Constraint-based multi-agent path planning

Malcolm Ryan
Centre for Autonomous Systems
School of Computer Science and Engineering
University of New South Wales, Australia

Abstract

Planning collision-free paths for multiple robots traversing a shared space is a problem that grows combinatorially with the number of robots. The naive centralised approach soon becomes intractable for even a moderate number of robots. Decentralised approaches, such as priority planning, are much faster but lack completeness.

Previous work has demonstrated that the search can be significantly reduced by adding a level of abstraction (Ryan 2008). I first partition the map into subgraphs of particular known structures, such as *cliques*, *halls* and *rings*, and then build abstract plans which describe the transitions of robots between the subgraphs. These plans are constrained by the structural properties of the subgraphs used. When an abstract plan is found, it can easy be resolved into a complete concrete plan without further search.

In this paper, I show how this method of planning can be implemented as a constraint satisfaction problem (CSP). Constraint propagation and intelligent search ordering further reduces the size of the search problem and allows us to solve large problems significantly more quickly, as I demonstrate this in a realistic planning problem based on a map of the Patrick Port Brisbane yard. This implementation also opens up opportunities for the application of a number of other search reduction and optimisation techniques, as I will discuss.

Introduction

A major aspect of solving any problem in artificial intelligence (AI) is *knowledge engineering*, that is taking the available background knowledge about a problem and expressing it in a way that it can be exploited by an AI algorithm. This task is crucial to solving any realistically large problem, including the one I address in this paper: multi-agent path planning.

Planning for a single robot, once issues of geometry and localisation have been addressed, becomes a simple matter of finding a path through the *road-map* – the graph G representing the connectivity of free space – between its starting and goal locations. When planning for multiple robots, however, we also need to take into account the possibility for collisions en route. A decentralised approach in which each robot simply planned its own path without reference to the others would not work.

A logical solution is to treat the entire collection of robots as a single entity and use a centralised planner to co-ordinate them. If we again ignore issues of geometry, this equates to finding a path through the *composite graph* $G^k = G \times G \times \ldots \times G$, where k is the number of robots. Each vertex in this graph is a k-tuple of vertices of G representing the positions of each robot. Each edge represents the movement of one robot between neighbouring vertices. Vertices which represent collisions are excluded. A plan is now a path between the vertex representing the robots' initial locations to the vertex representing their goals.

It is easy to see that the size of this graph grows combinatorially with the number of robots. Any algorithm which performs a naive search of the graph will soon require far too much time and memory to complete. A common solution is *prioritised planning* which gives each robot a priority and plan for them in order, with lower priority robots integrating their plans with those of higher priority. This effectively prunes the search space by eliminating certain possibilities (in which higher priority robots go out of their way to allow lower priority robots to pass). Searching this reduced space is much faster, but the pruning may eliminate the only viable solutions, making the algorithm incomplete.

In order to efficiently handle large numbers of robots without sacrificing completeness we need some way to incorporate more knowledge about the domain. In previous work (Ryan 2008)] I have shown how structural information about the road-map can be exploited to significantly reduce search. The map is decomposed into subgraphs of particular known structure, *cliques*, *halls* and *rings*, which place constraints on which robots can enter or leave at a particular time. Planning is done at a level of abstraction, in terms of the configuration of each subgraph and the robots' transitions between them. Once an abstract plan has been constructed the concrete details of robots' movement within each subgraph can be resolved algorithmically, without the need for further search. This approach is proven to be sound and complete.

In this work we extend these previous results by showing how the subgraph planning process can be encoded as a constraint satisfaction problem (CSP). With this formulation, a CSP-solver can make more efficient use of the domain knowledge to prune the search space to a much greater degree allowing us to solve problems significantly larger

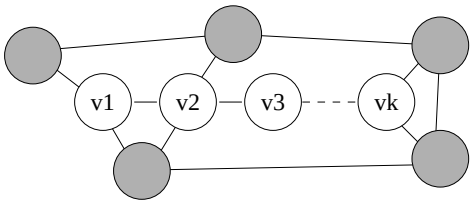


Figure 1: A hall subgraph.

than before. It also opens up the possibility for optimisation of plans and more complex planning tasks than simple goal achievement.

