Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

Improved Algorithms for Reasoning about Qualitative Temporal Constraint Problems

Hsien-Cheng Lin[1,2] and Ching-Chi Hsu[1]

[1] Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan 10617, R.O.C
CCHsu@csie.ntu.edu.tw
Fax:886-2-3628167 Tel:886-2-3917406

[2] Institute for Information Industry
Taipei, Taiwan 10617, R.O.C.
cclin@adc.iii.org.tw
Fax:886-2-5453950 Tel:886-2-7189123

Abstract

Representing and reasoning about incomplete and indefinite qualitative temporal information is an essential part of many artificial intelligence tasks. An Interval Algebra (IA) has been proposed by Allen as a model for representing qualitative temporal information about the relationships between pairs of intervals. In this paper, we present some algorithms to improve the tasks of reasoning with IA. First, we have improved Allen's path consistency algorithm by eliminating duplicate computations and by applying tightest constraint first heuristic that leads to faster convergence. Second, for finding a consistent scenario, we show that a backtrack algorithm with backward checking and simple backjumping techniques, yields better performance than old one. By computational experiments, we test the performance of our algorithms on a set of randomly generated consistent networks. The results show that the overall average execution times are better in our method.

1. Introduction

Representing and reasoning about incomplete and indefinite qualitative temporal information is an essential part of many artificial intelligence tasks. An interval algebra proposed by Allen [1] has been used as a model for representing qualitative temporal information about the relationships between pairs of intervals. It has been applied to solve many AI problems including planning [2,3], plan recognition [4], story understanding [5,6], and scheduling[7]. In this framework, the representation of temporal information can be viewed as binary constraint network [10,11,12,19,20,21] and constraint satisfaction techniques [13,14,15,16,18] can be used to reason about the information. Given possibly indefinite and incomplete knowledge of the relations between some intervals, the fundamental reasoning tasks are: (1) find a scenario that is consistent with the information provided, and (2) find the feasible relations between all pairs of intervals. However, the reasoning tasks have been shown to be NP-complete [8,9].

The reasoning tasks above are often solved by path consistency and backtracking algorithms [1,17]. In this paper, we present some improved algorithms for reasoning with Allen's Interval Algebra. First, we have improved Allen's path consistency algorithm by eliminating duplicate computations and by applying tightest constraint first heuristic that leads to faster convergence. Second, for finding a consistent scenario, we show that a backtrack algorithm that search through basic relations and prunes the search space with backward checking and simple backjumping techniques, yields better performance than old one. By computational experiments, the performance of the proposed algorithms are evaluated on a set of randomly generated consistent networks.

2. Background

Allen's interval algebra (IA) uses intervals as temporal objects. Each interval represents a period over which an event occurs. An interval, I, is an ordered pair of a beginning point and an ending point, $[I^-,I^+]$,where I^-,I^+ belong to real number. There are 13 basic relations (see Figure 1) specifying all possible relationships between any two intervals. To be able to represent indefinite information over pairs of intervals, the algebra allows disjunctions of these basic relations. There are 8192 (2^{13}) relations definable from the 13 basic relations. The empty set (containing no basic relation) denotes inconsistency. Universal set (containing all 13 basic relations) means no constraint over the

two corresponding intervals. IA is the algebra with the underlying 8192 sets, unary operator inverse, and binary operators union ($\cup$), intersection($\cap$), and composition($\circ$).

The binary operators, union and intersection, are defined as normal set operators. Let $R_{ij} = \{r_1,...,r_k\}$ be a set of basic relations constraining over two intervals, X_i to X_j. The inverse operation can be defined as, $R_{ij}^{-1} = \{r_1^{-1},...,r_k^{-1}\}$, where r_k^{-1} is the inverse relation of r_k . The composition of R_{ik} and R_{kj} denoted $R_{ik} \circ R_{kj}$ admits only basic relations for which there exists $p \in R_{ik}$ and $q \in R_{kj}$ such that when the intervals X_i and X_k are related by the basic relation p, and the intervals X_k and X_j are related by the basic relation q, then it is possible that Xi and Xj can be related by r. Allen defined the composition operator between two basic relations by using a composition table. Therefore, $R_{ik} \circ R_{kj}$ can be defined as follows:

