2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

Constraint-based multi-robot 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 prioritised planning, are much faster but lack completeness.**

Previous work has demonstrated that the search can be significantly reduced by adding a level of abstraction [1]. We first partition the map into subgraphs of particular known structure, such as *cliques* and *halls*, 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 easily be resolved into a complete concrete plan without further search.

In this paper, we show how this method of planning can be implemented as a constraint satisfaction problem (CSP). Constraint propagation and intelligent search ordering further reduce the size of the search problem, allowing us to solve large problems significantly more quickly. Empirical evaluation on a realistic planning problem shows the clear superiority of the constraint-based approach, but the value of abstraction is mixed: it allows us to solve more problems at the cost of a time-overhead on simple problems.

This implementation also opens up opportunities for the application of a number of other search reduction and optimisation techniques, as we will discuss.

I. 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 we address in this paper: multi-robot 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 plans its own path without reference to the others does 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 plans for them in order, with lower priority robots integrating their plans with those of higher priority. This effectively prunes the search space by eliminating those 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 [1] we 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, and it takes advantage of the fact that real-world maps are not random but contain common substructures.

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

A. 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 our own is not explicitly in robot path planning but in solving the Sokoban puzzle [2], [3]. Their division of a map into rooms and tunnels matches to some degree the subgraph decomposition we 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 [4] directly encodes such planning problems as constraint systems and uses a general purpose constraint solver to find plans. Another approach is to take the planning graph from an algorithm such as Graphplan [5] and convert it into a CSP, as done in the work of Do and Kambhampati [6] and Lopez and Bacchus [7]. A related approach is the 'planning-as-satisfiability' technique used in planners such as SatPlan [8].

B. Paper outline

In the next section we 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 UNSW AI Laboratory and we will conclude with discussion of related work and future directions.

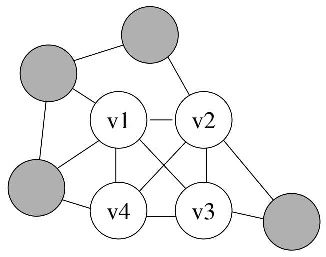
II. Subgraph Planning

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 [9]). 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.

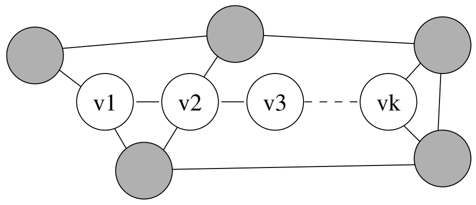
We shall assume that the map is constructed so that collisions only occur when one robot is entering a vertex v at the same time as another robot is occupying, entering or leaving this vertex. Robots occupying other vertices in the map or moving on other edges do not affect this movement. With appropriate levels of underlying control these assumptions can be satisfied for most real-world problems. [1]

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 two kinds of subgraph: the *clique* and the *hall*, illustrated in Figure 1.

A clique is a complete subgraph with each vertex linked to every other. In maps they usually represent large open spaces



(a) A clique



(b) A hall

Fig. 1. Types of subgraphs.

with many entrances and exits. The configuration of a clique can abstract the exact positions of the robots and merely record the set of occupants at any time. So long as the clique is not full, it is possible to rearrange the occupants arbitrarily. When the clique is full, we need separate configurations for each arrangement of robots.

A hall is a singly-linked chain of vertices with any number of entrances and exits. 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 must record the order of its occupants, 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.

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

III. 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

[1]This is also known as the Pebble Motion on Graphs problem [10].

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 we used the Gecode constraint solver [11].

IV. 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.[2]

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.

A. 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,

$T_i[r] \in V$ 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 subgraph (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.

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

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

$$A_i[r] = a \rightarrow F_i[r] \in V_a \tag{2}$$

$$A_i[r] = a \rightarrow T_i[r] \in V_a \tag{3}$$

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

$$distinct(F_i[1], \ldots, F_i[k]) \tag{4}$$

$$distinct(T_i[1], \ldots, T_i[k]) \tag{5}$$

[2]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 a maximum length or time limit is reached.