In the next section I will describe the subgraph planning approach in greater detail. This will be followed by a brief introduction to constraint programming leading into the constraint representation of our planning problem. The efficiency of this new approach will be evaluated on tasks using a map of the Patrick Port Brisbane facility and we will conclude with discussion of related work and future directions.

Subgraph Planning

We can formalise our problem as follows. The road-map is provided in the form of a graph $G = (V, E)$ representing the connectivity of free space for a single robot moving around the world (e.g. a vertical cell decomposition or a visibility graph, (LaValle 2006)). We also have a set of robots $R = \{r_1, \ldots, r_k\}$ which we shall consider to be homogeneous, so a single map suffices for them all. All starting locations and goals lie on this road-map.

We shall assume that the map is constructed so that collisions only occur when one robot is entering a vertex v at the same time as another robot is occupying, entering or leaving this vertex. Robots occupying other vertices in the map or moving on other edges do not affect this movement. With appropriate levels of underlying control these assumptions can be satisfied for most real-world problems.

The road-map is partitioned into a collection of induced subgraphs $\mathcal{P} = \{S_1, \ldots, S_m\}$ of known structure. In this paper we shall consider only one kind of subgraph: the *hall*. A hall is a singly-linked chain of vertices with any number of entrances and exits, as illustrated in Figure 1. They are commonly found in maps as narrow corridors or roads which may contain several robots but which prevent overtaking. Formally this is represented as $H = \langle v_1, \ldots, v_m \rangle$ with: $(v_i, v_j) \in E$ iff $|i - j| = 1$.

The configuration of a hall can abstract the exact positions of the robots and merely record their order, which cannot be changed without a robot entering or leaving. The new configuration created when a robot enters or leaves is based solely on the previous configuration and the position of the vertex by which it transitions. Resolving a step of the abstract plan means shuffling the robots in the hall left or right to either move the departing robot to its exit or to open a space at the appropriate vertex (and position in the sequence of occupants) and for an incoming robot to enter.

An abstract plan is thus an alternating sequence of hall configurations and subgraph transitions. Previous work has restricted this to a single robot transitioning on each step. The constraint formulation I shall present in this paper allows us to relax this restriction.

Constraint Programming

Constraint programming is a methodology for representing and solving combinatorial search problems through constraint propagation and intelligent search. Problems are represented as collections of *variables* over finite domains (usually subsets of the integers) and *constraints* which are relations between the variables that they are required to satisfy. Constraint solvers are designed to represent a large number of different constraints and use them to propagate information from one variable to another so that their domains are consistent (with some degree of strength) with the constraints between them.

Combining constraint propagation with search, we are able to prune the search space of a problem by alternately assigning values to variables and propagating the change to restrict the domains of other unassigned variables. Informed choice of the search order can maximise the benefits of propagation and further reduce the search. For this project I used Gecode/J – a Java-based constraint solver (Gecode Team 2006).

The Constraint Representation

To convert the planning task into a constraint satisfaction problem we need to describe it as a finite set of integer variables. As it stands the task is open ended: a plan can be of any length. To make it finite we need to restrict the plan to a fixed length. If a plan of a given length cannot be found, then a new CSP representing a longer plan can be constructed and the process repeated.[1]

To begin our representation we number each vertex, each robot and each subgraph. Let $V = \{1, \ldots, n\}$ represent the vertices, $R = \{1, \ldots, k\}$ represent the robots and $S = \{1, \ldots, m\}$ represent the subgraphs. Let V_i be the set of vertices for subgraph i. It is useful, as we will see later, to number the vertices so that each V_i contains consecutive integers. Let $\mathcal{E} = \{(a, b) \mid \exists v_a \in V_a, v_b \in V_b, (v_a, v_b) \in E\}$ be the relation defining adjacency between subgraphs. Let L be the length of the abstract plan.

Abstract plan steps

We can now define the variables we need. For each robot $r \in R$ and each step of the plan $i \in \{1 \ldots L\}$ we have three variables:

$A_i[r] \in S$ is the index of the subgraph occupied by r at step i,

$F_i[r] \in V$ is the index of the first vertex occupied by r at step i,

[1] Note that this makes the problem only semi-decideable. There is no sure way to know when no possible plan of any length exists. In practice, this is rarely a problem. Planning stops when plans get beyond a certain maximum length.

