Multi-Robot Path-Planning with Subgraphs

Malcolm Ryan

Centre for Autonomous Systems
University of New South Wales
Australia, 2052
malcolmr@cse.unsw.edu.au

Abstract

In this paper we consider the problem of planning paths for an homogeneous group of robots around a shared roadmap. We show that significant speed-up can be achieved if we decompose the roadmap into subgraphs of known structure, such as stacks and cliques, and build plans hierarchically, first between connected subgraphs, then inside the subgraphs. We show that this approach while suboptimal is nevertheless complete and can provide a significant improvement in planning time over a non-hierarchical planner.

1 Introduction

Coordinating multiple robots as they traverse a shared environment (be it an office [Hada and Takase, 2001], a warehouse [Everett *et al.*, 1994], a shipping yard [Alami *et al.*, 1998] or a mine [Alarie and Gamache, 2002]) is known to be a difficult problem [LaValle, 2006]. Centralised methods [Barraquand and Latombe, 1991], which treat the robots as a single composite entity, scale poorly as the number of robots increases. Decoupled methods [LaValle and Hutchinson, 1998], [Erdmann and Lozano-Pérez, 1986], which first plan for each robot independently then resolve conflicts afterwards, prove to be much faster but are incomplete because many problems require robots to deliberately detour from their optimal path in order to let another robot pass. Even if a priority ordering is used [van den Berg and Overmars, 2005], requiring low priority robots to plan to avoid high-priority robots, still problems can be found which cannot be solved with any static ordering.

It seems that centralised planning is unavoidable if we want a complete planning algorithm. This does mean, however, that a naive search of the composite configuration space is the only solution that remains. By analysing the structure of the space, we can find particular topological features which lend themselves to fast special-purpose planners. If we can decompose a map into a collection of such simple submaps, then we can build plans hierarchically, first planning the movements from one submap to another, then using special-purpose planners to build paths within each submap.

In this paper we propose such a planner. We limit ourselves to considering an homogeneous group of robots navigating using a shared roadmap. We identify particular kinds of subgraphs in this roadmap which place known constraints on the ordering of robots which pass through them. We use these constraints to make efficient planning algorithms for traversing each kind of subgraph, and we combine these local planners into a hierarchical planner for solving arbitrary problems. We show that this algorithm is complete (under our simplifying assumptions).

2 Problem Formulation

Since we are interested in topological, and not geometric, features of the world, we assume that our map is given to us in the form of a graph, representing the connectivity of free space for a single robot[1]. Because all our robots are alike, they all share this single roadmap.

We make some further simplifying assumptions about the map:

- The map is constructed such that two robots will only collide if they try to simultaneously occupy the same vertex in the graph. That is, the vertices must be spaced sufficiently far apart that two robots can occupy any pair of distinct vertices without colliding.

- A robot at vertex v_i can move to neighbouring vertex v_j provided v_j is unoccupied, and no other robot is simultaneously entering or leaving v_j. Robots occupying other vertices in the graph do not affect this movement.

- All paths are reversible, so if a path exists from v_i to v_j then a reverse path from v_j to v_i also exists.

- The initial and goal locations of all robots lie on the roadmap.

Obviously these assumptions do not always apply in an arbitrary roadmap, be with appropriate levels of underlying con-

---

[1] There exist many algorithms for producing such roadmaps. Which of these is employed is not important for this work.

trol, they should not be too difficult to achieve or approximate.

Given these assumptions, we can formalise our problem as follows. We represent the roadmap as an undirected graph $G = (V, E)$ containing vertices $V = \{v_1, \ldots, v_n\}$. An edge $e = (v_i, v_j) \in E$ is a simple path in the roadmap. We also have a set of robots $R = \{r_1, \ldots, r_k\}$, and two mappings $S_0, S^+ : R \rightarrow V$, such that $S[r_i] \neq S[r_j]$ iff $i \neq j$, representing the initial and goal positions of the robots respectively.

A plan for a single robot r is a path $P_r = (V_r, E_r)$ in G from $S_0[r]$ to $S^+[r]$. This path may contain loops. Our aim is to compute a multi-robot plan $\mathcal{P} = (\{P_r \mid r \in R\}, \prec)$, consisting of a single-robot plan P_r for each robot r, along with a partial ordering $\prec$ which constrains the possible orderings of robots' actions. This order has to satisfy two conditions:

1. All actions for a single robot r must be totally ordered:

$$E_r(i) \prec E_r(j), \text{iff } i < j$$

2. If two robots r and s visit the same vertex, then one must exit before another can enter:

$$V_r(i) = V_s(j) \Rightarrow E_r(i+1) \prec E_s(j) \lor E_s(j+1) \prec E_r(i)$$

Actions which are not ordered by these rules can be performed simultaneously.