$$\{p \circ q \mid p \in R_{ik}, q \in R_{kj}\}.$$

An IA network is a binary constraint satisfaction problem where the variables represent intervals. A binary relation between interval X_i and X_j is defined by a disjunction of the basic relations, R_{ij}. An IA network, R, can also be represented by a labeled directed graph where nodes represent intervals, and an arc i->j is labeled by the relation R_{ij}. Each relation, R_{ij}, implies an inverse relation R_{ji}. Only one of them is shown in the labeled graph. An IA network, R, of n nodes, can be defined by its $e = n(n-1)/2$ labeled arcs. If we assign arc numbers to these arcs such that the arc number k corresponds to the directed arc (i,j), the IA network, R, can be presented as a tuple, $(R_1,...,R_e)$.

A scenario is a tuple $S = (r_1,...,r_e)$, where $r_k \in R_k$, $1 \le k \le e$. A consistent scenario,S, is a scenario such that all basic relation in S hold. The IA network is consistent if at least one consistent scenario exists.

3. An improved path consistency algorithm

Since IA networks are node consistent and arc consistent, path consistency is the lowest order of consistency with acceptable order of complexity $O(n^3)$. Path consistency algorithm can also be applied prior to a backtrack algorithm to prune the search space.

3.1 Design concepts of the improved algorithm

Allen's [1] path consistency algorithm uses a first-in first-out queue. Initially all arcs of constrained labels are placed into the queue. The propagation of relations is initiated by removing an arc, say (i,j) from the queue, then the label Rij is used to constrain other labels in the network. When any label is updated, its corresponding arc is placed into the queue, since this label might further constrain other labels. The propagation continues in this fashion until the queue is empty, indicating that a fixed point has been reached.

We propose two improvements to Allen's path consistency algorithm. The first improvement is based on the observation that the arcs on the queue should not be duplicated. The repeated propagation of the same relation costs the unnecessary amount of work. We can replace the "queue" by maintaining a "set" of arcs using a $n \times n$ matrix instead. The entry (i,j) in this matrix has the value NULL if it is not in the set. Each time we want to place an arc (i,j) in the set, we check whether the entry (i,j) of the matrix has the value NULL or not. Only if it is NULL is it necessary to place that arc (i,j) in the set. Each time we get an arc, say (i,j), from the set, we need to reset the entry (i,j) of the matrix to NULL.

The second improvement is done by applying tightest constraint first strategy when selecting a tuple from the set to process. We observe that it make the algorithm converge faster. This strategy can be easily implemented by using an array of linked list. To arrange the arcs in an ascending order of the number of basic relations on the labels, we can maintain 12 doubly-linked lists. Each designated by the number of basic relations (from 1 to 12). It is unnecessary to keep label of universal relation because it has no effect in constraining other labels. If a tuple (i,j) is in a list, the entry (i,j) of the matrix above contains a pointer pointing to the element in the list. To get an entry (i,j) from the lists, we get it from the least-number nonempty list and set the entry (i,j) on the matrix to NULL. To add an entry (i,j) , we first computes the number of basic relations on the label, then the entry (i,j) is added on the corresponding list indexed by the number of basic relations. We check if the entry (i,j) on the matrix is not NULL; this means it points to an element on a list. We then use this pointer to delete the pointed element from the list (to avoid duplication), and set the entry on the matrix with a pointer pointing to the new element we insert previously.

After making the two improvements, the space complexity of this algorithm is $O(n^2)$ and its worst-case time complexity is still the same as Allen's one, $O(n^3)$; However, its execution time is improved as shown empirically in next section.

3.2 Empirical results

We generate random interval constraint networks with the following properties:

- Each basic relation within an arc occurs with the same probability.
- Each arc within the network occurs with the same probability.

We generate 100 networks of sizes 5 to 100. Figure 3 shows the percentage of networks of each size that are path-consistent. The graph shows a sharp jump from networks of size 14 to networks of size 25. We found that all general networks that we randomly generated with more than 25 variables are path inconsistent

With the random generation method we did not obtain any large path-consistent random networks of size $n \geq 30$. This is because in the larger networks, more label compositions are intersected, which reduces the number of possible relations between two intervals. Hence, the probability of two relations being path-consistent is low. Since we do not know of any method to generate truly random large path-consistent networks, we generated consistent networks in the following way: Pick a number of intervals lying on a metric scale $T_{min},...T_{max}$, of discrete time instances and compute the relations that hold between each interval pair. This gives a set of initial relations that, taken together, form an initial solution. Then produce a final network by taking the sum of the initial relations and some other randomly chosen labels. The generated networks are consistent with at least one solution: the initial solution. The larger the time scale, the less likely it is for two intervals to have one point in common. The basic relations occur with different probabilities.

