GENERATING, STORING AND USING THE COMPLETE COMPOSITION TABLE IN ALLEN'S TEMPORAL INTERVAL ALGEBRA

Troy Saulnier & André Trudel
Jodrey School of Computer Science
Acadia University
Wolfville, Nova Scotia, B4P 2R6, Canada
tsaulnie@dal.ca & andre.trudel@acadiau.ca

ABSTRACT
A fundamental problem in temporal reasoning is the composition of two subsets of Allen relations to produce a third subset. Past research concentrated on clever algorithms and heuristics. Our proposed approach pre-computes and caches all the results. This allows us to study properties of the results. We also experimentally compare our approach with previous algorithms and heuristics. Results show that for problems involving small networks, clever heuristics are useful. As network size increases, the cost of computing the heuristics becomes prohibitive. In this case, it is best to use our approach of a full composition table.

KEY WORDS
Temporal reasoning, Allen's 13 interval relations

1. Introduction

Allen [1] defines a temporal reasoning approach based on intervals and the 13 possible binary relations between them. The relations are before (b), meets (m), overlaps (o), during (d), starts (s), finishes (f), and equals (=). Each relation has an inverse. The inverse symbol for b is bi and similarly for the others: mi, oi, di, si, and fi. The inverse of equals is equals. We refer to the 13 relations as the basic labels.

A relation between two intervals is allowed to be a disjunction of the basic relations, which is represented as a set. For example, (A m B) V (A o B) is written as A {m,o} B. The relation between two intervals is allowed to be any subset of I = {b,bi,m,mi,o,oi,d,di,s,si,f,fi,=} including I itself.

An IA (interval algebra) network is a graph where each node represents an interval. Directed edges in the network are labeled with subsets of I. By convention, edges labeled with I are not shown. An IA network is consistent (or satisfiable) if each interval in the network can be mapped to a real interval such that all the constraints on the edges hold (i.e., one disjunct on each edge is true). A consistent scenario is a consistent instance of an IA network such that it has the same nodes and edges, and each edge has a singleton subset of the original label.

The standard algorithm for finding a consistent scenario of an IA network is to first pre-process the network with a path consistency algorithm, then apply a backtracking algorithm interleaved with the same path consistency algorithm. Allen [1] was the first to suggest this approach and van Beek and Manchak [2] describe an efficient implementation programmed in C (which we refer to as the "van Beek implementation"). Nebel [3] presents performance improvements to the backtracking algorithm for hard instances.

When finding a consistent scenario, which is NP-complete, the majority of the effort is expended doing path consistency. As noted in [2], the composition operation is the bottleneck in the path consistency algorithm. We thus focus our attention on the composition operation.

In the next section we describe the composition operation and two related algorithms. We propose an alternative method which involves pre-computing and storing all the solutions. The composition operation then involves doing a direct table look-up. We show that our method is faster for large networks. Our method also allows us to study the set of solutions.

2. Composition Operation

Let Cij be the label on the edge (i,j). The labels Cij and Cjk can potentially restrict the label Cik. For example, if Cij is "bi" and Cjk is "f", then Cik must be "bi". "bi" is the result of composing "bi" with "f".

Cij can be I or any of its subsets except for the empty set. There are (213-1) = 8191 possibilities. Since there are two labels involved, the composition operation must deal with (213-1) × (213-1) = 67,092,481 possibilities.

There are two standard algorithms for computing the result of a composition operation. The first is due to Allen

[1] which uses a 13 x 13 singleton set composition table and a nested for loop which requires 132 iterations in the worst case. The second method, due to Hogge [4] uses four tables. A series of bit-wise ANDs, ORs, and shift operators replace the nested loops in the composition function from Allen's method.

van Beek and Manchak [2] describe how to improve the two algorithms with heuristics. A skipping heuristic determines a priori that the result of the composition is I, which does not further restrict the third edge. Skipping occurs when one of the first two edge labels is I. Skipping also arises when the first edge contains {b} and the second edge contains {bi}, or the reverse, or the first edge contains {d} and the second edge contains {di}.