A simple centralised approach to computing $\mathcal{P}$ is given in Algorithm 1. For simplicity, it has been expressed as a non-deterministic algorithm with choice points and failure. A complete implementation of this algorithm would require an appropriate search mechanism to evaluate the different paths of the algorithm and find one which terminates successfully. For this paper we will assume that breadth-first search is used, with appropriate pruning of visited states, but other search techniques are also valid.

This algorithm does a complete search of the composite space $G^k = G \times G \times \ldots \times G$, where k is the number of robots. After eliminating vertices which represent collisions between robots, the size of the composite graph is given by:

$$\left|V(G^k)\right| = \frac{n!}{(n-k)!}$$

$$\left|E(G^k)\right| = |E(G)| \frac{k(n-2)!}{(n-k-1)!}$$

where $n = |V(G)|$ and $k = |R|$. The running time of this algorithm will depend on the implementation of the search used, but it can be expected to be very slow for moderately large values of n and k.

3 Subgraph Decomposition

It is standard practice in automated planning, when faced with an intractable search problem such as this, to look for structure in the domain which can be exploited to simplify the

---

Algorithm 1 A simple centralised planning algorithm

function PLAN(G, S_0, S^+)
  $P_r \leftarrow \emptyset, \forall r \in R$
  $\prec \leftarrow \emptyset$
  return PLAN(G, $\mathcal{P}$, S_0, S^+)
end function

function PLAN(G, $\mathcal{P}$, S_{curr}, S_{goal})
  if $S_{curr} = S_{goal}$ **then**
    return $\mathcal{P}$
  end if
  choose $r \in R$
  $v_f \leftarrow S_{curr}[r]$
  choose $v_t \in \{v \mid (v_f, v) \in G\}$
  $\mathcal{P} = $ MOVE($\mathcal{P}$, S_{curr}, r, v_f, v_t)
  return PLAN(G, $\mathcal{P}$, S_{curr}, S_{goal})
end function

function MOVE($\mathcal{P}$, S, r, v_f, v_t)
  if $\exists r' : S[r'] = v_t$ **then**
    fail
  end if
  add (v_f, v_t) to P_r
  $i \leftarrow \|P_r\|$
  set $E_r(i - 1) \prec E_r(i)$
  if $\exists r', j : V_{r'}(j) = v_t$ **then**
    set $E_{r'}(j + 1) \prec E_r(i)$
  end if
  $S[r] \leftarrow v_t$
  return $\mathcal{P}$
end function

---

search [Long and Fox, 2002]. In this case, useful structure can be found by identifying certain kinds of subgraphs in the base graph G. Useful subgraphs are those that place easily computable limitations on the order of robots passing though. Knowing these limitations we can build special-purpose planners for each subgraph type, which avoid the complexity of complete search.

3.1 Subgraph types

Two examples of such subgraph types are *stacks* and *cliques*, which we characterise below. These two were chosen for their simplicity, to illustrate the algorithm. While they do occur in realistic graphs, we emphasise that they would not be sufficient on their own. Other subgraph types are under development and will be reported in future publications.

Stacks

A *stack* (fig. 1a) represents a long narrow dead-end road or corridor in the map. It is too narrow for robots to pass each other so they must enter and leave this subgraph in a last-in-first-out order.

Formally it is an induced subgraph $S \subseteq G$ with vertices $V(S) = \{v_0, v_1, \ldots, v_m\}$ [Diestel, 2005] satisfying two conditions:

1. The nodes form a path without any shortcuts:

$$(v_i, v_j) \in E(S), \text{ iff } |i - j| = 1, \forall v_i, v_j \in V(S)$$
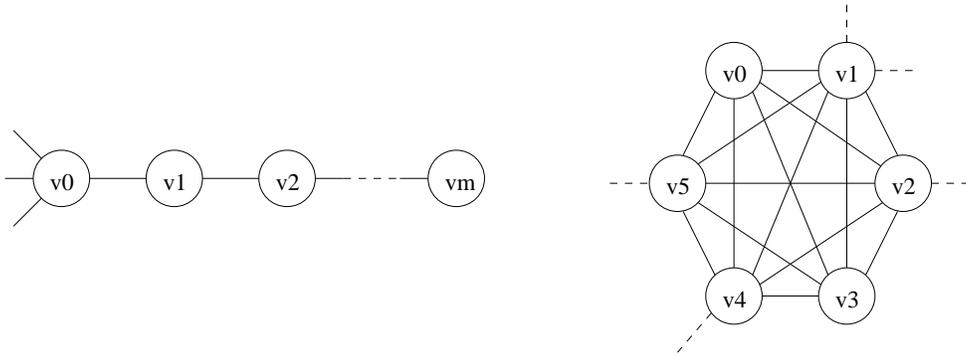
Figure 1: Subgraphs: (a) a stack, (b) a clique

2. Only the "head" node v_0 may connect with other subgraphs:

$$x \in V(S) \wedge (x, y) \in E(G) \Rightarrow y \in V(S) \vee x = v_0$$

These two properties place corresponding restrictions on the order of robots entering and leaving the stack. They are:

1. The order of robots inside the subgraph cannot change without a robot entering or leaving the stack.

2. Only the last robot to enter the stack can leave it.

Cliques

A *clique* (fig. 1b) represents a large open area in the map with many exit points (vertices) around its perimeter. Robots can cross directly from any vertex to another, and as long as the clique is not full, other robots inside can be shuffled out of the way to allow this to happen.

Formally it is an induced subgraph $S \subseteq G$ which is complete, i.e. every vertex in the subgraph is connected to every other. Cliques have the useful property that robots can enter and leave in any order. A robot in the clique can always exit by any connecting edge, without requiring other robots to leave the subgraph, provided the clique is not full (i.e. the number of robots in the clique is less than the size of the subgraph).

3.2 Partitioning the graph

We can now consider partitioning the base graph G into subgraphs of different types. Since obviously not every graph can be decomposed solely into stacks and cliques, we shall also define the *singleton* subgraph which consists of a single node. Now we can construct a partition $\mathcal{G} = \{G_0, G_1, \ldots, G_m\}$ where each G_i is a subgraph of one of these three types. Given this partition we can construct a minor X of G by contracting each subgraph to a single vertex. That is:

$$V(X) = \mathcal{G}$$
$$E(X) = \{(G_i, G_j) \mid \exists x \in G_i, y \in G_j : (x, y) \in G\}$$

We shall call X the *reduced graph*. This process is illustrated in Figure 2. For the purposes of this paper we assume that this partitioning has been done prior to planning. Whether partitioning is done by hand or by an automated process is not important at this stage.

3.3 Planning with subgraphs

Planning with the reduced graph is now a two step process. First we plan a path between the connected subgraphs, from the subgraph containing the starting state to the subgraph containing the goal. Algorithm 2 shows the process, which proceeds similarly to Algorithm 1 above. The difference is that checks need to be made when entering and exiting each subgraph to make sure that this action is permitted. The functions ENTER and EXIT are defined for each subgraph type, to check that it is legal for robots to enter and leave the subgraph in the order expressed in the current plan $\mathcal{P}$, and to add any necessary constraints to the partial order $\prec$. Also, before deciding that a plan is complete, we need to call the TERMINATE function on each subgraph to confirm that the robots inside that subgraph can be rearranged into their goal positions.

Each of these functions must only succeed if the given robot can enter/exit/terminate in the subgraph *without* needing any other robot currently in the subgraph to leave. In this way, the responsibility for planning movement *between* the subgraph is handled by the main planner, and movement *within* each subgraph is left up to the special-purpose planners for each subgraph type (which, it is assumed, can do so much more efficiently).

Task-specific planners for each subgraph type to expand all visits to a subgraph into appropriate sequences of internal edges. The resolution process also adds additional constraints to $\prec$ to order the robots' movements within each subgraph.

Two subgraph types, stacks and cliques, are described below. They were selected as examples for which the ENTER, EXIT, TERMINATE and RESOLVE functions can be computed simply, without requiring a complete search of all possible paths through the subgraph[2].

