Constraint-Based Multi-robot Path Planning with Subgraphs

Malcolm R. K. Ryan

malcolmr@cse.unsw.edu.au
ARC Centre of Excellence for Autonomous Systems
University of New South Wales, Australia

Technical Report
UNSW-CSE-TR-1108
May 2011

THE UNIVERSITY OF
NEW SOUTH WALES

School of Computer Science and Engineering
The University of New South Wales
Sydney 2052, Australia

Abstract

Coordinating a group of robots as they independently navigate a shared map without collision is a difficult planning problem. Traditional approaches have scaled badly as they pay little attention to the structure of the underlying search space and waste time exploring parts of the space that are never going to yield a solution. We would like to be able to eliminate these branches early in the search, but a naive representation of the problem does not provide enough information to allow this.

We present an alternative formulation of the task as a constraint satisfaction problem with a temporally and spatially abstract representation that allows search to focus on critical decisions early in the search and recognise untenable branches sooner. This representation is based on a partitioning of the map into subgraphs of particular structure, *cliques* and *halls*, which place easily representable constraints on the movement of their occupants. We plan first at the level of subgraph transitions and only choose concrete moment-to-moment positions once this abstract plan is complete. Experimental evaluation shows that this allows us to create plans for many more robots that a traditional approach.

Further, we show how these map partitions can be automatically generated using the betweenness property of the vertices to detect bottlenecks in the graph and turn them into *hall* subgraphs. This method is evaluated on 100 different maps and shown to be most effective on indoor maps with long branching corridors.

1 Introduction

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 (Barraquand & Latombe, 1991) soon becomes intractable for even a moderate number. Decoupled approaches, such as prioritised planning (LaValle & Hutchinson, 1998) or Cooperative A* (Silver, 2005), are much faster but lack completeness. In confined spaces they can often fail to find an available solution.

The fault with these traditional approaches lies in the search order. The critical parts of the plan, where the robots interact, are often left undetermined until late in the planning process. An early mistake, such as two robots entering a narrow hall in the wrong order, may not cause failure until much deeper in the search tree and a lot of time can be wasted backtracking before the decision is revised. This is a violation of the familiar Fail-First Principle (Haralick & Elliott, 1980) but a simple change of search ordering will not solve the problem. The standard concrete representation of the problem is not expressive enough. A richer representation with both temporal and spatial abstraction are required to be able to recognise and repair these mistakes as they occur.

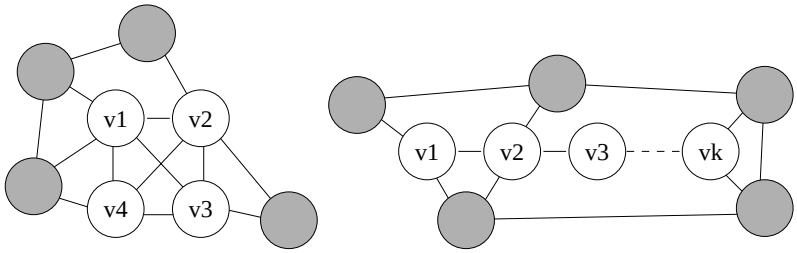
Previous work has demonstrated the advantage of planning using a subgraph abstraction (Ryan, 2008). We first partition the map into subgraphs of particular known structure, such as *cliques* and *halls* (Figure 2.1), and build abstract plans which describe the transitions of robots between the subgraphs. This abstraction is effective because it allows us to represent and propagate temporally and spatially abstract constraints without having to commit to unnecessary concrete details. 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 but at the cost of a time-overhead on simple cases.

We also look at the problem of partition generation. Hand-partitioning a single map is relatively easy, but for applications involving many maps it soon becomes a burden. We present a novel method to automatically decompose maps into halls – long chains of singly-connected vertices – based on the 'betweenness' measurement of centrality in a graph (Freeman, 1977). Vertices with a high betweenness are likely to be bottlenecks in a plan. Connecting them as halls allows us to impose an order on the robots that pass through these bottlenecks in the early stages of planning and thus avoid unnecessary backtracking.

2 Subgraph Planning

We can formalise the multi-robot planning problem as follows. A road-map is given in the form of a graph $G = (V, E)$ representing the connectivity of free space for a single robot moving around the world. We have a set of robots $R = \{r_1, \ldots, r_k\}$ which we shall consider to be homogeneous, so a single map



(a) A clique (b) A hall

Figure 2.1: Types of subgraphs.

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 in other vertices of 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 2.1.[2]

A clique is a complete subgraph with each vertex linked to every other. In maps they usually represent large open spaces with many entrances and exits. The configuration of a clique ignores the exact positions of the robots and only records 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.

3 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

[1] This is also known as the Pebble Motion on Graphs problem (Surynek, 2009).

[2] In previous work we also included a *ring* subgraph (a hall with linked ends) but we have not yet been able to find an efficient constraint representation for this structure.

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 $\mathcal{H} \subseteq S$ be the set of halls and $\mathcal{C} \subseteq S$ but the set of cliques. 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.

We now provide two constraint representations of this planning problem, one for the simple concrete problem and one for the abstracted version. For both approaches, we include an extension that allows us to implement prioritised planning with little extra effort.

3.1 Representing Concrete Plans

We can now define the variables we need. Let l be the length of the concrete plan. For each robot $r \in R$ and each step of the plan $t \in \{1 \ldots l\}$ we define $\mathbf{P}_t[r] \in V$ to be the position of robot r at time t. We constrain these variables as follows:

Robots can only move between neighbouring vertices.

$$\mathbf{P}_t[r] \neq \mathbf{P}_{t+1}[r] \rightarrow (\mathbf{P}_t[r], \mathbf{P}_{t+1}[r]) \in E \tag{3.1}$$

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

$$\mathit{distinct}(\mathbf{P}_t[1], \ldots, \mathbf{P}_t[k]) \tag{3.2}$$

A robot can only enter a vertex if it was previously vacant.

$$\mathbf{P}_t[r_x] \neq \mathbf{P}_{t+1}[r_y], \forall r_x \neq r_y \tag{3.3}$$

No-ops only occur at the end of the plan. (Symmetry breaking.)

$$(\forall r \in R : \mathbf{P}_{t-1}[r] = \mathbf{P}_t[r]) \rightarrow (\forall r \in R : \mathbf{P}_t[r] = \mathbf{P}_{t+1}[r]) \tag{3.4}$$

Prioritisation

In prioritised planning we assign a priority ordering to the robots and plan for them in sequence from high to low priority. Once a robot's plan is assigned no further backtracking on that plan is permitted, so lower priority robots must plan to avoid higher priority robots. Of course this means the search is incomplete but it often results in much faster planning. To represent this in our CSP we shall assume we have already found a plan for robots $1, \ldots, k-1$ with length l' and position values $p_t[r]$. We now construct a new plan for robots $1, \ldots, k$ as above, but constrain the positions of robots $1, \ldots, k-1$ to follow the previous plan. The new plan may be longer that the old, so we need to include additional variables to allow us to 'stretch' the old plan.