Other ordering heuristics are weight, cardinality and constraintedness. For the weight heuristic, the basic labels are ranked based on their ability to restrict, or produce a lower cardinality for the third edge in a composition operation. Calculating the weight for any given edge is a matter of summing the individual weights of its components. The cardinality heuristic is similar except that all basic relations contribute a weight of one to the overall cardinality. Finally, the constraintedness heuristic involves using the weights of the other two edges in a three-node sub-network to determine the effect on the third edge.

Skipping provides the best improvement - the Hogge implementation outperformed Allen - and of the ordering heuristics, only the weight heuristic surpassed the cost of computation and provided an improvement. Via experiments on relatively small networks, van Beek and Manchak [2] determine the order of performance from slowest to fastest to be: Allen, Allen+skip, Hogge, Hogge+skip, Hogge+skip+weight.

An alternative composition algorithm is proposed by Zhang and Zhang [5]. They represent the relationship between two intervals with a 5x5 matrix which contains only 0's and 1's. The composition operation involves a modified multiplication of two 5x5 matrices. There are problems with their approach. First, their theorem 1 on p. 12 is used to prove the correctness of their algorithm. Unfortunately, it is easy to generate a counter-example where the theorem is false: set interval K to be a subset of J, I disjoint from K, and I can either be a subset of J or disjoint from J. Another problem with their approach is that it is not clear if it scales up to large networks. In their largest experiment, they perform 30 composition operations. They compare their algorithm with Allen's. Both are implemented in Java. Unfortunately, they do not benchmark their work against van Beek's implementation. Because of these problems, we do not compare our work with [5].

3. Proposed Solution

van Beek and Manchak [2] suggest the possibility of using a $2^{13} \times 2^{13}$ full composition table, but dismiss the idea as impractical due to memory constraints. Since computing environments have changed considerably since 1996, we investigate this avenue of research. A full composition table allows for the direct lookup of the result of the composition operation.

We generate and save a full composition table which stores all the possible results of composing two edge label sets. This is represented as a matrix, where all label sets are enumerated across the top row and down the first column, then used as indices for the table. The result of composing the i'th and j'th set is stored in entry (i,j).

4. Implementation

The implementation and experiments (described later) were performed on an Intel x86 Machine (Dell 4300), with a 1.6 GHz Pentium 4 CPU, 512 MB of 266 MHz DDR SDRAM, and an 80 and 120 GB Hard Drive. This machine was running Linux Redhat 8.0, kernel version (2.4.18-17.8.0), gcc version (3.2), and DB2 version 8.1.

As proposed in [2], each subset of I is represented by an integer. Each basic relation is assigned a bit position. The bit is set to one if its associated relation is included in the subset, and zero otherwise. The problem now becomes one of efficiently storing and accessing a very large matrix of integers.

The C compiler did not allow the direct declaration of an array of 67 million integers. Instead, dynamic memory allocation is used. Indexing is set up to allow access to the memory as though it were a two-dimensional array. First 8191 contiguous pointers to integers are instantiated to be the rows of the table. Next, for each row, a loop creates a pointer to 8191 integers representing the columns. Since the columns are allocated contiguously and each row is indexed, any entry in the table can be accessed via double subscripts as in a two dimensional array. With the first row and column as the indexes, direct lookup of any composition result is possible.

The "eq" relation has the unique property that any label set composed with "eq" results in itself. Each subset of I, except for the empty set, is assigned an integer between 1 and 8191. I is assigned to 8191, and "eq" is strategically assigned to 1. The result is that the first row and column contain incrementing integer values from 1 to 8191 and can be used as indexes.

The table is read into a shared memory segment once and made available to other programs. The complete table is generated in less than 3 minutes.

5. Properties

One advantage of generating and storing the complete composition table in memory is that it can be studied. A summary of the properties found are presented in this section.

Cardinality: 60,826,089 or 91% of the entries in the composition table consist of I. The table is used during path consistency to hopefully constrain the label on a third edge. Since I entries do not constrain, most of the table is not useful. After I, entries of cardinality 12 occur the next most frequent. All basic relations are present in at least some label sets with a cardinality of 12. Each set with cardinality 12 is missing one relation. For example, there are 5,613 sets of size 12 in the complete composition table that do not contain "oi". Frequency decreases with cardinality. Singleton sets occur the least often. An anomaly occurs with sets of size 5. There are more sets of size 5 than of size 4 or 6.

