2nd International Conference on Autonomous Robots and Agents
December 13-15, 2004 Palmerston North, New Zealand

Simple Optimisation, through Parallelisation and Other Methods, of Temporal Networks

Tony A. Meyer
Institute of Information and Mathematical Sciences,
Massey University, Auckland, New Zealand
T.A.Meyer@massey.ac.nz

Abstract

Ensuring suitable temporal flow for a given sequence of events is essential for many applications. One method of describing permitted flow is use of an 'interval Allen' network; essentially a temporal constraint graph. While Pinhanez provides algorithms that create such graphs, these operate in $O(N^2)$ time, with respect to the number of nodes that the graph contains. This paper outlines a method in which many smaller graphs are made and then combined, which considerably reduces the required time to create the complete graph. The resulting algorithm is also well suited to parallel execution, further reducing the time required to create the graph. The arc consistency algorithm used to temporally iterate nodes through the network is also suitable for parallelisation.

Keywords: temporal flow, PNF constraints, parallelisation, optimisation

1 Introduction

The temporal flow of events is of central importance to many applications. One of the most established methods of controlling this flow is to use an interval constraint graph, as described by Allen [1]. Typically the concern is whether a particular set of temporal constraints are a valid solution (i.e. are possible) or not [2]; here the interest is more in utilising the resulting graph to control the execution of an agent – we assume that the given set of constraints is valid.

The specific domain that this technique is currently used in is control of synthetic characters in live stage performances [3, 4]. The synthetic characters are given a broad outline of the expected temporal flow of the performance, and must match this to the events that actually occur. The script may primarily consist of simple consecutive events, or may have complex event possibilities (involving overlaps, multiple branches, and so on).

As a specific example, the technique has been used to control both input and output robotic devices in live stage performances. Simple robotic tripods controlled the movement of cameras forming the vision system, and these moved as directed by the interval algebra. A robotic output device (a simple wheeled robot dressed up to look like a character) was also used in rehearsals (although not the final performance), again controlled through the interval algebra [3]. The technique has also been utilised in other areas, for example control of intelligent robotic television cameras [5].

2 Temporal Constraint Networks

2.1 Interval Allen Networks

Time intervals are the fundamental concept of the interval algebra proposed by Allen [1]. Within the network, temporal constraints are described in terms of disjunctions of the thirteen possible primitive relationships between two time intervals (equal, before, meet, overlap, during, start, finish and their inverses). An interval algebra network, often called an IA-network, is a binary constraint satisfaction network, where the nodes correspond to time intervals and the arcs are binary temporal constraints between the nodes.

Allen describes an algorithm to infer stronger temporal constraints between the intervals of an IA-network than might be present through the given description of the temporal flow. This algorithm essentially reduces the set of relations in each arc by computing its intersection with the transitive closure, for every node I, of the constraints between A and I and I and B. This is a classical path-consistency algorithm which ensures that, for every sub-network of three nodes A, B, and C, the set of relations presenting the temporal constraint between A and C is contained in the actual transitive closure of the constraints between A and B, and B and C [5].

2.2 PNF Networks

Pinhanez [5] outlines a method of reducing the IA-network to a simpler Past-Now-Future (PNF) Networks, which contains all the information that is required for the domain, but is less computationally