$T_i[r] \in V$ is the index of the last vertex occupied by r at step i.

The first of these variables tells us which subgraph the robot occupies in that step of the plan. It is also important to know the vertices at which the robot enters and leaves the subgraphs (the second and third variables respectively) as they will affect the possible configuration of the subgraph.

We constrain these variables as follows:

Robots can only move between neighbouring subgraphs.

$$A_i[r] \neq A_{i+1}[r] \rightarrow (A_i[r], A_{i+1}[r]) \in \mathcal{E} \quad (1)$$

$F_i[r]$ and $T_i[r]$ must belong to the given subgraph.

$$A_i[r] = a \rightarrow F_i[r] \in V_a \quad (2)$$
$$A_i[r] = a \rightarrow T_i[r] \in V_a \quad (3)$$

Two robots cannot be in the same vertex at the same time.

$$distinct(F_i[1], \ldots, F_i[k]) \quad (4)$$
$$distinct(T_i[1], \ldots, T_i[k]) \quad (5)$$

Consecutive sub-plans are linked by valid transitions.

$$(T_i[r], F_{i+1}[r]) \in E \quad (6)$$
$$T_i[r_x] \neq F_{i+1}[r_y], \forall r_x \neq r_y \quad (7)$$

No-ops only occur at the end of the plan.

$$(\exists r \in R : A_i[r] \neq A_{i+1}[r]) \rightarrow (\exists r \in R : A_{i-1}[r_y] \neq A_i[r]) \quad (8)$$

If a subgraph is full, its occupants cannot move.

$$A_i[r] = a \ \wedge \ count_{\rho \in R}(A_i[\rho] = a) = |V_a| \rightarrow F_i[r] = T_i[r] \quad (9)$$

These constraints apply to any abstract plan, regardless of the structure of its subgraphs, but they fail to completely specifiy the problem. In particular, they do not guarantee that the configuration given by $(T_i[1], \ldots, T_i[k])$ is reachable from $(F_i[1], \ldots, F_i[k])$. To ensure this we must refer to the particular properties of the subgraphs.

Hall ordering

In the case of the hall subgraph, we require that the order of robots in the hall does not change between transitions. If r_x is on the left of r_y at the beginning of a sub-plan it must also be so at the end (and vice versa). We can represent this more easily if we number the vertices in the hall consecutively from one end to the other. Then for two robots in the hall, we will require $F_i[r_x] < F_i[r_y] \Leftrightarrow T_i[r_x] < T_i[r_y]$.

It will be useful in the search for a plan to be able to explicitly choose an ordering between two robots without assigning them to particular vertices. To this end, we create a new set of variables to represent the ordering of robots in each sub-plan: $Ord_i[r_x, r_y] \in \{-1, 0, 1\}$. Conveniently we can use one set of variables to describe the configuration of all halls simultaneously, since the value is only important if two robots are in the same subgraph at the same time. If r_x and r_y are in different subgraphs, then $Ord_i[r_x, r_y]$ is 0. Otherwise it must be either -1 or 1, indicating the two possible orderings: r_x before r_y or r_y before r_x.

Formally we add the following constraints:

Robots are ordered iff they are both in the same hall.

$$A_i[r_x] \in \mathcal{H} \ \wedge \ A_i[r_x] = A_i[r_y] \Leftrightarrow Ord_i[r_x, r_y] \neq 0 \quad (10)$$

Ordering variables affect concrete positions.

$$Ord_i[r_x, r_y] = -1 \rightarrow F_i[r_x] < F_i[r_y] \wedge T_i[r_x] < T_i[r_y] \quad (11)$$

$$Ord_i[r_x, r_y] = 1 \rightarrow F_i[r_x] > F_i[r_y] \wedge T_i[r_x] > T_i[r_y] \quad (12)$$

Ordering variables persist across subplan transitions.

$$A_i[r_x] = A_{i+1}[r_x] \wedge A_i[r_y] = A_{i+1}[r_y] \rightarrow Ord_i[r_x, r_y] = Ord_{i+1}[r_x, r_y]$$

This completes our description. Any abstract plan which satisfies these constraints can be resolved into a correct concrete plan without further search.