Consecutive sub-plans are linked by valid transitions.

$$(T_i[r], F_{i+1}[r]) \in E \tag{6}$$

$$T_i[r_x] \neq F_{i+1}[r_y], \forall r_x \neq r_y \tag{7}$$

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

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

If a subgraph is full, its occupants cannot move.

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

These constraints apply to any abstract plan, regardless of the structure of its subgraphs, but they fail to completely specify the problem. They suffice to represent cliques but not halls, as they do not guarantee that the configuration given by $(T_i[1], \ldots, T_i[k])$ is in the same order as $(F_i[1], \ldots, F_i[k])$. To ensure this we must add further constraints.

B. 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.

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

Ordering variables affect concrete positions.

$$Ord_i[r_x, r_y] = -1 \rightarrow F_i[r_x] < F_i[r_y] \wedge T_i[r_x] < T_i[r_y] \tag{11}$$

$$Ord_i[r_x, r_y] = 1 \rightarrow F_i[r_x] > F_i[r_y] \wedge T_i[r_x] > T_i[r_y] \tag{12}$$

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.

V. 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 search-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 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.

VI. Resolution

In previous work we wrote special-purpose solvers to resolve the abstract plan steps into concrete plans. The highly-constrained nature of planning within a subgraph made such solvers possible but also makes them largely redundant. Once the abstract plan has been determined, the concrete plans can be represented as independent constraint problems and solved very quickly, usually without any backtracking.

VII. Experiment

To evaluate this new planning system we have applied it to a realistic planning problem. Figure 2 shows a map of the AI Laboratory at UNSW. The map is an undirected graph of 113 vertices and 154 edges. The map has been decomposed into 3 halls and 2 cliques, leaving 48 singleton vertices that are not part of a larger subgraph. The partitioning was chosen by hand to maximise the length of the halls, thus minimising the diameter of the reduced graph and, as a result, the size of our plans.

A. Approach

The map was populated with a number of robots which varied from 2 to 20. Each robot was assigned a random initial and final position. A single-robot shortest paths matrix was calculated for the reduced graph and used to calculate a lower bound on the length of the plan (equal to the length of the longest single-robot plan).

Eight different approaches were used to solve this problem, using all combinations of the following factors:

1) *Concrete vs Abstract* – whether or not the subgraph abstraction was used.
2) *Forward vs Informed Search* – whether the search was in temporal order or informed by variable domains.
3) *Complete vs Prioritised* – whether or not prioritised planning was used.

The Gecode constraint solver was used for all eight approaches to ensure that the results were comparable.

To simulate forward search the variables representing the state at time t were only created and constrained once the variables representing time $t - 1$ were bound. A depth-first search was then performed, stopping when the goal positions were reached.[3]

The informed search used an iterative deepening approach. 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.

Prioritised planning was performed by building a succession of CSPs $C_1, \ldots, C_k$, with C_i representing the plans for robots $\{1, \ldots, i\}$. In C_{i+1} the plans for robots $1, \ldots, i$ were constrained to contain the same sequence of transitions as in C_i.

For every approach there was an upper time limit of 10s placed on search. If a solution could not be found within this time then the search was deemed to have failed.

B. Results

One hundred different experiments were conducted for each approach and each number of robots.[4] Success rates are plotted in Figure 3. It is clear that the informed search on the complete CSP is much more successful than traditional forward search in all categories. The forward search begins to fail (running out of time) for only a small number of robots while the informed search on the abstract representation

[3]A best-first search using a relaxed distance metric (ignoring collisions) was also performed with comparable results.

[4]Experiments were run on a 2.16GhZ Intel Core Duo with 2GB of memory.