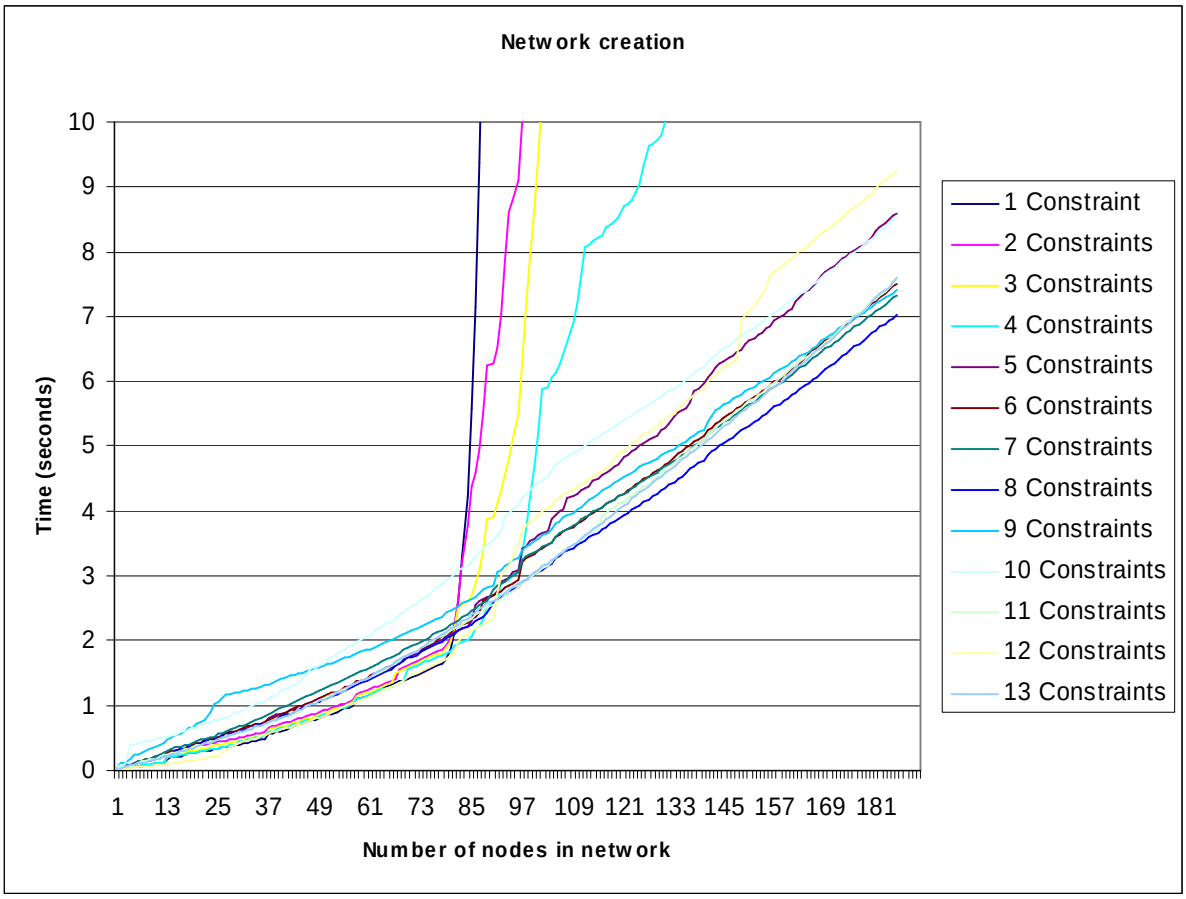

Figure 1: Time taken to construct an interval network for various sizes of network with varying numbers of processor nodes.

costly. These networks are constructed from IA-Networks, but Pinhanez describes an $O(N^2)$ (with respect to the number of nodes in the network) arc-consistency algorithm that works with the PNF Network in order to utilise the network to determine temporal flow.

3 Efficiency

3.1 Network creation

Allen's algorithm for updating the temporal network [1] is $O(N^2)$ with respect to the number of nodes in the network. While this poses no problems for small intervals, the computational time required quickly becomes unmanageable with large networks.

In the author's problem domain, each node (interval) is an event that occurs during a scene (or possibly the entire performance). At the minimum, there are as many intervals as there are lines of dialogue in the script, however once actions are added to this the number quickly increases. This is especially so if the intervals are automatically generated rather than hand tuned, and if the events described are very small (a series of small movements in the face of a synthetic actor, for example, rather than a more general "smile" event).

However, the exact time required to create the network greatly depends on the constraints between the nodes in the network. A constraint with all thirteen possible temporal disjunctions is much quicker to add (the intersection calculation can be optimised away) than a constraint with only one possible temporal disjunction. Effectively, adding a constraint with all thirteen relations adds almost no new information to the network (only that there is another node, but that any relation is possible between that node and the rest). Conversely, adding a constraint with a single relation adds much information to the network.