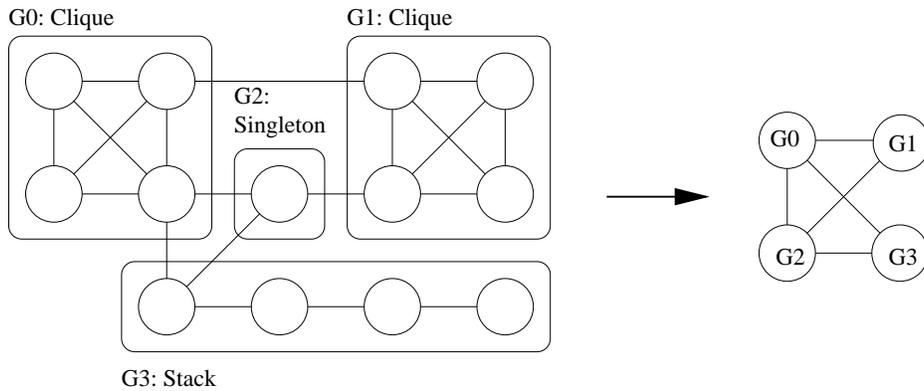[2]The full pseudocode for these functions has been omitted for

Figure 2: Partitioning a graph into subgraphs: (a) The partitioned graph, (b) The reduced graph

Stacks

Planning for stacks is simple. A robot can always enter, providing the stack is not full. A robot can only exit if it was the last to enter. Termination is possible if the order of robots in the stack matches the order of termination positions, so that the robots can move to their goals without having to pass each other.

To resolve a sequence of visits to a stack, we simply need to keep track of the depths of each robot inside it. When a new robot enters, robots in the stack are pushed deeper. When a robot exits, it moves directly to the head of the stack and then out.

Cliques

As with stacks, a robot can always enter a clique (by any incoming edge) so long as the clique is not full. However, exiting a clique is somewhat more complex. We must consider not only when the robot wants to exit, but also by which edge it departs.

When the clique is less than full, any robot in the subgraph can exit via any outgoing edge. If there is a robot in the way, it can be moved to an empty vertex in the clique. If, however, the clique is full then no rearrangement is possible. We say that the clique is *locked*. The robot that was last to enter a locked clique may only exit by edges that connect to the vertex it entered on (the *locking vertex*). Other robots in the subgraph can arrange themselves into position to leave from any vertex *before* the clique becomes full, but they cannot occupy the locking vertex (as it must be vacant for the final robot to enter) and thus they cannot exit from this vertex. Once one robot has exited, the clique is no longer full and robots can once more enter and exit without restriction.

Arranging robots for termination follows these same rules. If the clique is not full, termination is always possible. If the

clique is locked, then the last robot to enter can only terminate in the locking vertex, and other robots in the clique must terminate in non-locking vertices. If this is not the case, then termination will fail.

We can implement a RESOLVE function for cliques by iterating through the history of entrances to and exits from the clique. For each entrance, we first check whether the vertex to be entered is occupied. If it is, we move the occupant to another empty spot in the clique (the existence of which is guaranteed by the earlier requirement that no robot can enter a full clique). For each exit we move the robot directly to the exiting vertex, moving any previous occupant out of the way into another empty vertex. We then move the exiting robot out of the clique.

A complication arises if a robot locks the clique. We need to rearrange the other robots in the clique *before* the locking robot enters, so that the next robot to exit subsequently will be in its exiting vertex. Alternatively, if all the robots terminate in the clique, then we need to ensure that they are all in their proper terminating positions before the clique is locked.

This algorithm is sound and complete. It will always generate a plan when one exists, but it is not optimal in terms of the length of the plans. Depending on the selection of empty vertices for moving occupants out of the way, a robot in the clique might be shunted around unnecessarily many times. This could be improved, but only at the expense of a lot more search to the resolution procedure.

4 Experiments

To demonstrate the improved efficiency of our subgraph-based planner, we shall test it in two simple constructed domains, one using stacks and one using a cliques.

For testing stacks, we used a series of maps of different sizes, constructed from three connected Stacks X, Y and Z, as shown in Figure 3(a). All three stacks are of equal size n. We start with k robots $\{r_1, \ldots, r_k\}$ arranged in order in stack X with r_1 at x_n, r_2 at x_{n-1} and so forth. The goal is to
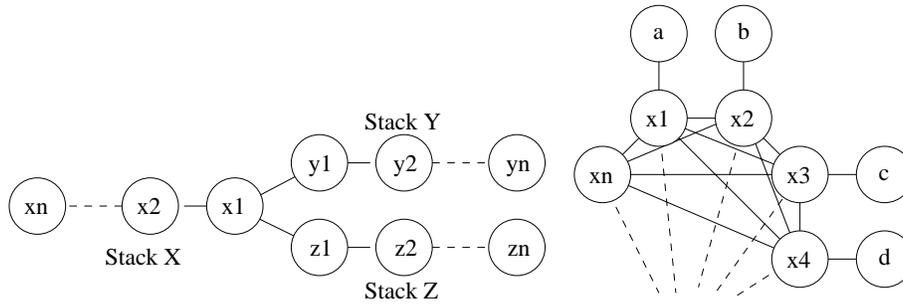
Figure 3: The graphs used for experiments using (a) three stacks, and (b) a clique