[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 a maximum length or time limit is reached.

Create index variables $\mathbf{I}_t \in \{1, \ldots, l'\}, \forall t \in \{1, \ldots, l\}$ with the constraints:

Indices are in sequence.

$$\mathbf{I}_t \leq \mathbf{I}_{t+1} \leq \mathbf{I}_t + 1 \tag{3.5}$$

The first and last indices are defined.

$$\mathbf{I}_1 = 1 \tag{3.6}$$
$$\mathbf{I}_l = l' \tag{3.7}$$

The positions of the higher priority robots are constrained.

$$\mathbf{P}_t[r] = p_{\mathbf{I}_t}[r], \forall r \in \{1, \ldots, k-1\} \tag{3.8}$$

Search

With this representation planning becomes a simple matter of giving values to the position variables. By enumerating all the variables in advance we have the option of assigning them values in whatever order we wish, rather than the standard temporal ordering employed in forward search. The method we used was to choose the most constrained position variable at each point of the search and assign it the value of the vertex closest to that robot's next assigned position (based on a pre-computed single-robot all-shortest-paths matrix). Since the ends of the plan are more constrained than the middle, this usually results in the plan growing from either end towards the centre.

By calculating the single-robot shortest path lengths for each robot we can determine a lower bound for the length of the plan. From this starting point iterated depth-first search can be used to find a solution to the multi-robot problem.

3.2 Representing Abstract Plans

The representation of abstract plans follows much the same formulation as that of concrete plans. We have an array of variables representing the position (now a subgraph) of each robot at each time-step and we constrain the transitions between them. Only now we also need to consider the entry and exit vertices used on each transition, as this affects the configuration of the subgraph. Extra variables are required:

$\mathbf{A}_i[r] \in S$ is the index of the subgraph occupied by r at step i,

$\mathbf{F}_i[r] \in V$ is the index of the first vertex occupied by r at step i,

$\mathbf{T}_i[r] \in V$ is the index of the last vertex occupied by r at step i.

We now constrain these variables as follows:

Robots can only move between neighbouring subgraphs.

$$\mathbf{A}_i[r] \neq \mathbf{A}_{i+1}[r] \rightarrow (\mathbf{A}_i[r], \mathbf{A}_{i+1}[r]) \in \mathcal{E} \tag{3.9}$$

$\mathbf{F}_i[r]$ and $\mathbf{T}_i[r]$ must belong to the given subgraph.

$$\mathbf{A}_i[r] = a \rightarrow \mathbf{F}_i[r] \in V_a \tag{3.10}$$
$$\mathbf{A}_i[r] = a \rightarrow \mathbf{T}_i[r] \in V_a \tag{3.11}$$

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

$$\mathit{distinct}(\mathbf{F}_i[1], \ldots, \mathbf{F}_i[k]) \tag{3.12}$$
$$\mathit{distinct}(\mathbf{T}_i[1], \ldots, \mathbf{T}_i[k]) \tag{3.13}$$

Consecutive sub-plans are linked by valid transitions.

$$(\mathbf{T}_i[r], \mathbf{F}_{i+1}[r]) \in E \tag{3.14}$$
$$\mathbf{T}_i[r_x] \neq \mathbf{F}_{i+1}[r_y], \forall r_x \neq r_y \tag{3.15}$$

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

$$(\exists r \in R : \mathbf{A}_i[r] \neq \mathbf{A}_{i+1}[r]) \rightarrow (\exists r \in R : \mathbf{A}_{i-1}[r] \neq \mathbf{A}_i[r]) \tag{3.16}$$

If a subgraph is full, its occupants cannot move.

$$\mathbf{A}_i[r] = a \ \wedge \ \mathit{count}_{\rho \in R}(\mathbf{A}_i[\rho] = a) = |V_a| \rightarrow \mathbf{F}_i[r] = \mathbf{T}_i[r] \tag{3.17}$$

Constraint 3.9 is redundant, as it is implied by the latter constraints, but it is included to make propagation more efficient. It allows the subgraph adjacency restrictions to be propagated even when the $\mathbf{F}_i[r]$ and $\mathbf{T}_i[r]$ variables are unassigned.

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 $(\mathbf{T}_i[1], \ldots, \mathbf{T}_i[k])$ is in the same order as $(\mathbf{F}_i[1], \ldots, \mathbf{F}_i[k])$. To ensure this we must add further constraints.

3.3 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 in front 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 $\mathbf{F}_i[r_x] < \mathbf{F}_i[r_y] \Leftrightarrow \mathbf{T}_i[r_x] < \mathbf{T}_i[r_y]$.

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: $\mathbf{Ord}_i[r_x, r_y] \in \{-1, 0, 1\}$. 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 $\mathbf{Ord}_i[r_x, r_y]$ 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. We express this formally with the following constraints:

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

$$\mathbf{A}_i[r_x] \in \mathcal{H} \ \wedge \ \mathbf{A}_i[r_x] = \mathbf{A}_i[r_y] \Leftrightarrow \mathbf{Ord}_i[r_x, r_y] \neq 0 \tag{3.18}$$

Ordering variables affect concrete positions.

$$\mathbf{Ord}_i[r_x, r_y] = -1 \quad \rightarrow \quad \mathbf{F}_i[r_x] < \mathbf{F}_i[r_y] \wedge \mathbf{T}_i[r_x] < \mathbf{T}_i[r_y] \tag{3.19}$$
$$\mathbf{Ord}_i[r_x, r_y] = 1 \quad \rightarrow \quad \mathbf{F}_i[r_x] > \mathbf{F}_i[r_y] \wedge \mathbf{T}_i[r_x] > \mathbf{T}_i[r_y] \tag{3.20}$$

Ordering variables persist across time steps.

$$\mathbf{A}_i[r_x] = \mathbf{A}_{i+1}[r_x] \wedge \mathbf{A}_i[r_y] = \mathbf{A}_{i+1}[r_y] \rightarrow \mathbf{Ord}_i[r_x, r_y] = \mathbf{Ord}_{i+1}[r_x, r_y] \quad (3.21)$$

This completes our description. Any abstract plan which satisfies these constraints is guaranteed to have a valid concrete resolution.