This is demonstrated in Figure 1, where the time taken to generate networks is compared for networks where the arcs are of fixed sizes. Once the network grows sufficiently large, the time required for creation grows much more rapidly with arcs of a smaller size (as more information is added with each arc).

3.2 Arc consistency

The arc consistency algorithm described by Pinhanez is linear with respect to the number of constraints, but

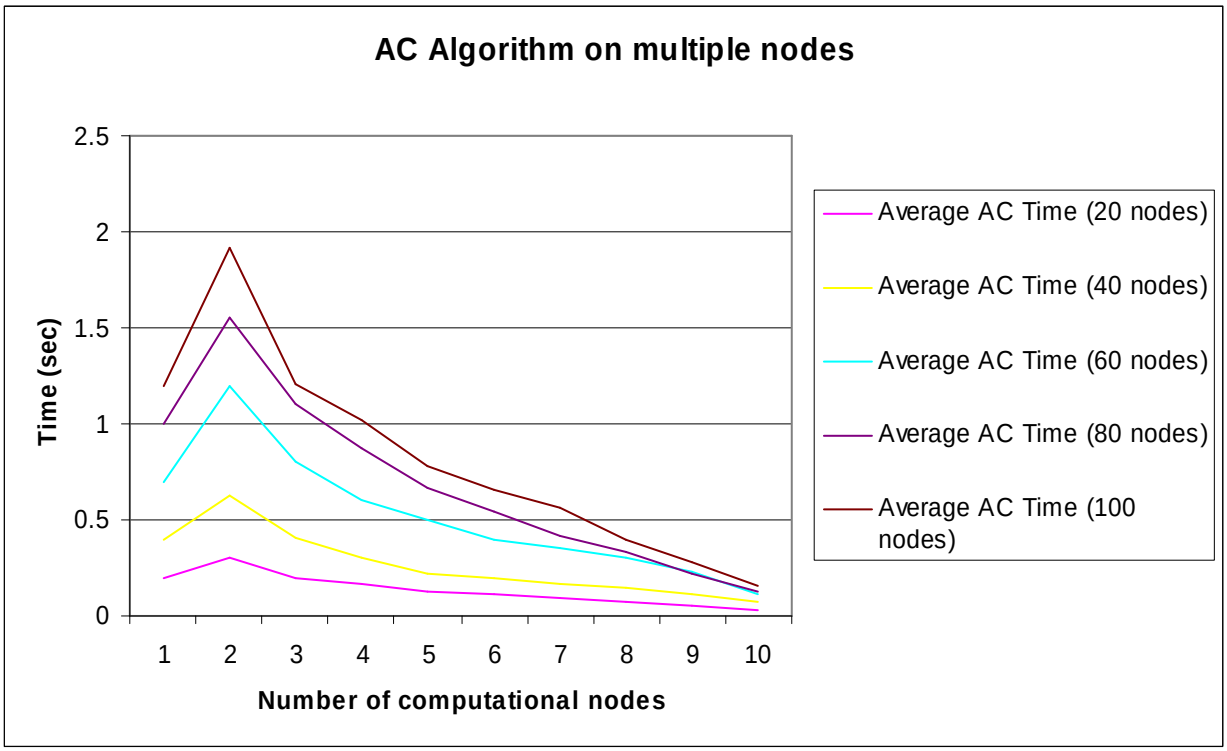

Figure 2: Average time taken for a single iteration of the arc consistency algorithm for various sizes of network with varying numbers of processor nodes.

of order N^2 with respect to the number of nodes (time intervals) in the network. This presents the same problem as the creation of the IA-Network itself.

The arc consistency algorithm is, in fact, run at least twice in Pinhanez's 'interval engine' (once to obtain a set of possible new states, and once to reduce the set to a single selected state). The problem is made worse as a result of this because in one execution the algorithm runs through all intervals whose state is not currently 'future' (more specifically, whose state does not 'time-expand' to 'past-now-future'). At the start of the sequence of events, all events are in the future, but as time progresses, these all move towards the 'past' state.