Singleton sets: Singletons are label sets consisting of a basic temporal relation. Singletons are rare and account for only 0.002% of the entire table. Each of the 13 singleton sets occurs at least once. Only {eq} occurs exactly once in the table. It results from composing eq with eq. Of the singleton sets, {b} and {bi} occur the most often with 625 entries each. All the other singleton sets each occur 37 or fewer times. The count of the total number of occurrences for each singleton set appears in the second column of Table 1.

Coverage: Does each of the 213-1 possible label sets occur in the table? The trivial answer is yes because as explained in the previous section, each possible set occurs in the first row and column of the table. Recall from above that "eq" occurs in position (1,1) of the table and no where else. If we remove the first row and column, do all the possible label sets (except for {eq}) occur in the remainder of the table? Also recall that 91% of the entries are I. Surprisingly the answer is yes, all possible sets (except for {eq}) occur more than once outside the first row and column. The rarest sets are {s}, {si}, {f}, and {fi} which only each occur 3 times outside the first row and column (their total occurrence is 5 as indicated in the second column of Table 1).

Basic relations: Let us ignore I entries. Of the remaining entries, how often is {eq} a subset (i.e., how many non-I entries contain an "eq")? The answer is 75% of the entries. The percentages for each of the other basic labels are given in the second column of Table 1. Note that for the basic labels, "b" and "bi" occur the most often on their own (625 occurrences each), but occur the least often in non-I entries.

Table 1 - Basic label counts

	Number of occurrences of a basic label singleton set	Percentage of non-I entries that contain a basic label
eq	1	75%
b	625	68%
bi	625	68%
d	37	89%
di	37	89%
o	9	95%
oi	9	95%
m	14	79%
mi	14	79%
s	5	88%
si	5	88%
f	5	88%
fi	5	88%

Visualization: An image of the full composition table appears in Figure 1. Each entry is represented by a single pixel. I entries are shown in white, while non-I entries appear in black. The upper left corner is entry (1,1) which is "eq" composed with itself. The lower right corner is (8191,8191) and is I composed with itself. Note that there are repeating patterns in the table.

Skipping heuristic: The purpose of a skipping heuristic is to detect I entries. 91% of the table contains I entries. How successful is the skipping heuristic described earlier? It detects 58% of the I entries. I entries caught by the skipping heuristic are highlighted in Figure 2. As in Figure 1, each entry is represented by a single pixel. I entries caught by the skipping heuristic are shown in black. All other entries appear in white. The upper left corner is entry (1,1) which is "eq" composed with itself. Note that only the upper left hand corner is shown in order to highlight the pattern which is consistently duplicated throughout the entire table.

Dropping heuristic: Figure 1 has two large blocks of I entries in the bottom right hand corner: from row 6830 by column 6580 to row 8191 by column 7167 and from row 6830 by column 7510 to row 8191 by column 8191. We define a new version of a skipping heuristic called the dropping heuristic which detects that the result is I whenever the subscripts are in the range of one of the two large I blocks. Note that these two areas are substantial in size. They consist of over 1.7 million entries. Some of these entries are already identified by the existing skipping heuristic. Experimental results for the dropping heuristic are given in the next section.

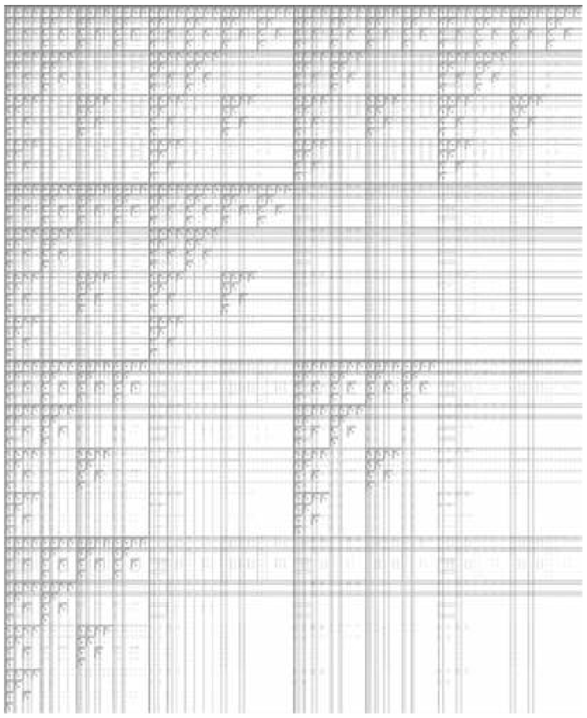
Figure 1 - Visualization of the full composition table