Prioritisation

Prioritisation of abstract plans can be implemented in the same manner as for concrete plans, using a set of index variables to connect the abstract position variables $\mathbf{A}_i[r]$ to their values from an earlier plan. Note however that we do not need to find a concrete resolution of the earlier plan before adding the extra robot. As we will see later, this makes prioritised abstract plans much more flexible than their concrete equivalents.

Search

We now have three sets of variables to assign: the position variables $\mathbf{A}_i[r]$, the order variables $\mathbf{Ord}_i[r_x, r_y]$ and that transition variables $\mathbf{F}_i[r]$ and $\mathbf{T}_i[r]$. For any particular robot and time step, it makes sense to assign them in this order, since each is strongly constrained by the one that comes before. In our experiments we found that the best approach was:

1. If there is a step i and robots r_x and r_y such that $\mathbf{A}_i[r_x] = \mathbf{A}_i[r_y]$ and $\mathbf{Ord}_i[r_x, r_y]$ is unassigned, then assign $\mathbf{Ord}_i[r_x, r_y]$,

2. Otherwise choose the unassigned position variable $\mathbf{A}_i[r]$ with the smallest domain.

3. Once all the $\mathbf{Ord}_i[r_x, r_y]$ and $\mathbf{A}_i[r]$ variables have been assigned, choose the unassigned transition variable $\mathbf{F}_i[r]$ or $\mathbf{T}_i[r]$ with the smallest domain. It is only necessary to assign values to $\mathbf{F}_i[r]$ or $\mathbf{T}_i[r]$ if the robot is respectively entering or leaving the subgraph on this step. Otherwise they can be left unassigned until the resolution phase (below).

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 $\mathbf{A}_i[r]$ variables we apply the first principle by choosing the subgraph which is closest to the next assigned subgraph for robot r (based on a pre-computed 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 $\mathbf{Ord}_i[r_x, r_y]$ is to consider the concrete values that it immediately affects $\mathbf{F}_i[r_x]$, $\mathbf{T}_i[r_x]$, $\mathbf{F}_i[r_y]$ and $\mathbf{T}_i[r_y]$. 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 $\mathbf{F}_i[r_x]$ and $\mathbf{T}_i[r_x]$ are chosen to minimise the distance between the beginning and end of the plan step.

Once again we can compute single-robot shortest-path distances to obtain a lower bound for the length of the plan. From this starting point we employ iterated depth-first search to find a complete abstract plan.

Resolution

Once the plan is found we need to resolve it into a concrete plan. In previous work, hand-written planners were provide for each subgraph type to construct concrete sub-plans for each step of the abstract plan without the need for further search. While this method is possible, it turns out that the resolution problem is well enough constrained that it can be solved just as efficiently by a standard constraint solver.

For each step i of the abstract plan we construct a concrete sub-plan as described in section 3.1 above, with the additional constraints:

Source and destination match transition variables.

$$\mathbf{P}_1[r] = \mathbf{F}_i[r] \tag{3.22}$$
$$\mathbf{P}_l[r] = \mathbf{T}_i[r] \tag{3.23}$$

Ordering constraints apply to every time-step.

$$\mathbf{Ord}_i[r_x, r_y] = -1 \rightarrow (\forall t : \mathbf{P}_t[r_x] < \mathbf{P}_t[r_y]) \tag{3.24}$$
$$\mathbf{Ord}_i[r_x, r_y] = 1 \rightarrow (\forall t : \mathbf{P}_t[r_x] > \mathbf{P}_t[r_y]) \tag{3.25}$$

The same search techniques as described in Section 3.1 above can be applied to solve this problem. This can almost always be done without any backtracking, since we are planning on a much simpler subgraph.

4 Experiment 1: The K17 Map

To evaluate this new planning system we have applied it to a realistic planning problem. Figure 4.1 shows a map of the AI Laboratory at UNSW. The map is an undirected graph of 113 vertices and 154 edges, 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.

4.1 Approach

The map was populated with a number of robots which varied from 2 to 20. Each robot was assigned a random initial and goal location. 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.

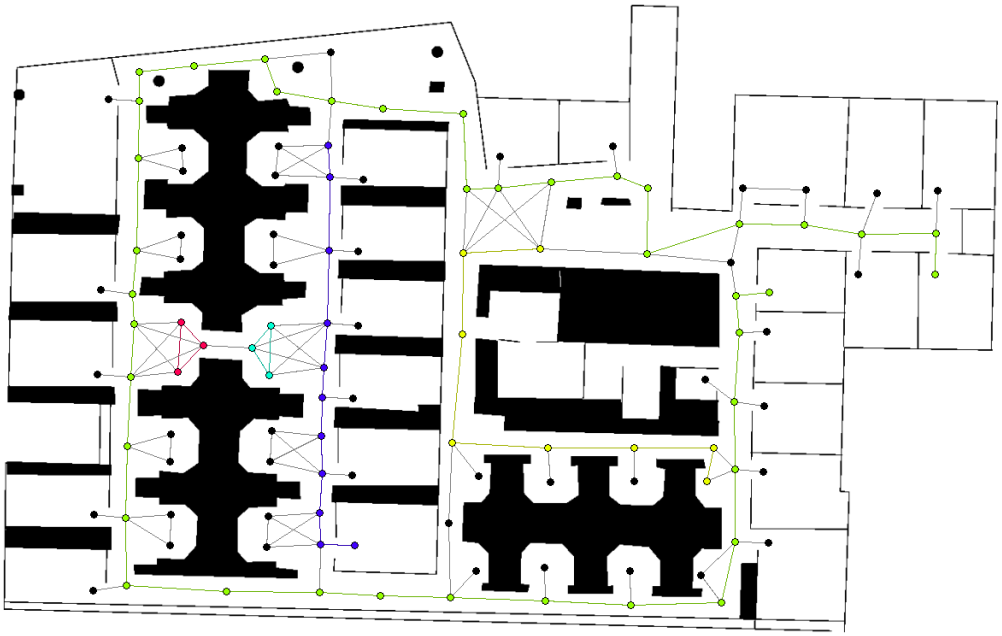


Figure 4.1: The floor-plan of the AI Lab with the corresponding roadmap. Subgraphs are indicated by colours.