As a result, the more time progresses, the slower the engine will run, resulting in a lopsided performance (here, the engine is controlling the actions and reactions of synthetic actors, so the actors appear to get slower and less responsive as time progresses).

If we are certain that the arc consistency will never generate an invalid state (i.e. that a valid state will always be possible from the current state), then this can be improved so that when execution reaches a midpoint the algorithm becomes faster and faster again until the original speed is reached. To achieve this, we simply exclude from the central loop of the algorithm any intervals whose state is already minimised (i.e. 'past', 'now, or 'future'), as these cannot be further reduced (other than to an empty, invalid, state). However, while improved, this still causes a lopsided performance.

As such, while the PNF Network and arc consistency algorithm are well matched to the problem, gains must be made in the efficiency of the algorithm in order for it to be of practical use for this project. These gains do not need to reduce the time to linear, simply negate some of the losses due to the N^2 nature.

4 Adaptions

4.1 Network Creation

While the constraints are specified as input to the system and so not able to be altered, the speed of algorithm can be increased by creating a number of smaller sub networks and then combining these into one larger network. The constraints in each of the sub networks are likely to include more than one relation, as a result of the updating process. As a result, although the updating cycle must be run more times, the time for many of the cycles is reduced.

The resulting process is also easily parallelised for execution on multiple processor nodes for further gains in execution time. Each of the sub networks can be created on a separate computational node, and then combined (either by the root computational node, or also as a combined effort) to give the result network.

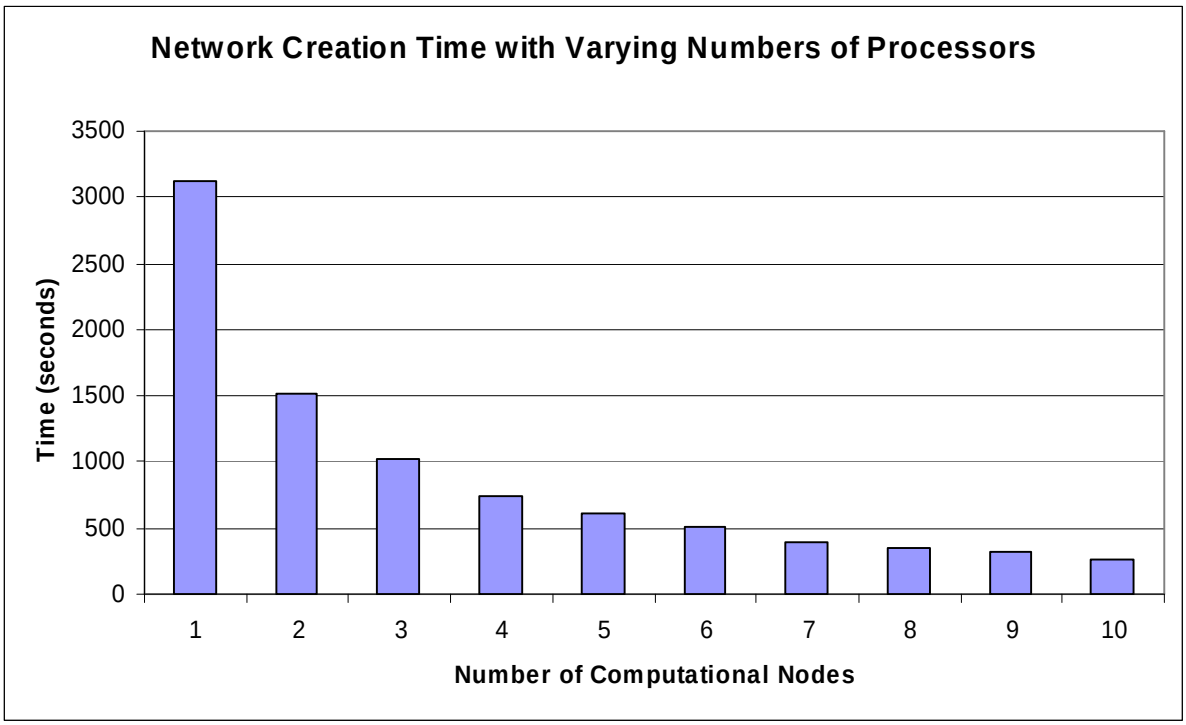

Figure 3: Average time taken for network creation for varying numbers of processor nodes.