Now, some empirical results are given to show the performance of the proposed path consistency algorithm compared to Allen's original algorithm. We compare their total execution time to count the overhead of the algorithms. We will present results of some computational experiments on three path consistency algorithms: Allen's original algorithm (denoted by *Allen's PC*), the algorithm that avoids the duplicated pairs (denoted by *Duplicates removed*), and algorithm that adopts both avoiding duplication and the tightest constraint first strategy (denoted by *Tightest first*).

The algorithms are tested on two sets of networks using Intel Pentium 75. In the first set, networks are randomly generated. In the second set, networks are generated and forced consistent as described above. For each network size, 100 networks are generated and tested. In the first set, we compute average execution time for path consistent networks and for path inconsistent networks separately. The experimental results are shown in Table 1. The three path consistent algorithms detect path inconsistent networks very quickly especially for tightest first one. The experimental results for the second set are shown in Figure 4. We found that the *tightest first algorithm* always gave the results faster than the other two algorithms, and that *duplicates removed* one is faster than *Allen's PC*.

4. An improved backtrack algorithm (NEW)

A straightforward way of solving an IA network is to decompose it into several scenarios, and solve each of them separately. The consistency of each scenario can be checked in $O(n^2)$, where n is the number of intervals. The original network is consistent if and only if there is at least one consistent scenario. And if we combine all consistent scenarios together, the minimal labeling network is obtained. The complexity of solving a general IA network by generating all the scenarios and checking for their consistencies independently is $O(n^2 13^e)$.

This brute-force enumeration process can be pruned significantly by running a backtrack search algorithm. Backtrack assigns a basic relation to an arc incrementally, as long as the partial assignment is consistent; otherwise, the algorithm backtracks. Allen proposed to use such backtrack algorithm (denoted as OLD) in which a consistent scenario is incrementally constructed through basic relation assignment. The best case complexity of the backtrack algorithm is $O(n^4)$, which occurs when no backtrack takes place on any of $O(n^2)$ arcs; each arc is assigned a label and the $O(n^2)$ consistency checking algorithm is executed. However, the worst-case complexity of the backtrack algorithm is $O(n^2 13^e)$.

When used with backward checking, backjumping can enhance the performance of a backtrack algorithm. Backjumping is the idea of going back several levels in a backtrack situation, rather than going back to the chronologically most recent decision.

4.1 Design concepts

Our proposed algorithm (denoted as NEW) assigns a basic relation to each arc, and traverses the arcs in a predetermined order. We assume the order of the arcs in an incrementally constructive fashion, namely, starting with a complete graph of m nodes and at each step

forming the complete graph of m+1 nodes by adding m arcs, one at a time. The algorithm assigns consistent basic relations to a subsequence $(r_1,...,r_k)$ of labels and attempts to append to it a new assignment of r_{k+1}. The algorithm backtracks to the most recent arc (in case of normal backtrack), or backtracks directly to the source of conflict (in case of backjumping), then changes its assignment and continues from there. Our NEW algorithm for finding one consistent scenario is given in Figure 5.

The algorithm assumes an input IA network $R=(R_1,...,R_e)$, and a current assignment $S=(r_1,...,r_k)$. The algorithm runs a path consistency algorithm as a preprocessing step to reduce the search space before a backtrack algorithm starts, sets the Boolean variable *found* to FALSE, and then calling Backtrack with k = 0. The function Backward_Check$(r_1,...,r_k,R_{k+1})$ selects all basic relations of the label R_{k+1} that are consistent with the current assignment $(r_1,...,r_k)$. This is the backward checking step where we use the assignment of previous labels to filter and keep only the possible basic relations for the next arc. Each basic relation in the result of Backward_Check is guaranteed to be consistent with the previous assignments.

The subnetwork is incrementally built toward a complete graph. To filter the relations for the (k+1)th arc, we can consider only m-2 triangles, since the first k arcs are labeled with basic relations, and they are already consistent with each other.