Search

Constraint propagation alone will not solve this problem; the constraints are not powerful enough to eliminate every wrong solution. We must also perform a search, experimentally assigning values to variables until a complete plan is found that satisfies all the constraints. By enumerating all the variables at the outset, we are able to assign values to them in any order we wish, unlike standard path-planning algorithms which generally operate in forward temporal order only.

Common wisdom in constraint solving is to assign variables so that failures, if they are going to occur, happen early at shallow levels of the tree so that large amounts of backtracking are avoided. The standard heuristic is to assign the most constrained variables first. In this particular problem it makes sense to assign the subgraph variables $A_i[r]$ first, followed by the order variables $Ord_i[r_x, r_y]$ and finally the transition variables $F_i[r]$ and $T_i[r]$, since each is strongly constrained by the one that comes before. In each case we choose the variable with the smallest domain.

When choosing a value for the variable there are two things to consider: 1) choose a value which is most likely to lead to a solution, 2) choose a value which places the least constraint on other variables. When choosing subgraph values for the $A_i[r]$ variables we apply the first principle by choosing the subgraph which is closest to the next assigned subgraph for robot r (based on a precomputed single-robot all-shortest-paths matrix). If there are two such options, then the subgraph with the fewest occupants is selected, according to the second principle.

The heuristic for selecting the ordering value for $Ord_i[r_x, r_y]$ is to consider the concrete values that it immediately affects $F_i[r_x]$, $T_i[r_x]$, $F_i[r_y]$ and $T_i[r_y]$. For each ordering we can easily compute the resulting domain sizes