Algorithm 2 Planning on the reduced graph

---

function PLAN($\mathcal{G}$, X, R, S_0, S^+)
  $P_r \leftarrow \emptyset, \forall r \in R$
  $\prec \leftarrow \emptyset$
  for $r \in R$ **do**
    $G_0[r] \leftarrow G \in \mathcal{G}$, s.t. $S_0[r] \in G$
  end for
  $\mathcal{P} \leftarrow$ PLAN($\mathcal{G}$, X, R, $\mathcal{P}$, G_0, S^+)
  for $G \in \mathcal{G}$ **do**
    $\mathcal{P} \leftarrow G$.RESOLVE($\mathcal{P}$)
  end for
  return $\mathcal{P}$
end function

function PLAN($\mathcal{G}$, X, R, $\mathcal{P}$, G, S^+)
  if $\forall S^+[r] \in G[r]$ **then**
    for $r \in R$ **do**
      $\mathcal{P} \leftarrow G[r]$.TERMINATE($\mathcal{P}$, r, $S^+[r]$)
    end for
    return $\mathcal{P}$
  end if
  choose $r \in R$
  $G_f \leftarrow G[r]$
  choose $G_t \in \{G' \mid (G_f, G') \in X\}$
  choose $(v_f, v_t) \in \{(x, y) \in \mathcal{G} | x \in G_f, y \in G_t\}$
  $\mathcal{P} \leftarrow G_f$.EXIT($\mathcal{P}$, r, v_f)
  $\mathcal{P} \leftarrow G_t$.ENTER($\mathcal{P}$, r, v_t)
  add (v_f, v_t) to P_r
  $G[r] \leftarrow \mathcal{G}_t$
  return PLAN($\mathcal{G}$, X, R, $\mathcal{P}$, G, S^+)
end function

---

reverse the order of the robots.

The map used for testing cliques is shown in Figure 3(b). It consists of a single clique of n vertices connected to several outlying singleton nodes. Robots start in the singleton nodes. Each robot has to move to the next singleton clockwise around the circle, by passing through the clique. In some experiments we also added robots inside the clique in order to make the passage more complicated.

We report on three experiments we conducted using these two maps. These experiments are design to demonstrate how our subgraph planner scales with regard to three variables: 1) The number of states in the graph (keeping the number of subgraphs constant), 2) the number of subgraphs (keeping the

number of states constant), and 3) the number of robots.

While we do not claim that these example maps are in any way realistic, they do illustrate the power of this technique. More complex subgraph types are in development for application in real problems (see Section 5 below).

4.1 Experiment 1: Scaling the subgraph size

In the first experiment we varied the size of the subgraphs while keeping the structure of each graph and the number of robots constant. We would expect the subgraph planner to be fairly impervious to changes in the size of the subgraphs as it does not effect the search space particularly. The naive planner, on the other hand, was expected to show an increase in running time roughly exponential with the overall size of the map.

For the stack-based map, we started with three robots at the bottom of stack X and the goal was to reverse their order. The size of the three stacks was varied from 3 to 10. For the clique-based map we started with four robots in four singleton nodes connected to a clique of size n, where n varied from 4 to 10.

The average running times of the two planners (the naive planner in Algorithm 1 and the subgraph planner in Algorithm 2) were computed over 100 runs[3] The results are shown in Figures 4(a) and (b).

These results confirm our expectations: while the running time of the naive planner grows quickly with the number of vertices, the subgraph planner shows a constant running time. In all cases we see a significant improvement in planning time from using the subgraph information.

The only other feature of note is the significantly longer running time for the subgraph planner in the clique map with clique size four. This appears to be because in this case the planner has to handle the possibility of the clique becoming locked (when all four robots enter) which created many more dead-end branches in the search space. When the clique is larger, this can never occur.