4.2 Arc Consistency

The arc consistency algorithm consists of two loops, some table lookups, adding and removing items from a queue, and one intersection calculation. Of these, only the intersection presents itself as an optimisation target; however, the intersecting sets have a maximum length of three (past, now, future), and so no gains are to be made here, either (other than general implementation tricks).

However, the arc-consistency algorithm is well suited to parallelisation. Simply put, the algorithm loops through all the nodes in the network and checks if the node is consistent with the domain; if the node is not, then its state is updated and it is re-added to the list of nodes to check. Throughout this process, the constraint network remains unchanged, and so in the inner loop, running through every node in the network, each iteration is independent. As such, the computational requirement for the loop can be spread over a number of computational nodes, as long as each has access to the constraint network (which typically is constant throughout the run) and the current state of the nodes.

5 Results

5.1 Network Creation

Figure 3 shows the results of providing additional processors to the network creation task. The time required decreases at a higher rate than the number of processors increase. This demonstrates the added efficiencies gained by creating smaller networks and combining them (reducing the time required for the set operations).

5.2 Arc Consistency

Figure 2 shows the results of running the arc consistency algorithm on varying numbers of processors. The time taken is the average time that a single iteration through the algorithm took, adjusted for the creation of the various sub-networks (i.e. normalised to case of a single network).

Although the gains are not as significant as those from modifying the network creation, the time taken is reduced to a much more manageable level. Larger networks gained more than smaller ones; for the most part the smaller networks were reasonably efficient with the simpler algorithm and so did not have much to gain.

In practice, it therefore appears that providing additional computational nodes to the system, as the number of nodes in the interval network increases, would result in the time responsiveness of the system being maintained. Helpfully, the relationship between the number of interval nodes added and the additional number of processor nodes required is not linear, so the requirements are not as high.

6 Conclusions & Future Work

Through these small optimisations, and particularly through the use of parallelisation, the $O(N^2)$ nature of the algorithms in Pinhanez's interval engine are able to remain suitable for the synthetic actor domain even with much larger numbers of intervals.

Future work will investigate whether more fundamental alterations to the network creation and arc consistency algorithms are able to offer more gains in efficiency. One such possibility is utilising the speed of parallelised network creation to fully constrain (rather than the 3-node constrain with the current algorithm) the network on creation. Although this problem is NP-hard, through use of parallelised computation moderately sized network creation should be feasible (particularly given that this typically needs to be only carried out once, prior to execution). If this is possible, then temporally iterating nodes through the network would be much simpler and faster.

7 References

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

[2] A. K. Pujari and T. Adilakshmi, "A Novel Heuristic to Solve IA Network by Convex Approximation and Weights," in *PRICAI 2004: Trends in Artificial Intelligence*, vol. 3157, *Lecture Notes in Computer Science*, C. Zhang, H. W. Guesgen, and W. K. Yeap, Eds. Auckland, New Zealand: Springer-Verlag, 2004, pp. 948-949.

[3] T. A. Meyer and C. H. Messom, "Development of extemporaneous performance by synthetic actors in the rehearsal process," presented at 3rd International Conference on Entertainment Computing, Eindhoven, The Netherlands, 2004.

[4] C. Pinhanez, "Computer Theatre," M.I.T. Media Lab, Perceptual Computing Section Technical Report 378, 1996.

[5] C. Pinhanez, "Representation and Recognition of Action in Interactive Spaces," Massachusetts Institute of Technology, 1999.