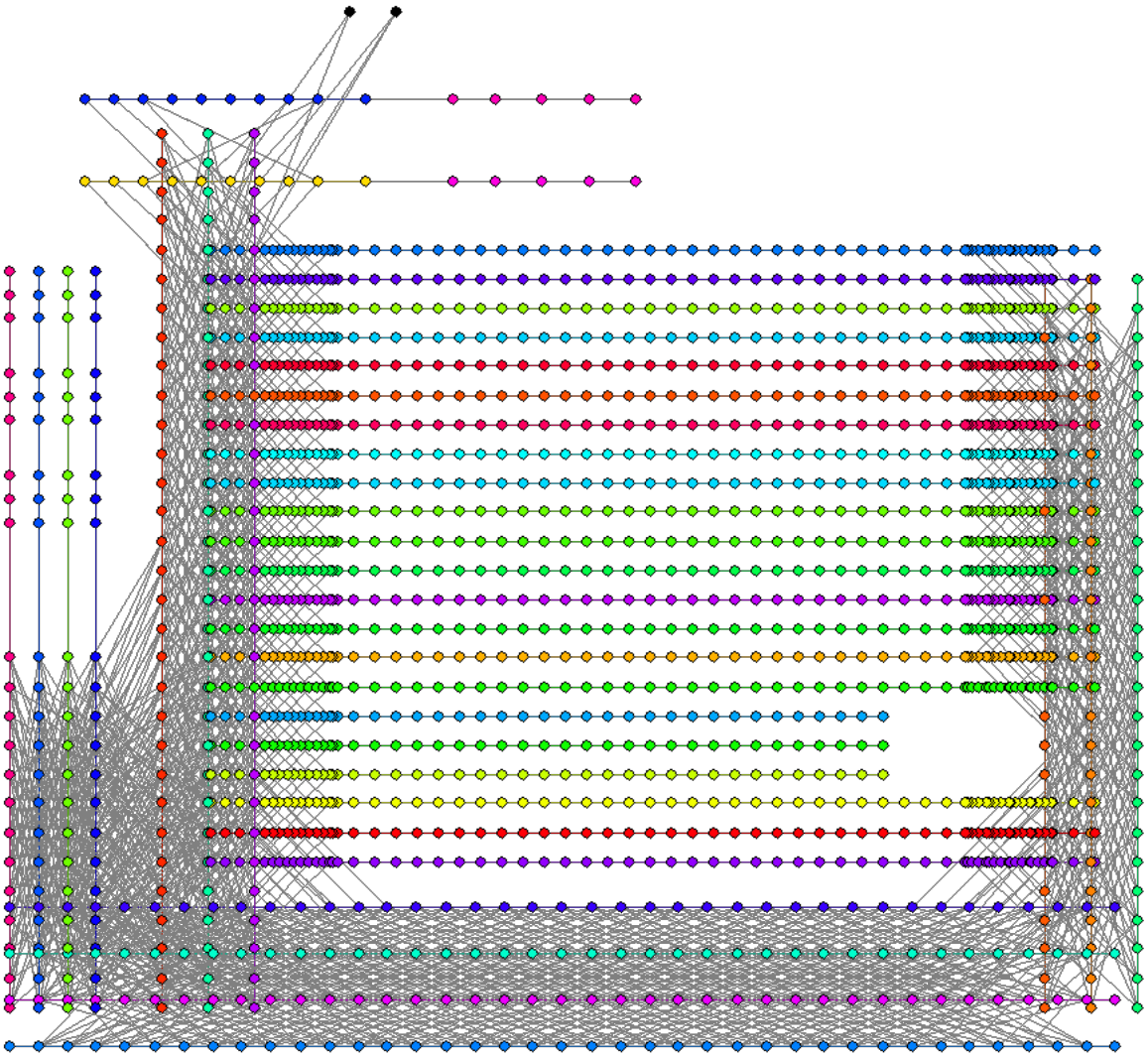


Figure 2: The map of the Patrick yard at Port Brisbane.

for each of these variables (ignoring the effect of any other constraints). The ordering which leaves the largest number of alternatives is preferred, by the second principle above.

Finally, values for the concrete steps $F_i[r_x]$ and $T_i[r_x]$ are chosen to minimise the distance between the beginning and end of the plan step.

Experiment: The Patrick Port

To evaluate this new planning system I have applied it to a realistic planning problem. Figure 2 shows a map of the Patrick yard at Port Brisbane in Queensland, Australia. This map is used to plan the movement of straddle-carriers – enormous, automated vehicles for moving shipping containers around the yard. Efficient, co-ordinated planning for these vehicles is important for the smooth running of the facility.

The problem domain

The map is an undirected graph of 1808 vertices and 3029 edges. The vertices are naturally connected in long straight chains representing the roads around the facility. These roads mean that the vertices can be partitioned into 40 hall subgraphs, with only 2 vertices left over, which must be treated as singletons. The reduced graph has 187 edges connecting neighbouring subgraphs.

This reduced graph was constructed by hand with the aid of a simple interactive tool. Choosing the partition was not difcult; the roads around the port are obvious in the map and provide a natural set of halls. No effort was made to optimise this partition in any fashion to suit the algorithm.

Approach

The map was populated with a number of robots which varied from 1 to 40. Each robot was assigned a random ini-

tial and final position. A single-robot shortest paths matrix was calculated for the reduced graph and used to calculate the minimum length of the plan as the length of the longest single-robot plan.

A constraint problem was constructed in Gecode/J as described above. The initial and goal states were constrained to their appropriate values and then a search was conducted. An iterative deepening approach was used. A minimum estimate of the plan length was computed by taking maximum shortest path distance for each robot individually. If a plan of this length could not be found, then the length was incremented and the search repeated, until a solution was found or the planner exceeded a 2 minute time limit.

The same problems were also solved with a prioritised planner (also encoded as a CSP in Gecode/J). I compare the results below.

Results

One hundred different experiments were conducted for each number of robots.[2] The results are plotted in Figures 3(a) and 3(b). The graphs show the median values for total time to construct the CSP and search for a solution and the total memory usage, with whiskers showing the rst and third quartiles. Experiments were run with a time limit of 120 seconds and a maximum heap size of 2 gigabytes. Experiments which exceeded these limits are treated as taking infinite time and memory.

The difference between the two approaches is quite pronounced. The abstract approach shows a much slower rate of increase in both runtime and memory. Performance is comparable for up to 4 robots, but after that point the abstract approach is clearly superior. At 33 robots the graph of the prioritised planner ends because it began to fail more than 50% of the time. The abstract planner was able to handle up to 40 robots, taking only slightly more than 10 seconds in the median case.

There is a noticeable change in both the time and memory graphs around the point of 15 or 16 robots. The explanation for this threshold can be found in the subplan sizes. Because of the high connectivity of the graph, two subgraphs randomly chosen can almost always be connected by at most one intermediate subgraph. The probability that more than one is required is small, approximately 4%. As more robots are added to the plan the probability increases that at least one will require a longer plan. The probability reaches 50% about the 15-16 robot mark. So at this point the majority of experiments begin with a plan of 4 abstract steps, while a majority of the smaller problems require only 3. The longer plan requires more variables per robot to represent and thus more time and memory to complete.