Suppose that the IA network, R, has nodes (1,...,n); then we assume that the order of arcs, k, is related to its corresponding pair of nodes (i,j) as follows:

$$k <=> (j-1)(j-2)/2 + i,\ 1 \le i < j \le n$$

The function Backward_Check does the backward checking step for pruning and uses a simple backjumping for identifying the first and farthest source (arc) of conflict as an index for backjumping when inconsistency occurs. We can consider only all triangles such that the arc R_{k+1} is one of their sides. If the subnetwork built so far has m nodes, then there are only m - 2 triangles to be checked. The relations on these m -2 triangles prune all the inconsistent basic relations in R_{k+1} by the way of backward checking. Either R_{k+1} becomes empty, indicating inconsistency, or R_{k+1} gets smaller containing only those consistent with the previous assignments. Assume that k and (i,j) are related according to equation (1). Given (i,j), we can consider all triangles (i,j,p) where $p \neq i$, $p \neq j$, and $1 \le p < j$.

The time complexity of Backward_Check is O(m) (for loops run at most m-2 times), where m is the number of nodes in the subnetwork that consists of the arc 1,...,k+1. Each level averagely takes n/2 composition operations and there are n(n-1)/2 levels, so in the best case (no backtracking), the time complexity is $O(n^3)$.

4.2 Empirical results

We compare the performance of our backtrack algorithm, NEW, to that of the OLD on a set of randomly generated consistent IA networks. For each network of size n, we generate 25 consistent IA network. The times reported are in seconds for Intel Pentium 75. The experimental results are shown in Table 2 The numbers in parentheses of Table 2show the number of cases (out of 25 tests cases) when the algorithms did not find solutions within 25 minutes of CPU time. No timing information is reported for these cases; hence they are not included in the average execution time. We conducted two set of test cases: In Table 2(a), the tightness of the generated consistent networks is set to 0.1, and the connectivity of them is set to 0.75. In Table 2(b), the tightness of the generated consistent networks is set to 0.5, and the connectivity of them is set to 0.5. For example, in Table 2(a) where n = 25, the average execution time for our algorithm NEW was 127.66 seconds: this is an average for over 15 cases where 10 cases run out of 25 minutes time bound. The OLD algorithm gave 6 answers with a 291.71 seconds average, and 19 cases were left unanswered. The average time run for path consistency preprocessing was 0.08 seconds, which is negligible when compared to the time for backtrack search step. But the effect of path consistency preprocessing for pruning search space is significant. The number of out-of-bound cases is another important factor to measure the performance of the algorithms. The lower the number, the more solutions the algorithm gave within the limited time bound.

If we take the average over all 300 tested networks in Table 2(a), the average execution time of algorithms OLD and NEW became 37.4 and 31.02 seconds, where out-of-bound cases are 196 and 142, respectively. By using these figures, the average execution time of NEW was an improvement of about 21% over OLD; the numbers of out-of-bound cases were reduced by about 28%. These results, however, should be qualified in at least two ways. First, the statistics are valid only relative to the parameters of tightness and connectivity. Note that in Table 2(b), tested cases of size n > 20 were almost unanswered within the time bound. Second, the

variances of execution times for each problem size were quite different from their average times owing to random generation. Although, the algorithms found solutions very quickly (within 5 minutes), they did not solve the problems at all within the 25-minute bound in most cases, respectively, 65% (OLD) and 47% (NEW) of the total.

5. Conclusions

In this paper, we present some improved algorithms for reasoning with Allen's interval algebra. We have improved Allen's path consistency algorithm by eliminating duplicate computations and by applying tightest constraint first heuristic that reduces the total number of computations. We also propose a new backtrack algorithm that search through basic relations and prunes the search space with backward checking and simple backjumping techniques, yields better performance than old one.

References

[1] J.F. Allen, "Maintaining knowledge about temporal intervals", Comm. ACM, 26, 1983, pp.832-843.

[2] J.F. Allen, J.A. Koomen, "Planning using a temporal world model", IJCAI-83, pp.741-747.

[3] J.F. Allen, "Planning as temporal reasoning ", 2nd Int. Conf. Principles of knowledge representation and Reasoning, 1991, pp.3-14.

[4] F. Song, R. Cohen, "Temporal reasoning during plan recognition", Proc. 9th NCAI, 1991, pp.247-252.