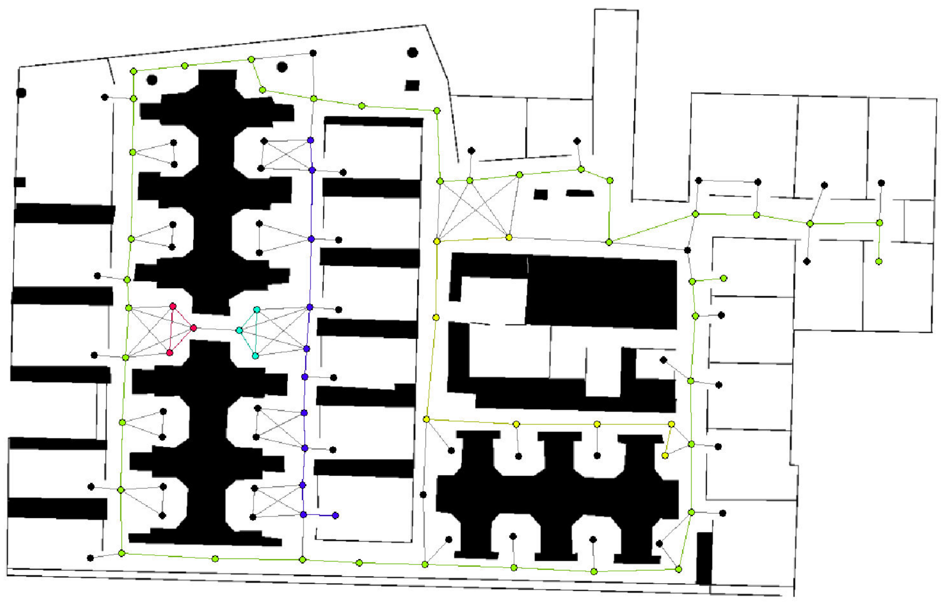


Fig. 2. The floorplan of the AI Lab with the corresponding roadmap. Subgraphs are indicated by colours.