Analysis of median values doesn't give us the full picture. Table 1 shows the number of experiments which failed to complete due to time or memory limits. Many more prioritised experiments failed (23% in total) than abstract experiments (only 1%). In most cases failure occurred because the experiment exceeded the time limit. As stated earlier, constraint-based planning is only semi-decidable, so there is no way to definitely conclude that a problem has no solution, but the data suggests that the incompleteness of the prioritised algorithm may be the issue.

[2]Running times were measured on a 3.20GHz Intel(R) Xeon(TM) CPU running Sun JDK 6.0 with 2Gb of heap.

Robots	% Failures		# Robots	% Failures	
	Abs.	Pri.		Abs.	Pri.
7	0	1	24	1	25
8	0	1	25	0	26
9	0	2	26	2	28
10	0	2	27	0	29
11	0	5	28	0	43
12	0	3	29	2	34
13	0	6	30	3	46
14	0	10	31	4	29
15	0	13	32	4	33
16	0	6	33	4	48
17	1	11	34	2	59
18	1	19	35	1	54
19	0	16	36	7	54
20	2	14	37	3	58
21	0	17	38	9	64
22	0	11	39	7	63
23	0	29	40	6	78

Table 1: Number of failed experiments by problem size. Experiments 1 to 6 showed no failures for either approach.

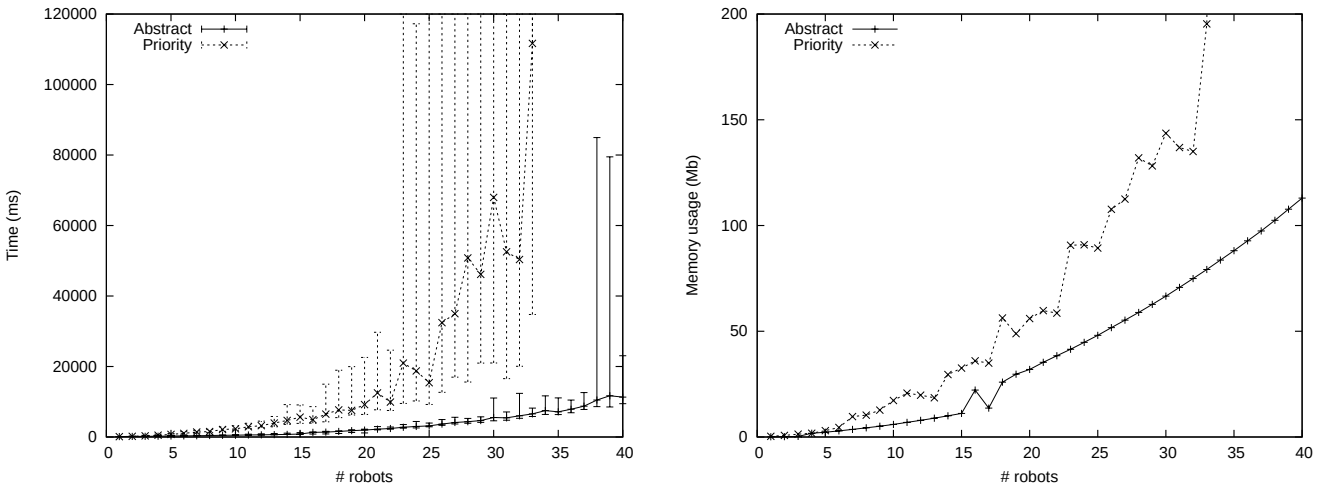
Conclusion

I have demonstrated how the multi-robot path planning problem can be effectively solved for large numbers of robots by making use of appropriate structural knowledge about the map, in the form of a subgraph decomposition. This knowledge can be encoded precisely as a constraint satisfaction problem and solved using a combination of constraint propagation and heuristic search. This allows us to solve problems of unprecedented size, using time and memory that is significantly smaller than the standard approach of prioritisation.