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.

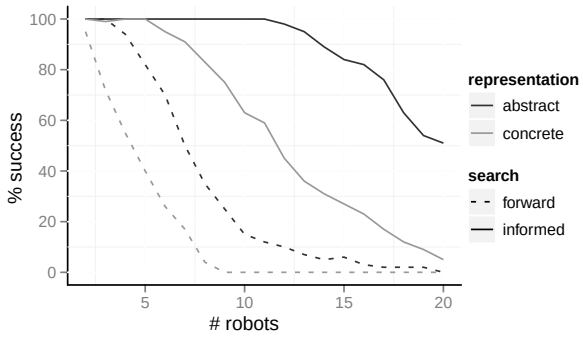
The Gecode constraint solver was used to implement all eight approaches to ensure that the results were comparable. To simulate forward search the variables representing the state at time t were only created and constrained once the variables representing time $t-1$ were bound. A depth-first search was then performed, stopping when the goal positions were reached.[1]

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.

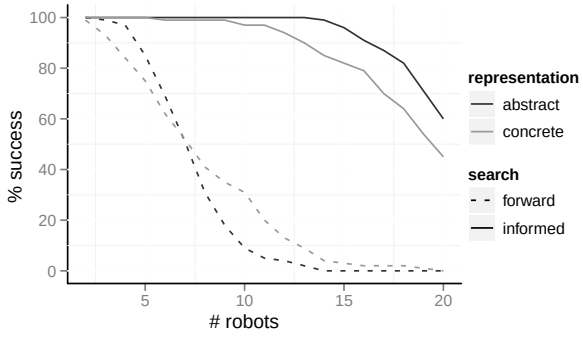
Prioritised planning was performed by building a succession of CSPs $C_1, \ldots, C_k$, with C_i representing the plans for robots $\{1, \ldots, i\}$. In C_{i+1} the plans for robots $1, \ldots, i$ were constrained to contain the same sequence of transitions as in C_i.

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

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



(a) Complete



(b) Prioritised

Figure 4.2: Success rates for different approaches to the planning problem.

4.2 Results

One hundred different experiments were conducted for each approach and each number of robots.[2] Success rates are plotted in Figure 4.2. It is clear from these graphs 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 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 4.3) 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 4.4 shows how the problem difficulty varies with the number of robots. While the difficulty increases pre-

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

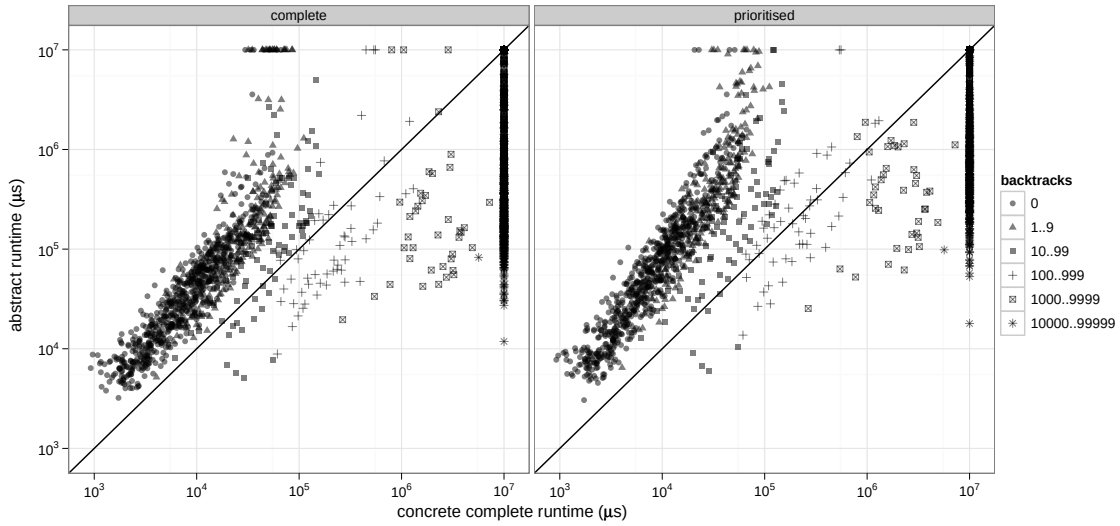


Figure 4.3: A comparison of search times for the concrete and abstract representations. The plot symbol shows the number of backtracks made by the concrete planner in each case.

dictably with the number of robots, it is worth noticing that there are some difficult problems even for small numbers of robots.[3]

The graphs for the complete and prioritised planners both 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.

5 Partitioning Maps

These results are encouraging, but the question remains of how to construct the map partition in the first place. For a single small map it might be done manually, but in some applications the number of maps may be in the hundreds or even thousands. For example, the computer game *Fallout 3* by Bethesda

[3]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.

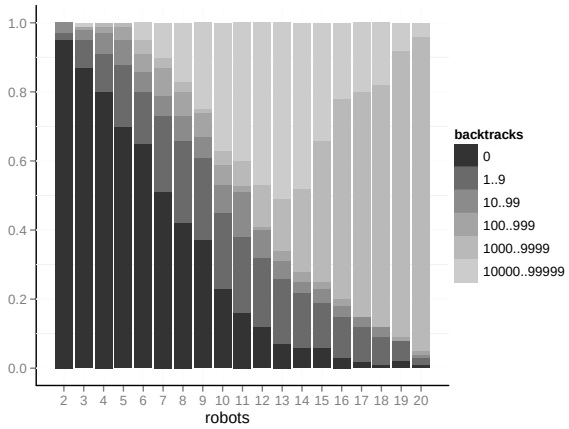


Figure 4.4: Problem difficulty: As the number of robots increases the complete concrete planner backtracks more often.