---

[3]All experiments in this paper were run using Sun JDK 1.5.0 with 1Gb of heap space on an Intel(R) Xeon(TM) CPU 3.20GHz processor.
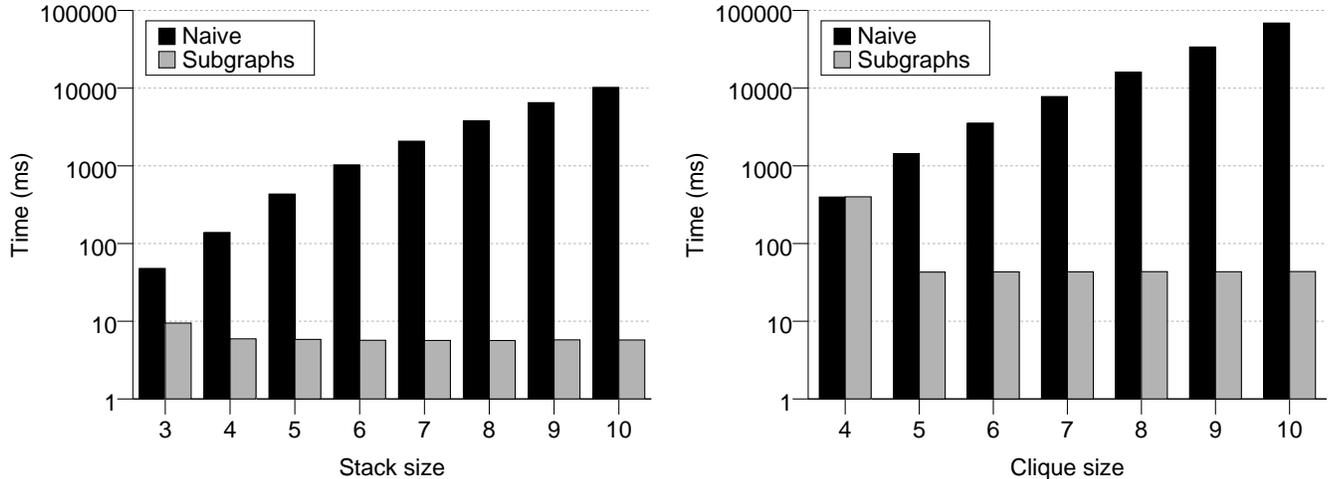
Figure 4: Varying the subgraph size in (a) the stack-based and (b) the clique-based graphs.

4.2 Experiment 2: Scaling the number of subgraphs

In the second experiment we kept the overall number of edges and vertices in the maps constant, while we varied the number of subgraphs. In this case we would expect the opposite result to that shown above. That is, the naive planner should show little variation in performance while the subgraph based planner should take more time to execute as the number of subgraphs increases (increasing the size of the search space).

For the stack-based map we set the total size of the map to 21 vertices which were then divided into m stacks of approximately equal size. We varied m from 3 to 7. The stacks were connected with an edge from the head of the starting stack X to the heads of every other stack. Three robots began in stack X and the goal was to reverse their order.

For the clique-based map we kept the structure of the map constant and simply changed the subgraph decomposition. A single clique of 24 vertices was connected to 3 singleton nodes. The clique was partitioned into m equal-sized cliques, for $m = 1, 2, \ldots, 6$. Three robots started in the three singleton nodes, with the goal of each robot moving to the singleton on its right.

The results, shown in Figures 5(a) and (b), confirm out expectations. In the subgraph-based map, the naive planner showed slightly improved performance as the number of subgraphs was increased (presumably because the total search depth decreased), while the running time of subgraph planner grew quickly. It is still notable that even with a large number of stacks, the subgraph planner outperforms the naive planner significantly.

The naive planner was unable to solve the clique-based problem (running out of stack space) but we can see that the subgraph planner gives us the results we expected, taking pro-

gressively longer as the search space increases. The leap from one clique to two is particularly pronounced, and may be because our current algorithm requires us to consider every edge which connects the two cliques even when this does not affect rest of the search.

4.3 Experiment 3: Scaling the number of robots

In the third experiment we kept the map constant and varied the number of robots. Both algorithms should be affected by the and we would expect to see rapid increases in the running time of each as the search space increases roughly exponentially.

For the stack-based map, we used three stacks of length 10. The number of robots was varied from 1 to 6. For the clique-based map we set the clique size to 10, with four singletons around the outside. Four robots were started in these four vertices and $k = 0, \ldots, 4$ extra robots were started within the clique at positions $x_1, \ldots, x_k$. The goal of the outer robots was to move to the next singleton in a clockwise direction. The inner robots were to finish in the same position as they started.