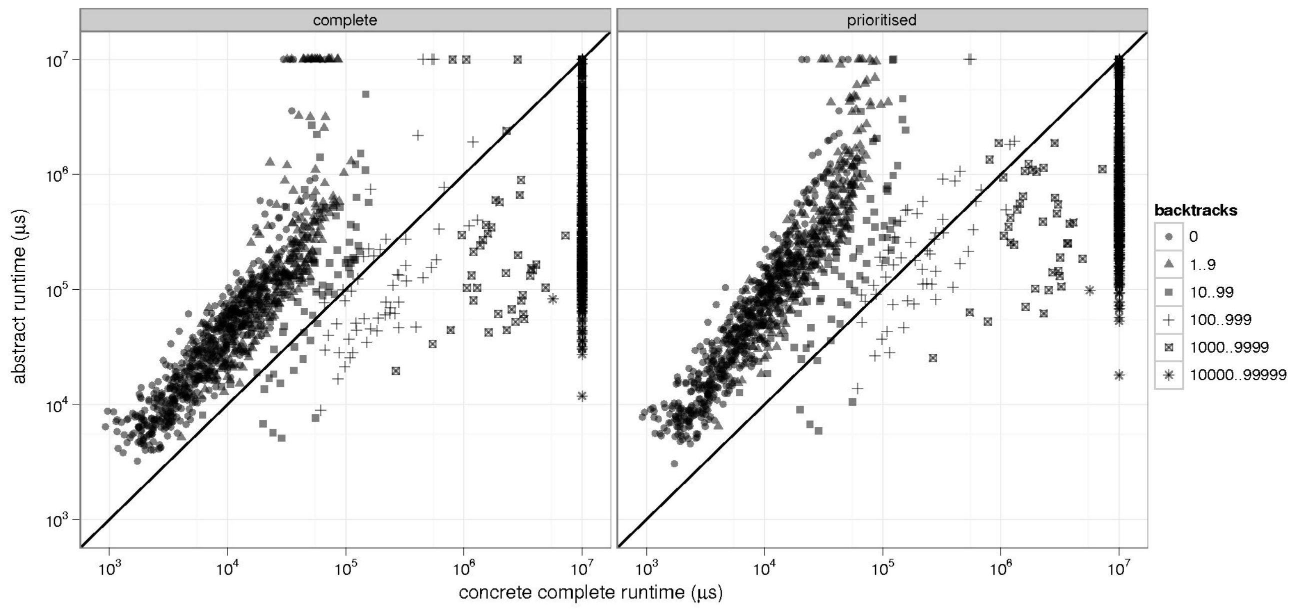


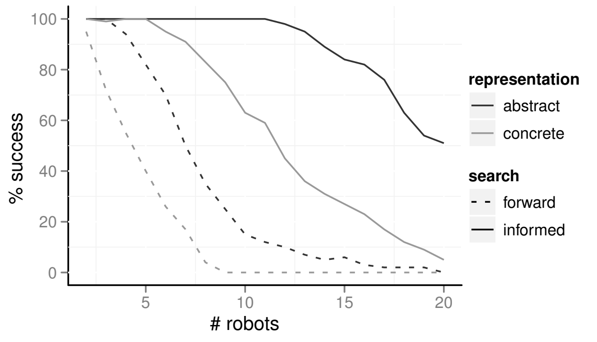
Fig. 4. A comparison of search times for the concrete and abstract representations. The plot symbol shows the number of backtracks made by the concrete planner in each case.

shows 100% success for as many as 11 robots with complete search and 13 robots with prioritised.

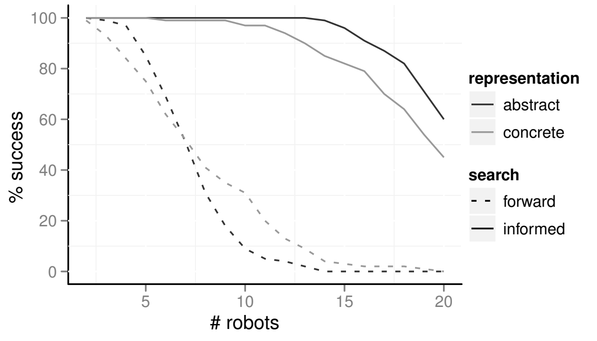
The graphs also show a general superiority of abstract planning over concrete. An examination of run times (Figure 4) gives a clearer picture. These two graphs plot the concrete and abstract search times for each problem using informed search, with a diagonal line indicating equality. We can identify clusters of problems which we will call "easy" and "hard", based on the performance of the complete concrete planner. Easy problems require fewer than 100 backtracks. while hard problems take 100 or more. Under this division 51% of problems are easy and 49% are hard. Figure 5 shows how the problem difficulty varies with the number of robots. While the difficulty increases predictably with the number of robots, it is worth noticing that there are some difficult problems even for small numbers of robots.[5]

The graphs for the complete and prioritised planners both

[5]The drop off in the number of backtracks for cases with more than 14 robots is not because the problems are getting easier, but because the planner is running out of time.



(a) Complete



(b) Prioritised

Fig. 3. Success rates for different approaches to the planning problem.

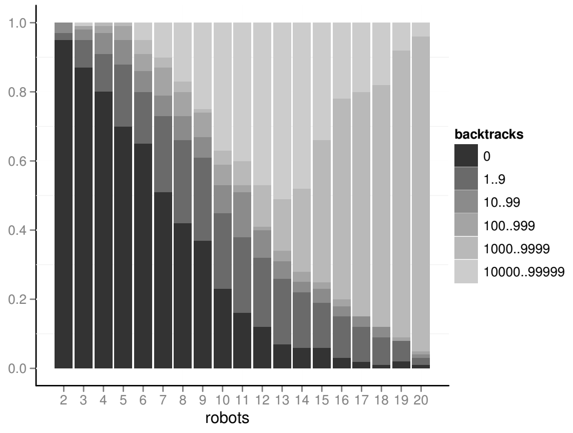


Fig. 5. Problem difficulty: As the number of robots increases the complete concrete planner backtracks more often.