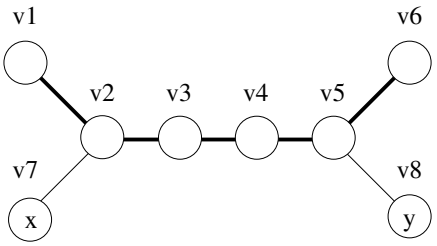


Figure 5.1: A simple roadmap with a bottleneck. The dark lines represent a partitioning in which the bottleneck is represented as a hall. The two robots x and y wish to exchange positions.

Softworks (2008) contains over 7000 individual maps of size up to a thousand vertices. Hand partitioning of these maps would be completely impractical; an automated approach is essential

A good subgraph decomposition is one that focuses the early choices in the search for a plan on the critical parts of the plan, so that failures are encountered early and less time is wasted backtracking. In the multi-robot planning problem the critical regions are the shared vertices and edges in the robots' paths, so our partition is designed to focus search on these bottlenecks first.

The hall subgraph is useful for representing narrow bottlenecks as the $\mathbf{Ord}_i[r_x, r_y]$ variables explicitly determine an ordering on its occupants and ensure that this ordering is maintained as robots enter and leave, without having to commit to their concrete positions. Once it has been decided that the robot needs to pass through a particular bottleneck (hall) these variables can be instantiated to ensure that the order of transitions is valid.

For example, consider the situation in Figure 5.1. The two robots x and y wish to exchange positions, but must pass through the intervening bottleneck. In a concrete search, much time will be wasted trying various alternative paths unsuccessfully. By representing the bottleneck as a hall, these impossible paths

are immediately pruned by constraint propagation. If the two robots enter the hall (indicated with bold edges) simultaneously then their ordering is established with x before y. For y to exit to v_7 it must leave through v_2. We can immediately deduce that x must move to v_1 without the need for search.

The approach we have taken in this paper is to try to find a single partitioning of the map that will provide good results for many different planning problems on that map. In case we can reasonably spend significant time precomputing the partition in the knowledge that it will be amortised over many planning problems, but we lose specific information about the starting and goal locations for particular problems. Instead we investigate general properties of the roadmap graph, looking particularly at the 'betweenness' measure.

6 Bottleneck Detection with Betweenness

Betweenness is a centrality measure of a vertex in a graph, equal to the proportion of all shortest paths between pairs of nodes in the graph that pass through that vertex. Formally:

$$B(v) = \sum_{s,t \in V, s \neq v, t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Where σ_{st} is the number of shortest paths from s to t and $\sigma_{st}(v)$ is the number of these that pass through v. Calculating betweenness for an entire graph takes $O(VE)$ time using Brandes' algorithm (Brandes, 2001).

If we consider a random selection of starting locations and goals, then the betweenness of a vertex provides an estimate of the proportion of robots that are likely to pass through that vertex. Vertices with high betweenness are suitable candidates for hall construction, for two reasons: 1) being heavily used they are likely to be sources of contention and putting them in a hall will help control this, and 2) as they lie on many shortest paths, combining them into subgraphs with their neighbours is likely to reduce the average path length significantly. Both of these factors can contribute towards faster planning.

Thus we take a simple greedy approach to creating halls, as shown in Algorithm 1. The betweenness of every vertex is computed. The vertex with the highest value is selected as the seed for a hall. Among its neighbours the vertex with the next highest betweenness is selected and added to the hall. Further vertices are added to either end of the hall always selecting the one with the highest betweenness (and making sure that the hall does not contain a loop). When no more vertices are available, the hall is complete. These vertices are removed from consideration and the process is repeated to create as many halls as possible, until all vertices have been assigned subgraphs.

7 Experiment 2: The Fallout Maps

Fallout 3 is a post-apocalyptic computer role-playing game in which the player explores a wide variety of environments, both indoor and outdoor. For the purposes of this research we are interested in it only as a source of a large number of realistic maps of different sizes and structures. The world of the game is impressively large and its landscape and interior architecture are laid

Algorithm 1 Betweenness-based hall partitioning.

```
1: function PARTITION(G = ⟨V, E⟩, B)
2:     U ← V
3:     P ← ∅
4:     while U ≠ ∅ do
5:         v_h ← arg max_{v∈U} B(v)
6:         v_t ← v_h
7:         H ← ⟨⟩
8:         N ← {v ∈ U | (v, v_h) ∈ E}; N_h ← N; N_t ← ∅
9:         U ← U − v_h
10:        while N ≠ ∅ do
11:            v ← arg max_{v∈N} B(v)
12:            U ← U − v
13:            if v ∈ N_h then
14:                H ← v_h.H
15:                v_h ← v
16:            else
17:                H ← H.v_t
18:                v_t ← v
19:            end if
20:            N_h ← {v ∈ U | (v, v_h) ∈ E ∧ ¬∃u ∈ H ∪ {v_t} : (v, u) ∈ E}
21:            N_t ← {v ∈ U | (v, v_t) ∈ E ∧ ¬∃u ∈ H ∪ {v_h} : (v, u) ∈ E}
22:            N ← N_h ∪ N_t
23:        end while
24:        if H ≠ ∅ then
25:            Add hall v_h.H.v_t to P
26:        else
27:            Add singleton v_h to P
28:            Add singleton v_t to P
29:        end if
30:    end while
31:    return P
32: end function
```

out quite realistically, so it provides a good sample of the kinds of 'real-world' structures that we hope to exploit. Furthermore the work of building road-maps has already been done for us. Each map has been partitioned into a 'navigation mesh' (a triangular cell-decomposition) which is used for planning the paths of AI characters in the game.

We took a sample of 100 maps from the game, chosen to provide a distribution across the range of graph size in terms of both the number of vertices (from 100 to 2000) and the average degree (from 2 to 4) of the graph (Figure 7.1). This is not a uniform sampling – the preponderance of maps in the game have fewer than 500 vertices and there are few graphs at either extreme of degree – but it gives us a wide variety of maps with which to test our planner. In particular it includes a mixture of indoor and outdoor maps.

The partitioning algorithm was applied to the 100 selected maps. We then used these maps in two experiments, one to compare abstract planning against concrete and one to compare the heuristic against random partitions.