The results are shown in Figures 6(a) and (b). In both cases the naive planner ran out of memory and could not complete the larger problems. Both approaches show a very rapid increase in running time as the number of robots increases, but the subgraph planner can clearly handle significantly more robots than the naive planner, due to the decrease in the size of the reduced graph.

4.4 Discussion

These experiments clearly show the advantages of the subgraph decomposition method. By partitioning the base graph into subgraphs and then planning on the reduced graph, we significantly decrease one of the two variables that affects the
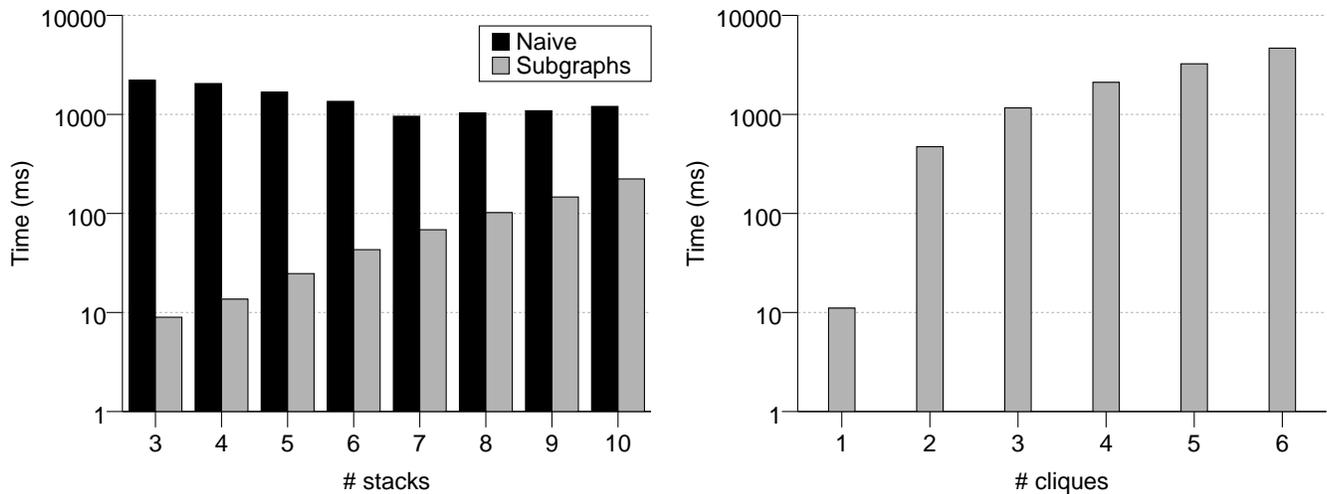
Figure 5: Varying the number of (a) stacks, and (b) cliques. No results are shown for the naive planner in (b) as it was unable to solve any of the problems within the available memory limit.

running time of the algorithm, that is, the map size. If we can reduce the map into only a handful of connected subgraphs then running time improves significantly. Our approach still does not scale well with an increasing number of robots, but the gains from pruning the map do allow us to compute plans with several more robots than might otherwise be possible.

5 Conclusion and Further work

We have outlined a method for reducing the search-space in multi-robot path planning problems by decomposing the map into connected subgraphs of known structure. Planning becomes a search for a path from one subgraph to another, using special-purpose code to determine when transitions between subgraphs are possible, and for planning paths within each subgraph. We have demonstrated how this process can significantly reduce the search time for planning, allowing us to build plans with more robots.

The subgraph types we examined in this paper, stacks and cliques, are still fairly simple and are unlikely to occur often in real planning problems. To extend this work to more realistic maps we are investigating two more complex subgraph types: halls and rings. Both are essentially long chains of vertices with many entrances and exits along their length, where rings contain a link from the final vertex to the first, while halls do not. The special-purpose planners for these two subgraphs types are more complex than those presented here, but are still not too difficult to produce.

It should be possible to algorithmically detect the presence of all of these different subgraph structures in a map, and so we also hope to produce a preprocessor which can decompose a map automatically. Finding an optimal decomposition may be intractable, but even a greedy search could be expected to produce good results.

Finally, we want to tackle the remaining problem of scaling towards really large numbers of robots. We imagine that this might be done by taking advantage of symmetries and almost-symmetries in the composite search space, as per [Porteous *et al.*, 2004], but much work remains to be done before this can be attempted.