Related work

There has been little previous work in the use of abstractions and modern search techniques in multi-robot path planning. The work that bears most similarity to my own is not explicitly in robot path planning, but in solving the Sokoban puzzle (Botea, Müller, and Schaeffer 2003; Junghanns and Schaeffer 2001). Their division of a map into rooms and tunnels matches to some degree the subgraph decomposition I adopt here. The particular structures they represent are different, but the general ideas of partitioning into independent local subproblems and identifying abstract states from strongly connected components, are the same as those employed in this work. They have not as yet attempted to translate these structures into a formal constraint satisfaction problem.

CSPs have however been applied to a different kind of planning, that is AI task-planning. CPLan (van Beek and



(a) run time (b) memory usage

Figure 3: A comparison of median run-times and memory usage for abstract planning and prioritised planning. Error bars show the first and third quartile.

Chen 1999) directly encodes such planning problems as constraint systems and uses a general purpose constraint solver to find plans. Another approach is to encode the planning graph from an algorithm such as Graphplan (Blum and Furst 1997) and convert it into a CSP, as done in the work of Do and Kambhampati (Do and Kambhampati 2001) and Lopez and Bacchus (Lopez and Bacchus 2003). A related approach is the 'planning-as-satisfiability' technique used in planners such as SatPlan (Kautz, Selman, and Hoffmann 2006).

Future work

This constraint-based approach opens the door to a number of new possibilities. More complex planning problems can be expressed by adding appropriate constraints to the system. If we extended the representation to include variables for the concrete sub-plans in their entirety, we could add extra constraints to prevent certain vertices from being simultaneously occupied, to add a buffer zone between robots. We could specify goals that involve visiting several locations in sequence. It is already possible to have robots that have no particular goal but to stay out of the way.

If we add variables representing the lengths of the concrete plans, we can begin to work on optimisation. As it stands, the algorithm makes no guarantees that concrete plans will be optimal. Finding perfectly optimal plans is likely to be very time consuming, but a branch-and-bound algorithm could provide a viable alternative, yielding the best plan found in the available time.

This leads us to consider what other advanced CSP-solving techniques could be useful. The most immediately obvious is sub-problem independence (Mann, Tack, and Will 2007). Once the $A_[i]$[r] variables have been set, the other variables in this problem are partitioned into a number of subsets which do not affect each other. Solving these sub-problems independently could prevent a lot of unnecessary backtracking.

In conclusion, this paper demonstrates the successful combination of domain knowledge and intelligent problem solving tools. It offers not just a fast planning algorithm, but also a validation of constraint programming as an effective knowledge engineering methodology, and one which we should continue to improve upon.

Acknowledgements

I'd like to thank Guido Tack, Mikael Lagerkvist and Christian Schulte from the Gecode development team for all their patient assistance and advice.

References

Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90(1-2):281–300.

Botea, A.; Müller, M.; and Schaeffer, J. 2003. Using abstraction for planning in sokoban. In *Computers and Games: Lecture Notes in Computer Science*, volume 2883. Springer. 360–375.

Do, M., and Kambhampati, S. 2001. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artificial Intelligence* 132(2):151–182.

Gecode Team. 2006. Gecode: Generic constraint development environment,. Available from http://www.gecode.org.

Junghanns, A., and Schaeffer, J. 2001. Sokoban: Enhancing general single-agent search methods using domain knowledge. *Artificial Intelligence* 129(1-2):219–251.

Kautz, H.; Selman, B.; and Hoffmann, J. 2006. SatPlan: Planning as Satisfiability. In *Abstracts of the 5th International Planning Competition*.

LaValle, S. M. 2006. *Planning Algorithms*. Cambridge University Press.

Lopez, A., and Bacchus, F. 2003. Generalizing GraphPlan by Formulating Planning as a CSP. In *Proceeding of the International Joint Conference on Artificial Intelligence (IJCAI'03)*.

Mann, M.; Tack, G.; and Will, S. 2007. Decomposition during search for propagation-based constraint solvers. Technical Report arXiv:0712.2389v2, Cornell University Library.

Ryan, M. R. K. 2008. Exploiting subgraph structure in multi-robot path planning. *Journal of Artificial Intelligence Research* 31:497–542.

van Beek, P., and Chen, X. 1999. CPlan: A constraint programming approach to planning. In *Proceedings of the AAAI National Conference*, 585–590.