In the experiments that follow, we will show that the *dimension* of a graph is a good predictor of the performance of our algorithms. This is a graph property

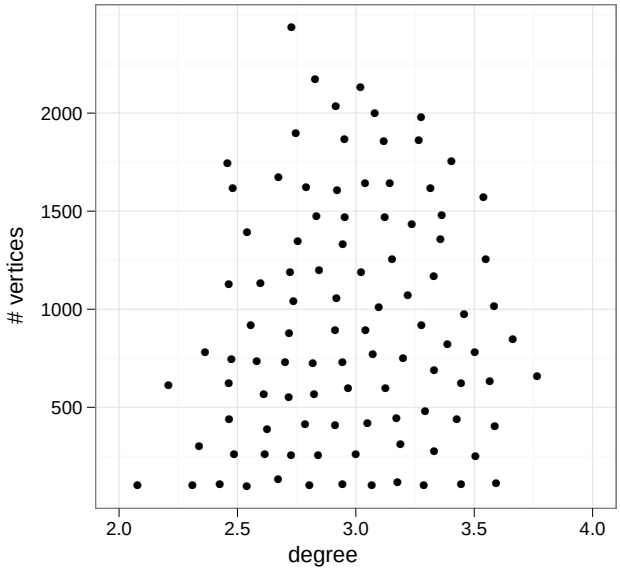


Figure 7.1: The distribution of size and average degree of the 100 maps used in this study.

of our own invention which describes the overall shape of the graph. We define the dimension of a graph $G = \langle V, E \rangle$ as:

$$D(G) = \frac{log(|V|)}{log(diameter(V))}$$

A long singly-linked chain has $D(G) \approx 1$, a square mesh has $D(G) \approx 2$, a cubic mesh has $D(G) \approx 3$ and so on. The *Fallout* maps have dimensions ranging from 1.05 to 1.8.

7.1 Concrete vs. Abstract Search

First we examine the performance of the abstract planning algorithm in comparison to a standard concrete planner. We selected 100 different planning tasks for each map; ten robots were placed at random starting locations and assigned random goals. We then applied four different approaches to each problem:

1. Complete concrete planning on the original roadmap,
2. Complete abstract planning on the roadmap partitioned using the betweenness heuristic,
3. Prioritised concrete planning on the original roadmap,
4. Prioritised abstract planning on the roadmap partitioned using the betweenness heuristic.

All four approaches were implemented as constraint problems and solved using the Gecode CSP solver, using depth-first search guided by the single-robot

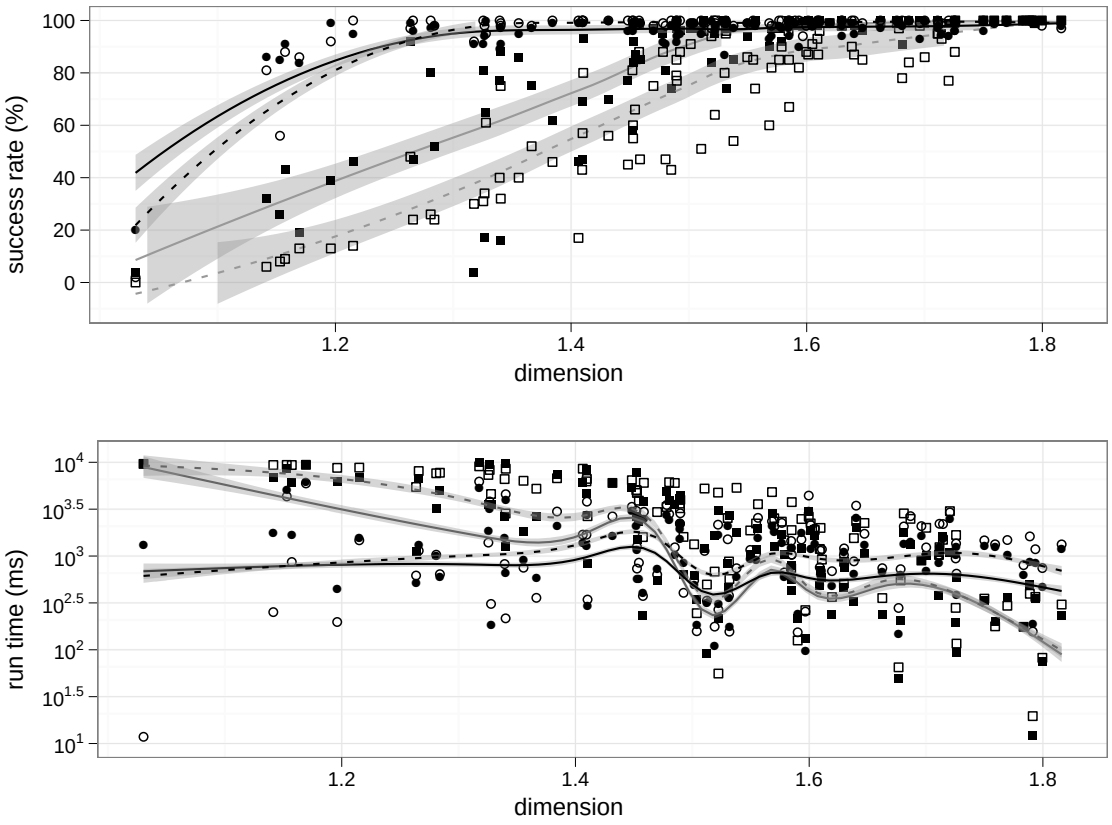


Figure 7.2: A comparison of success rates and run times for Experiment 1. Dark lines and round points are used for the abstract planner, grey lines and square points for the concrete planner. Hollow points and dashed lines indicate prioritised planning. Solid lines and filled points indicate complete planning. Lines of best fit have been calculated using local polynomial regression fitting with the shaded area showing a 95% confidence interval.

shortest-path heuristic. A time limit of 10 seconds was placed on each planner. If a plan could not be found in this time, the planner was deemed to have failed.[1]

Results

The results are shown in Figure 7.2. The performance of the four different approaches varies predictably with dimension. In general, low dimensional graphs present harder planning problems as there are fewer paths between vertices and therefore more potential conflicts between robots. Performance varied from one map to another but an overall trend is clear. In terms of the number of problems successfully solved, the both abstract planners performed comparable well, outperforming the concrete planners by a significant margin on the harder problems. The prioritised concrete planner was the poorest performer, even failing

[1]These experiments were run on 3.00GHz Intel Pentium machines.

up to 20% of the time on some high-dimensional problems.

The average run-times (Figure 7.2b) show a more complex picture. For low dimension problems the abstract planners are have a much lower average runtime, with the same ranking of performance as we see above. However as the dimension increases, the concrete approaches become more successful and we see the ranking change. The abstract planners takes significantly longer than the concrete. The overhead of two-pass planning is greater than any reduction in search that it may yield.