6. Experiments

A precision of 10% (γ=0.1) is used to estimate the number of IA networks to solve. This determines the relative error in a confidence interval within the constraints of the variance for the distribution. As the number of data points, or n, increases so do the degrees of freedom, thus producing a more accurate t-statistic and a correspondingly narrower confidence interval.

Confidence intervals of 95% are used. In the first series of runs, the Central Limit Theorem (CLT) ensures that the data points are normally distributed so that the t-statistic can be used in calculating the confidence intervals. Runs of 400 tests are done, skipping the first ten values (to ensure the system steady state has been achieved); thirty groups of thirteen points are taken as the averages of averages to satisfy the CLT. For the second set of tests involving variable numbers of nodes, runs of ten are done. This approach is taken after having determined that the timing results are roughly normally distributed so the application of the CLT is not required.

Note that in all the results presented in this section, we do not include the time required to generate the full composition table. The table is efficiently generated once, stored, and re-used in subsequent tests.

For the first set of experiments, we set the IA network size at 100 nodes and generate random networks where the sparseness of I varies from 10% to 90% in increments of 5% (see table 2 and figure 3). A sparseness of 10% means that 10% of the edges in the network are labeled with I. Each entry in table 2 reports the average time for a particular algorithm and sparseness to make the network path consistent. Note that we are not finding a consistent scenario. Each algorithm uses the same path consistency code; we vary the composition operations and heuristics used. The shaded entry in each row highlights the fastest algorithm. "fct" is shorthand for our proposed storage of the full composition table in memory. Hogge's composition table algorithm used in conjunction with the skipping heuristic is the fastest method at 20% and over 35% sparseness. Our proposed fct produces better results at 10, 20, 25 and 30 percent. Allen's algorithm is consistently the slowest.

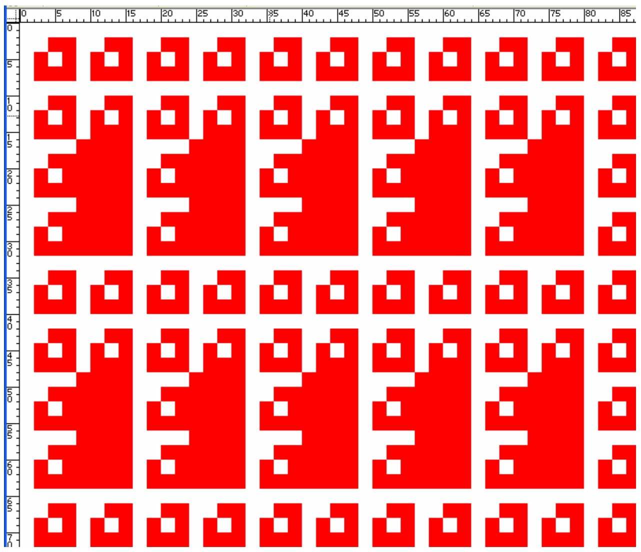
Figure 2 – Effect of skipping (only the upper left hand corner is shown)

For the next set of experiments, we fix the sparseness at 60% and vary network size from 50 to 900 nodes in 50 node increments. In order to not give our algorithm an unfair advantage, we choose a sparseness of 60%. This is the approximate value in figure 3 where our algorithm has its worst performance.

Results appear in table 3. The average is computed from 10 samples and 95% CIs within a precision of γ=0.1. Once again, we are only making the networks path consistent. Hogge+skip and Hogge+skip+weight are faster for small networks between 50 and 250 nodes. Our fct algorithm produced the fastest times from 300 to 900 node IA networks!

7. Future Work

One could look for additional patterns in the full composition table by using the visualization program to graph other conditions of interest. Additionally, multidimensional visualization software could be used to observe any interaction between more than two variables.

Another possibility is to study the effect of altering the assignment of basic relations to individual bit positions. Of the 13! different assignments of relations to bits, is there one assignment which maximizes the size of a contiguous block of I? This could be used to improve the dropping heuristic.