References

[Alami *et al.*, 1998] R. Alami, S. Fleury, M. Herrb, F. Ingrand, and F. Robert. Multi-robot cooperation in the MARTHA project. *Robotics & Automation Magazine, IEEE*, 5(1):36–47, 1998.

[Alarie and Gamache, 2002] S. Alarie and M. Gamache. Overview of Solution Strategies Used in Truck Dispatching Systems for Open Pit Mines. *International Journal of Surface Mining, Reclamation and Environment*, 16(1):59–76, 2002.

[Barraquand and Latombe, 1991] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *International journal of robotics research*, 10(6):628–649, 1991.

[Diestel, 2005] Reinhard Diestel. *Graph Theory*. Springer-Verlag, 2005.

[Erdmann and Lozano-Pérez, 1986] M. Erdmann and T. Lozano-Pérez. On Multiple Moving Objects. Technical Report 883, M.I.T. Artificial Intelligence Laboratory, 1986.

[Everett *et al.*, 1994] H.R. Everett, D.W. Gage, GA Gilbreth, R.T. Laird, and R.P. Smurlo. Real-world issues in ware-
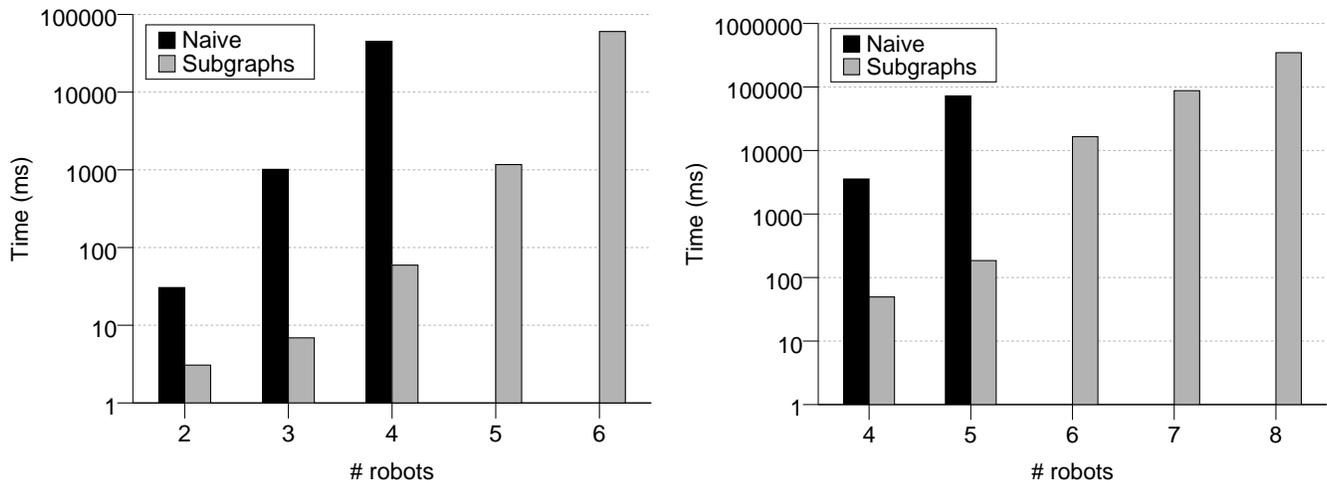
Figure 6: Varying the number of robots in (a) the stack-based, and (b) the clique-based map. Missing bars indicate that the naive planner was unable to solve the larger problems within the memory limit.

house navigation. *Proceedings of the SPIE Conference on Mobile Robots IX*, 2352, 1994.

[Hada and Takase, 2001] Y. Hada and K. Takase. Multiple mobile robot navigation using the indoor global positioning system (iGPS). *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, 2, 2001.

[LaValle and Hutchinson, 1998] Steven M. LaValle and Seth A. Hutchinson. Optimal Motion Planning for Multiple Robots Having Independent Goals. In *IEEE Transactions on Robotics and Automation*, volume 14, 1998.

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

[Long and Fox, 2002] D. Long and M. Fox. *Planning with Generic Types*, chapter 4, pages 103–138. Morgan Kaufmann, 2002.

[Porteous *et al.*, 2004] J. Porteous, D. Long, and M. Fox. The Identification and Exploitation of Almost Symmetry in Planning Problems. In K. Brown, editor, *Proceedings of the 23rd UK Planning and Scheduling SIG*, 2004.

[van den Berg and Overmars, 2005] J. van den Berg and M. Overmars. Prioritized Motion Planning for Multiple Robots. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 430–435, 2005.