Discussion

The dimensionality of a graph provides a clear indication of how difficult a planning problem it entails. A low-dimensional graph with long narrow corridors presents a significant co-ordination problem for a large group of robots. An open space with many alternative plan is much easier to solve. For easy problems, the concrete planner is just as successful as the abstract and usually more efficient, but in harder problems the hall abstraction becomes very valuable.

The incompleteness of prioritised planning is particular problematic in low-dimensional spaces. A bottleneck will often require a particular ordering on the robots that pass through it, especially if one of the robots terminates within the bottleneck. In ordinary concrete prioritised planning, if a high-priority robot terminates at such a location then it may become impossible for lower priority robots to pass through and reach their goals.

Prioritised abstract planning avoids this situation. A high-priority robot does not commit to terminating in a particular vertex. Rather, it terminates within the hall containing its goal. It is still free to move about within the hall so as to let other robots pass by. So long as there is enough room within the hall for it to get out of the way, lower priority robots can still reach their destinations. This makes abstract prioritised planning more flexible while maintaining the benefits of a smaller search space.

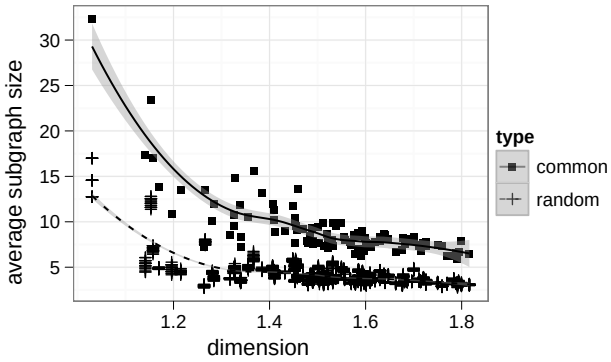
7.2 Heuristic vs. Random Partitions

For the sake of comparison, twenty alternative partitions were created using the same algorithm but a table of random values in place of the betweenness heuristic, so we can tell whether the heuristic-guided partition performs any better than a random partition. For each partition we ran the same set of experiments as above, using the both complete and prioritised abstract search.

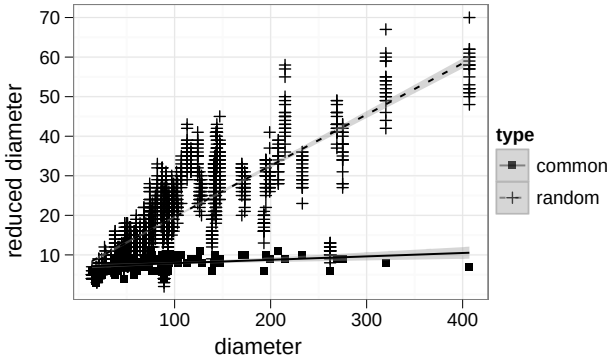
Results

Looking first at the properties of the partitions produced, we can see in Figure 7.3(a) that the average subgraph size for each map is significantly larger in the heuristic-driven partitions than the random. For both approaches, the best results were achieved in low-dimensional graphs. In high-dimensional graphs the algorithm tends to create halls that loop back on themselves and thus end prematurely.

The average degree of the reduced graph (treating each subgraph as a single vertex) is similar between the heuristic and random partitions, ranging between 3 and 4.5 in proportion to the degree of the original graph. This is significantly



(a) subgraph size



(b) diameter

Figure 7.3: A comparison of random partitioning vs using the betweenness heuristic, in terms of (a) subgraph size and (b) diameter of the reduced graph.

higher than the unreduced degree because a long hall can have many more outgoing edges.

The biggest difference between the heuristic and random partitions lies in the diameter of the reduced graph. The diameter of the heuristically partitioned graph is significantly lower than the of the randomly partitioned graph, as shown in Figure 7.3(b). In both cases it showed growth proportional to the diameter of the original graph, but the betweenness-based graphs had diameter about one sixth that of the randomly partitioned graphs. The diameter of the graph gives us a rough expectation of plan length.

A smaller diameter promises shorter abstract plans and therefore faster planning, as is reflected in the success rates, shown in Figures 7.4(a) and 7.4(b). As the plotted curves of best fit indicate, the betweenness heuristic works significantly better than average in the majority of cases. We can be confident, therefore, that the betweenness measure is an appropriate choice of partitioning heuristic.

8 Related Work

The use of spatial abstraction of the roadmap to perform hierarchical path planning is common in the single-agent setting (Botea, Müller, & Schaeffer, 2004; Sturtevant & Buro, 2005) where the automatic construction of abstract states is simply a matter of selecting connected subgraphs (Sturtevant, 2007), however without more careful consideration of the structural properties of these subgraphs, there is no guarantee that these abstractions will work in the multi-robot setting. The work that bears most similarity to my own is not explicitly in robot path planning, but in solving the Sokoban puzzle (Botea, Muller, & Schaeffer, 2002; Junghanns & Schaeffer, 2001), where the collision-free movement of many boxes in a grid world is the goal. 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, tailored to the specific structure of the Sokoban game, 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.

In spite of the power offered by the constraint programming approach, there appears to be little work done in propagating state constraints beyond a single time-step or varying the search order or in multi-robot planning. There is some work by Peng and Akella (Peng & Akella, 2005) which uses mixed integer linear programming to for coordinating robots, but this work assumes that the paths have already been determined and limits the search problem to finding velocities for each robot so that they avoid collisions.

Other notable work on this problem goes under the heading of 'pebble motion on a graph' (Kornhauser, Miller, & Spirakis, 1984). If the roadmap is a tree of n vertices Auletta, Monti, Parente, and Persiano (1999) provide an $O(n + l)$ algorithm for finding a plan of length l, if one exists. Surynek (2009) provides an $O(n^3)$ algorithm for solving bi-connected graphs. In future work we plan to investigate how these methods might be incorporated into our planner.

9 Conclusion

We have demonstrated how modern search techniques can be applied to the problem of multi-robot planning. In particular, we have shown how the Fail-First Principle leads us to reformulate the planning problem to consider critical decisions early and to recognise failures sooner.

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.