show a significant performance difference between easy and hard problems. The large clusters above the diagonal on the left hand side of each graph consist of primarily easy problems. In these cases, the two abstract planners take significantly longer than the concrete planner, due to the overhead of additional constraints. The log-log scale here is deceptive. Calculating lines of best fit, we see that the search time of the complete abstract planner is about 22.9 times that of the concrete planner, and the prioritised planner is worse, taking 26.6 times as long.

On the other hand, the performance on hard problems is much better. The complete abstract planner takes only 0.29 of the time of the concrete planner, and the prioritised planner is even faster at 0.26 of the time.

It is apparent from these results that the abstraction offers the ability to successfully solve more problems and to solve difficult problems quickly, at the expense of a significant overhead for easier problems.

VIII. Conclusion

We have demonstrated how the multi-robot path planning problem can be encoded as a constraint satisfaction problem and solved using a combination of constraint propagation and heuristic search. These techniques allow us to solve problems of much greater size than traditional forward-search techniques.

For small problems, we have found that a simple CSP representing the concrete planning problem is efficient but as the number of robots grows, the rapid increase in backtracking makes this representation untenable. A more effective representation has been presented which makes use of structural knowledge about the map, in the form of a subgraph decomposition. This knowledge allows the planner to focus the early stages of the search on the critical subgraph transitions and fill in the concrete details later. This approach has significant overheads for small problems but proves worthwhile when dealing with large numbers of robots.

A. 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 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 [12]. 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.

Automating the subgraph partitioning process would obviously be desirable. Work on this problem is already underway with promising preliminary results.

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

[1] M. R. K. Ryan, "Exploiting subgraph structure in multi-robot path planning," *Journal of Artificial Intelligence Research*, vol. 31, pp. 497–542, 2008.

[2] A. Botea, M. Müller, and J. Schaeffer, "Using abstraction for planning in sokoban," in *Computers and Games: Lecture Notes in Computer Science*. Springer, 2003, vol. 2883, pp. 360–375.

[3] A. Junghanns and J. Schaeffer, "Sokoban: Enhancing general single-agent search methods using domain knowledge," *Artificial Intelligence*, vol. 129, no. 1-2, pp. 219–251, 2001.

[4] P. van Beek and X. Chen, "CPlan: A constraint programming approach to planning," in *Proceedings of the AAAI National Conference*, 1999, pp. 585–590.

[5] A. Blum and M. Furst, "Fast planning through planning graph analysis," *Artificial Intelligence*, vol. 90, no. 1-2, pp. 281–300, 1997.

[6] M. Do and S. Kambhampati, "Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP," *Artificial Intelligence*, vol. 132, no. 2, pp. 151–182, 2001.

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

[8] H. Kautz, B. Selman, and J. Hoffmann, "SatPlan: Planning as Satisfiability," in *Abstracts of the 5th International Planning Competition*, 2006.

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

[10] P. Surynek, "A novel approach to path planning for multiple robots in bi-connected graphs," in *Proceedings of the IEEE International Conference on Robotics and Automation*. Kobe, Japan,: IEEE Press, May 2009, pp. 1–8.

[11] Gecode Team, "Gecode: Generic constraint development environment,," Available from http://www.gecode.org., 2006. [Online]. Available: http://www.gecode.org

[12] M. Mann, G. Tack, and S. Will, "Decomposition during search for propagation-based constraint solvers," Cornell University Library, Tech. Rep. arXiv:0712.2389v2, 2007. [Online]. Available: http://arxiv.org/abs/0712.2389v2