Table 2 - Effect of sparseness on path consistency algorithms for 100 node IA networks

% I	Allen	fct	Allen+skip	fct+skip	Hogge	Hogge+skip	fct+skip+drop
10	0.158	0.054	0.072	0.070	0.068	0.062	0.072
15	0.180	0.062	0.083	0.081	0.069	0.064	0.084
20	0.224	0.072	0.093	0.093	0.073	0.070	0.095
25	0.249	0.074	0.099	0.098	0.080	0.075	0.102
30	0.281	0.080	0.110	0.108	0.087	0.082	0.113
35	0.401	0.099	0.122	0.120	0.096	0.085	0.124
40	0.518	0.117	0.133	0.132	0.105	0.092	0.134
45	0.790	0.157	0.144	0.147	0.126	0.098	0.150
50	0.930	0.169	0.144	0.144	0.127	0.095	0.148
55	1.069	0.181	0.130	0.132	0.116	0.082	0.135
60	1.143	0.150	0.075	0.081	0.090	0.044	0.081
65	1.025	0.105	0.041	0.047	0.062	0.023	0.047
70	0.813	0.071	0.022	0.029	0.047	0.012	0.029
75	0.786	0.059	0.015	0.022	0.044	0.009	0.022
80	0.748	0.048	0.009	0.015	0.041	0.005	0.015
85	0.746	0.040	0.006	0.010	0.039	0.004	0.009
90	0.729	0.031	0.003	0.005	0.038	0.002	0.005

The differences identified in the occurrence of basic relations in I and high cardinality sets could be used to implement and test a weighting heuristic. Preference for "o" and "oi", neutrality towards "d", "di", "s", "si", "f", "fi" and "eq", and a lower weighting for "b" and "bi" would be used.

A meta program to select the optimal composition algorithm depending on the particular IA network instance may be useful.

Since 91% of the table's entries consist of I, compression techniques may be useful.

Freksa [6] defines two basic labels to be neighbors if one can be directly transformed into the other by continuously shortening, lengthening, or moving the intervals (e.g., "b" and "m" are neighbors). A subset of I is a conceptual neighborhood if the elements are path-connected through neighbor relations (e.g., {b,m,o} is a conceptual neighborhood). Freksa presents a compact representation and efficient reasoning algorithms for computing the composition of two conceptual neighborhoods. We plan to generate a complete conceptual neighborhood composition table by removing rows and columns from our complete composition table that do not have a conceptual neighborhood set as its first entry. We will then study the new table's properties as was done in section 5 of this paper.

8. Conclusion

Initial results on small networks suggest that the full composition table is approximately an order of magnitude faster than Allen's base algorithm. However, it does not outperform the original heuristic based approaches. For IA networks between 100 and 150 nodes, our experimental results concur with those reported in [2].

The skipping heuristic provides excellent benefits for smaller IA networks. But as network size increases, the heuristic's usefulness decreases.

For large networks over 250 nodes, our full composition table outperforms the other algorithms. In the extreme case, the full composition table provided more than a 3.4 times improvement over Allen with skipping in a 900-node IA network.

For problems involving small networks, clever heuristics are useful. As network size increases, the cost of computing the heuristics becomes greater than their usefulness. In this case, it is best to use our approach of a full composition table.

To our knowledge, this is the first time the full composition table has been generated, stored, studied and visualized. This allowed us to investigate interesting new properties of the table.

9. Acknowledgements

The second author is supported by an NSERC Discovery Grant.

References:

[1] J.F. Allen, Maintaining Knowledge about Temporal Intervals, *Communications of the ACM, 26*, 1983, 832-843.

[2] P. van Beek, & D. Manchak, The Design and Experimental Analysis of Algorithms for Temporal Reasoning, *Journal of Artificial Intelligence Research, 4*, 1996, 1-18.

[3] B. Nebel, Solving Hard Qualitative Temporal Reasoning Problems: Evaluating the Efficiency of Using the ORD-Horn Class, *Constraints, 1,*(3), 1997, 175-190.

[4] J.C. Hogge, TPLAN: A temporal interval-based planner with novel extensions. Technical Report, UIUCDCSR87, Department of Computer Science, University of Illinois, USA, 1987.