Partitioning maps by hand can be slow and tedious, so we offer an heuristic algorithm for the automatic construction of hall subgraphs which detects bottlenecks in the road-map based on the betweenness property. Experimental results have shown that this approach is most valuable in low-dimensional domains, that is maps with many long and narrow corridors, in which case it can improve the success rate markedly. In open spaces with few bottlenecks it offers no less benefit over concrete planning and the overhead of a two-stage planning algorithm outweighs any improvements it might yield.

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.

References

Auletta, V., Monti, A., Parente, M., & Persiano, P. (1999). A linear-time algorithm for the feasibility of pebble motion on trees. *Algorithmica*, *23*(3), 223–245.

Barraquand, J., & Latombe, J.-C. (1991). Robot motion planning: A distributed representation approach. *International Journal of Robotics Research*, *10*(6), 628–649.

Bethesda Softworks (2008). Fallout 3. Computer Game.

Botea, A., Muller, M., & Schaeffer, J. (2002). Using abstraction for planning in sokoban. In Schaeffer, J., Müller, M., & Björnsson, Y. (Eds.), *Proceedings of the 3rd International Conference on Computers and Games CG-02*, Vol. 2883 of *Lecture Notes in Computer Science*, pp. 360–375, Edmonton, Canada. Springer.

Botea, A., Müller, M., & Schaeffer, J. (2004). Near optimal hierarchical path finding. *Journal of Game Development*, *1*(1), 7–28.

Brandes, U. (2001). A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, *25*(2), 163–177.

Freeman, L. (1977). A set of measures of centrality based on betweenness. *Sociometry*, *40*(1), 35–41.

Haralick, R., & Elliott, G. (1980). Increasing tree search efficiency for constraint satisfaction problems. *Artificial intelligence*, *14*, 263–313.

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

Kornhauser, D., Miller, G., & Spirakis, P. (1984). Coordinating pebble motion on graphs, the diameter of permutation groups and applications. In *Proceedings of the 25th Annual Symposium on Foundations of Computer Science*, pp. 241–250, Washington DC, USA. IEEE Computer Society.

LaValle, S. M., & Hutchinson, S. A. (1998). Optimal Motion Planning for Multiple Robots Having Independent Goals. *IEEE Transactions on Robotics and Automation*, *14*(6), 912–925.

Peng, J., & Akella, S. (2005). Coordinating multiple robots with kinodynamic constraints along specified paths. *International Journal of Robotics Research*, *24*(4), 295–310.

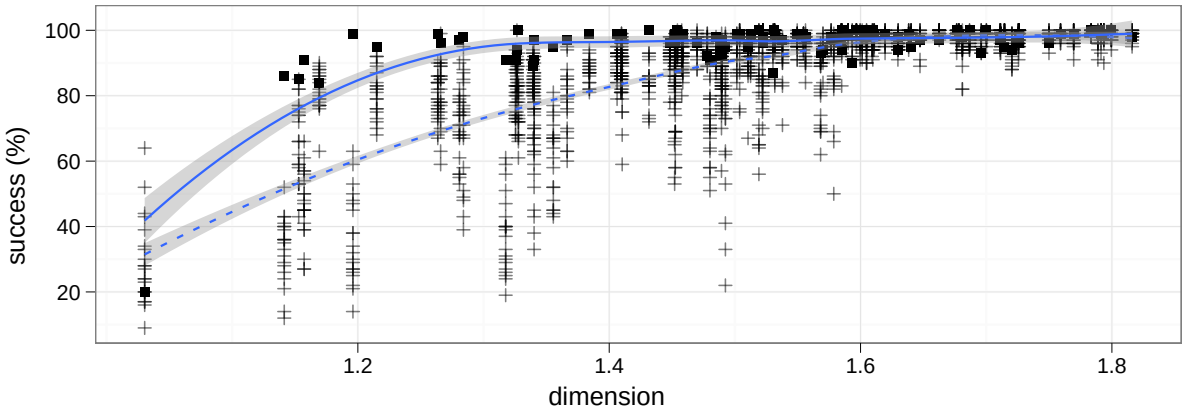
Ryan, M. (2008). Exploiting subgraph structure in multi-robot path planning. *Journal of Artificial Intelligence Research*, *31*, 497–542.

Silver, D. (2005). Cooperative pathfinding. In *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment conference (AIIDE 2005)*. AAAI Press.

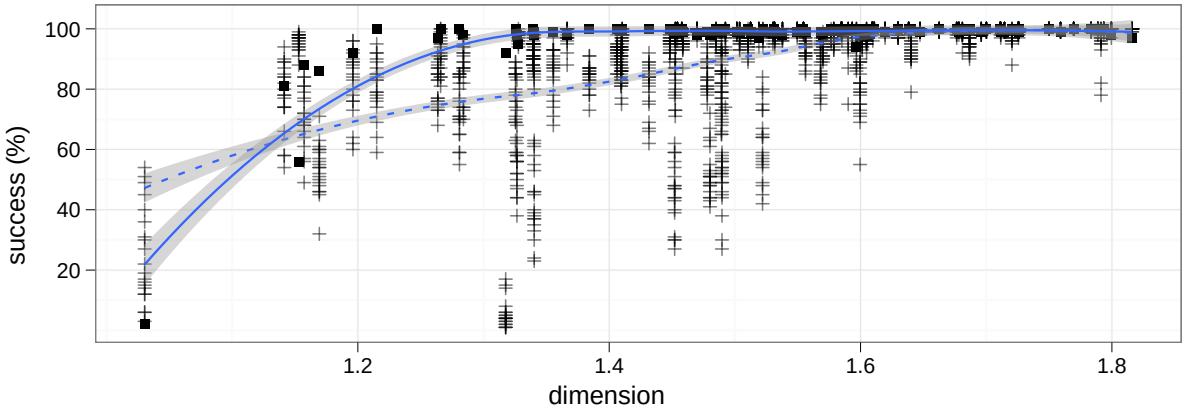
Sturtevant, N. (2007). Memory-efficient abstractions for pathfinding. In *Proceedings of the Third Artificial Intelligence and Interactive Digital Entertainment conference (AIIDE 2007)*. AAAI Press.

Sturtevant, N., & Buro, M. (2005). Partial pathfinding using map abstraction and refinement. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 1392–1397, Pittsburgh, Pennsylvania.

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



(a) complete



(b) prioritised

Figure 7.4: The success rate of the abstract planner using the automatically generated partition versus using a random partition.