[5] J.F. Allen, "Towards a general theory of action and time", J. of AI, 26(2), 1984, pp.123-154.

[6] F. Song, R. Cohen, "The interpretation of temporal relations in narrative", Proc. 7th NCAI, 1988, pp.745-750.

[7] A.C. Meng, B.A. Raja, "Logos-tes: An expert system for operation management based on temporal constraint satisfication", Proc. 6th Coonf. Artif. Intell. Appl., 1990, pp.215-221.

[8] M. Vilain, H.A. Kautz, "Constraint propagation algorithms for temporal reasoning", AAAI-86, 1986, pp.377-382.

[9] M. Vilain, H.A. Kautz, P.van Beek, " Constraint propagation algorithms for temporal reasoning", revised version of AAAI-86, pp.377-382, 1986, from Readings on Qualitative Reasoning about Physical Systems (1990), pp.373-381.

[10] R. Dechter, I. Meiri, J. Pearl, "Temporal constraint networks", J. of AI, 49, 1991, pp.61-95.

[11] A.K. Mackworth, "Consistency in networks of relations", J. of AI, 8, 1977, pp.99-118.

[12] E.C. Freuder, "Synthesizing constraint expressions", Comm. ACM, 21, 1978, pp.958-966.

[13] A.K. Mackworth, E.C. Freuder, "The complexity of some polynomial network consistency algorithms for constraint satisfication", J. of AI, 41, 1989, pp.89-95.

[14] B.A. Nadel, "Tree search and arc consistency in constraint satisfication algorithms", Search in Artificial Intelligence, edited by L. Kanal and V. Kumar, Springer-Verlag, 1988, pp.287-342.

[15] R.M. Haralick, G.L. Elliott, "Increasing tree search efficiency for constraint satisfication problem", J. of AI, 14, 1980, pp.263-313.

[16] R. Dechter, "Enhancement schemes for constraint processing: backjumping, learning, and cutset decomposition", J. of AI, 41, 1989, pp. 273-312.

[17] A. Reinefeld and P.B. Ladkin, "Fast solution of large interval constraint networks", Proc. Ninth Biennnial Conf. of the Canadian Society for Computational Studies of Intelligence, Vancouver, BC (1992).

[18] R. Dechter, J. Pearl, "Network-based heuristics for constraint satisfication problems", J. of AI, 34, 1988, pp.1-38, and also in Search in Artificial Intelligence, edited by L. Kanal and V. Kumar, Springer-Verlag, 1988, pp.370-425.

[19] P.B. Ladkin, R. Maddux, "On binary constraint networks", Technique Report KES.U.88.8, Kestrel Institute, Palo Alto, Calif., 1988.

[20] I. Meiri, "Combining qualitative and quantitative constraints in temporal reasoning, " in Proc. Ninth Natl. Conf. Artif. Intell., 1991, pp.260-267.

[21] H. Kautz and P.B. Ladkin, "Integrating metric and qualitative temporal reasoning, " in Proc. Ninth Natl. Conf. Artif. Intell., 1991, pp.239-246.

Relation	Illustration	Symbol	Inverse
x **before** y	x|---| |-----|y	b	bi
x **meets** y	x|---| |----|y	m	mi
x **during** y	x|---| |----------|y	d	di
x **overlaps** y	x|------| |--------|y	o	oi
x **starts** y	x|-----| |---------|y	s	si
x **finishes** y	x|---| |-------|y	f	fi
equals y	x|----| y|----|	eq	eq

Figure 1. Basic relations between two intervals .

```
function Revise(i,k,j)
{ Z <- R_ij ∩ (R_ik ° R_kj);
 if (Z == ∅) {report inconsistent network; exit;}
 if (Z == R_ij ) return FALSE;
       else { R_ij <- Z; R_ji <- Z^-1; return TRUE;}
}

procedure Path_Consistency()
{ Q_1<- {(i,j)| 1 ≤ i < j ≤ n}; // l is the number of
                                // basic relations of (i,j)
  while Q is not empty
  { select and delete a (i,j) with tightest constraint
    from Q;
    for k <- 1 to n  do
      if (k <> i and k <> j)
       { if Revise(i,j,k) Q_1 <- Q_1 ∪ (i,k);
           if Revise(k,i,j) Q_1 <- Q_1 ∪ (k,j);
       }
  }
}
```

Figure 2. Improved path consistency algorithm for interval algebra.