[5] S. Zhang, & C. Zhang, Propagating temporal relations of intervals by matrix, *Applied Artificial Intelligence, 16*, 2002, 1-27.

[6] C. Freksa, Temporal reasoning based on semi-intervals, *Artificial Intelligence, 54* (1-2), 1992, 199-227.

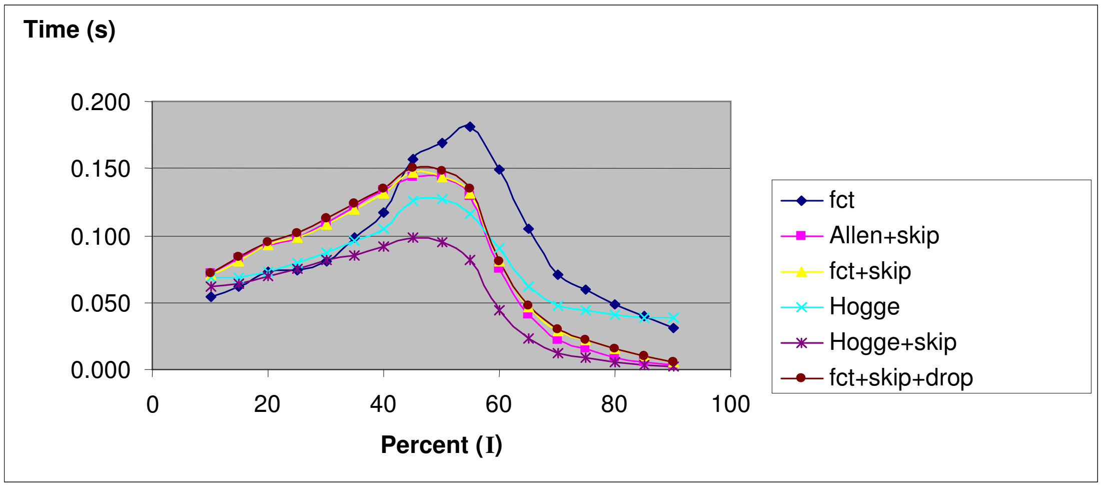

Figure 3 - Effect of sparseness on path consistency algorithms (except Allen) for 100 node IA networks

Table 3 - Effect of IA network size on path consistency algorithms with 60% sparseness

Nodes	Allen	Allen+skip	Allen+skip+drop	fct	fct+skip	fct+skip+drop	Hogge	Hogge+skip	Hogge+skip+weight
50	0.106	0.003	0.003	0.019	0.007	0.007	0.005	0.002	0.002
100	1.125	0.076	0.078	0.151	0.083	0.082	0.087	0.044	0.032
150	2.423	0.504	0.519	0.391	0.434	0.452	0.436	0.338	0.179
200	3.019	1.106	1.141	0.586	0.954	0.989	0.796	0.809	0.420
250	3.736	2.078	2.127	0.825	1.856	1.930	1.336	1.650	0.819
300	5.122	3.694	3.710	1.241	3.388	3.511	2.197	3.289	1.610
350	6.301	6.200	6.231	1.721	5.893	5.938	3.304	5.532	2.777
400	8.514	10.122	10.145	2.338	9.015	9.492	4.450	8.917	5.116
450	11.469	15.240	15.487	3.359	13.612	14.290	6.328	14.072	7.361
500	15.650	22.299	22.463	4.574	20.139	20.793	9.406	20.409	10.665
550	19.995	31.485	31.992	6.440	26.913	28.235	11.757	28.794	14.917
600	25.632	41.656	42.735	8.921	36.947	39.303	16.808	39.564	20.319
650	32.843	54.943	56.125	12.038	48.278	50.903	21.927	51.746	26.992
700	43.400	71.718	72.762	16.720	63.579	64.947	28.895	66.821	35.337
750	53.434	89.251	90.569	20.601	81.409	84.021	37.008	84.115	
800	68.724	113.175	112.964	29.228	96.987	102.744	48.512	104.516	
850	83.541	137.169	139.400	37.622	122.175	126.551	61.292	129.914	
900	102.660	166.449	168.105	48.401	147.402	155.651	74.516	160.818	