specification methodology and its application to transducer synthesis. Report UCB/CSD 88/430, Computer Science Division (EECS), University of California, Berkeley, 1988. Ph.D. thesis.

[2] J. A. Brzozowski, T. Gahlinger, and F. Mavaddat. Consistency and satisfiability of waveform timing specifications. *Networks*, Jan 1991.

[3] Tony Gahlinger. Coherence and satisfiability of waveform timing specifications. Research Report CS-90-11, University of Waterloo, 1990. Ph.D. thesis.

[4] Andy Hung and Teresa H.-Y. Meng. Asynchronous self-timed circuit synthesis with timing constraints. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, May 1990.

[5] F. Mavaddat and T. Gahlinger. On deducing tight bounds from partial timing specifications. Research Report CS-89-66, University of Waterloo, 1989. Ph.D. thesis.

[6] Chris Myers and Teresa H.-Y. Meng. Synthesis of timed asynchronous circuits, 1992. Submitted to this conference.

[7] P. Vanbekbergen, G. Goossens, and H. De Man. Specification and analysis of timing constraints in signal transition graphs. In *European DAC*, 1992. To appear.