Figure 3. Consistent network distribution pattern

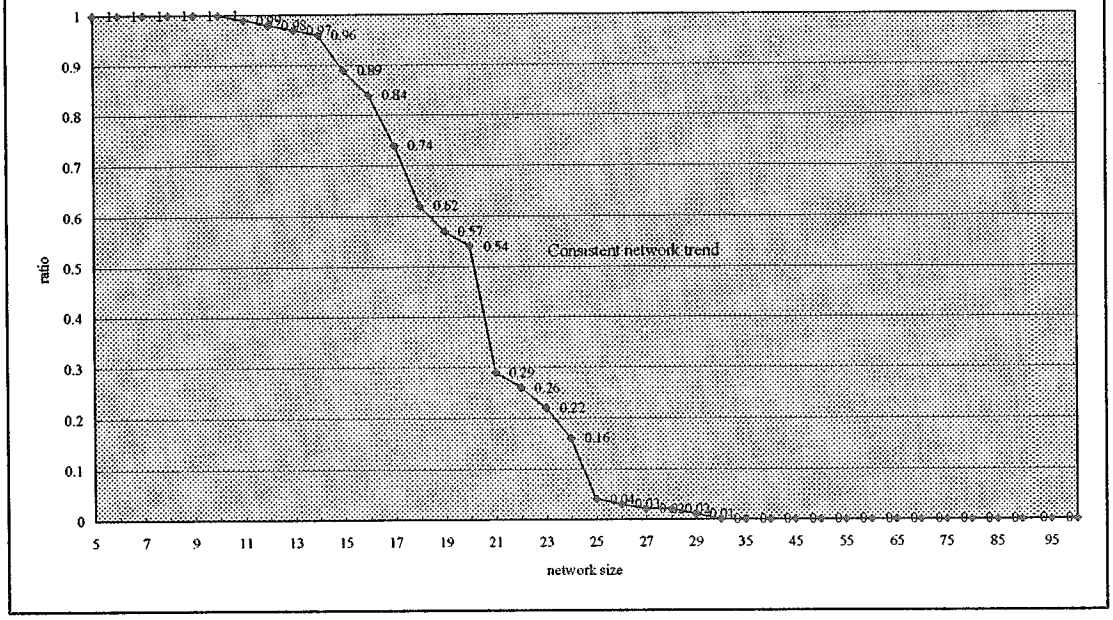

Table 1. Average execution time of path consistent algorithms for the randomly generated networks (first test case).

node size	ratio	for path consistent networks: tightest first	duplicate removed	Allen's PC	for path inconsistent networks: tightest first	duplicate removed	Allen's PC
10	0.99	0.000	0.000	0.000	0.000	0.000	0.000
15	0.89	0.055	0.055	0.055	0.000	0.000	0.000
20	0.54	0.165	0.165	0.220	0.000	0.110	0.165
25	0.04	0.329	0.384	0.495	0.000	0.165	0.165
30	0	N/A			0.000	0.165	0.165
40	0	N/A			0.000	0.220	0.220
50	0	N/A			0.000	0.275	0.275
60	0	N/A			0.000	0.330	0.330
70	0	N/A			0.000	0.440	0.440
80	0	N/A			0.000	0.440	0.440
90	0	N/A			0.000	0.604	0.604
100	0	N/A			0.000	0.704	0.704

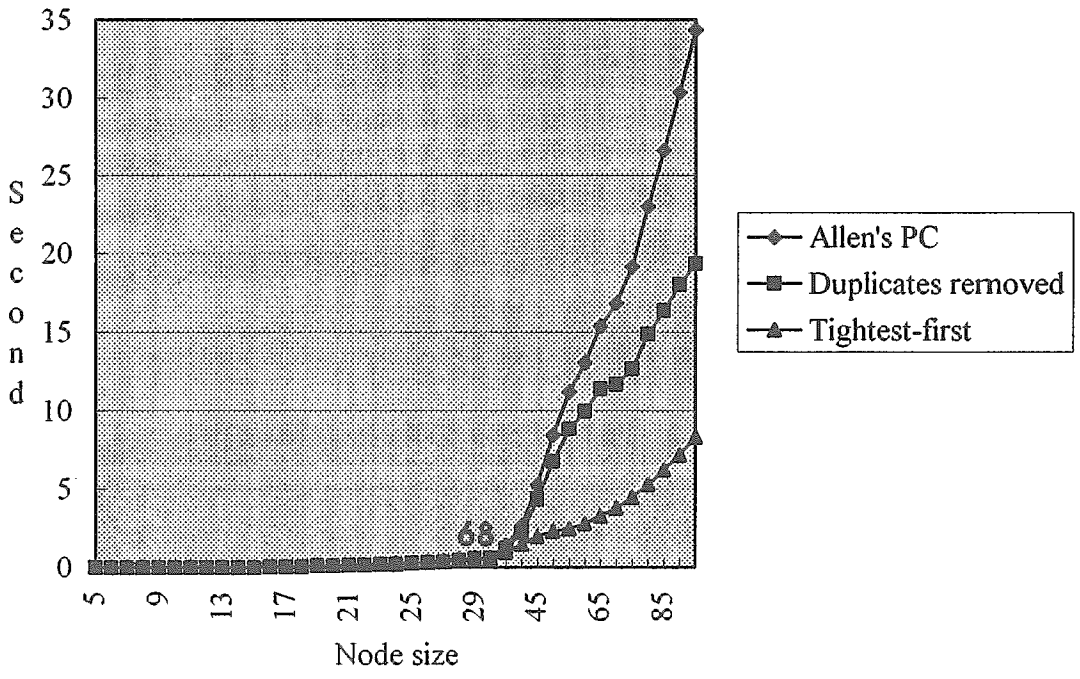

Figure 4. Path consistency processing time for the forced consistent networks (second test case) (Tightness=0.5,Connectivity=0.5,100 consistent networks for each node size).

```
procedure NEW()
{ run path consistency algorithm as in Figure 2.;
 found <- FALSE;
 Backtrack(0);
}
procedure Backtrack(k)
 {
  if (k==n(n-1)/2)
       {found <- TRUE;
        exit with current assignment;
       }
  T_{k+1} <- Backward_Check(r_1,...,r_k,R_{k+1});
  while ((not found) and (T_{k+1} is not empty)) do
  {r_{k+1} <- select a basic relation in T_{k+1};
   remove r_{k+1} from T_{k+1};
   Backtrack(k+1);
   if (not found) backjumping to the designated
k;
  }
}
function Backward_Check(r_1,...,r_k,R_{k+1})
{
 (i,j) <- changing the arc number k+1 into
 its corresponding pair of nodes by equation (1);
 for p <- i+1 to j-1 do
   {R_ij <- R_ij ∩ (r_ip ∘ R_pj);
    if (R_ij == ∅)
     {(i,p) gives the index k (the arc number)
           to jump back;
      return ∅;
     }
   }
for p <- 1 to i-1 do
 {R_ij <- R_ij ∩ (r_ip ∘ r_pj);
 if (R_ij == ∅)
 {(p,j) gives the index k (the arc number)
  to jump back;
  return ∅;
 }
 }
 return R_ij;
}
```

Figure 5. NEW backtrack algorithm for finding a consistent scenario.

Table 2 Average execution times (in seconds) of two backtrack algorithms. (Run on Intel Pentium 75, 25 cases for each size, limited time 25 mins)

(a) (for Connectivity=0.1, Tightness=0.7)

node size	time for path consistency	time for OLD	time for NEW
5	0.00	0.00(0)	0.00(0)
10	0.00	0.55(1)	0.05(0)
15	0.02	0.85(6)	0.21(1)
20	0.05	100.72(10)	0.13(3)
25	0.08	291.71(19)	27.66(10)
30	0.13	9.73(20)	1.45(11)
35	0.20	60.35(19)	96.45(12)
40	0.28	21.25(22)	111.61(15)
45	0.76	124.79(24)	42.23(18)
50	0.99	N/A	134.53(23)
55	1.43	N/A	23.37(24)
60	1.98	N/A	N/A

(b) (for Connectivity=0.5, Tightness=0.5)

node size	time for path consistency	time for OLD	time for NEW
5	0.00	0.00(0)	0.00(0)
10	0.03	16.63(1)	1.58(1)
15	0.12	231.25(19)	229.36(16)
20	0.34	10.61(24)	8.76(22)
25	0.67	N/A	45.41(24)
30	1.12	N/A	N/A