Graph Decomposition for Efficient Multi-robot Path Planning

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

Abstract

In my previous paper (Ryan, 2006) I introduced the concept of subgraph decomposition as a means of reducing the search space in multi-robot planning problems. I showed how partitioning a roadmap into subgraphs of known structure allows us to first plan at a level of abstraction and then resolve these plans into concrete paths without the need for further search so we can solve significantly harder planning tasks with the same resources. However the subgraph types I introduced in that paper, stacks and cliques, are not likely to occur often in realistic planning problems and so are of limited usefulness.

In this paper I describe a new kind of subgraph called a *hall*, which can also be used for planning and which occurs much more commonly in real problems. I explain its formal properties as a planning component and demonstrate its use on a map of the Patrick's container yard at the Port of Brisbane in Queensland Australia.

1 Introduction

Coordinated path planning for multiple robots is difficult because the search space grows combinatorially with the number of robots (LaValle, 2006). We need to plan not only for the paths of individual robots but also for the ordering of their movements to prevent them from colliding with one another. Often it is necessary for robots to detour away from their shortest paths in order to let other robots pass. As a result centralised planning methods (eg Barraquand & Latombe, 1991), which plan for all robots at once, have traditionally been abandoned in favour of prioritised planners which plan for only one robot at a time in priority order (Erdmann & Lozano-Pérez, 1986; LaValle & Hutchinson, 1998; Berg & Overmars, 2005). Collisions are avoided by requiring lower priority robots to plan to avoid those of higher priority. This is much faster since the search space is kept relatively small but can be incomplete. It is not difficult to construct a problem for which there is no prioritised solution.

In my previous paper (Ryan, 2006) I proposed a way to perform centralised planning in a much more efficient manner by dealing with the problem at a higher level of abstraction. If we partition the map into a set of connected subgraphs then we can first of all plan for the movement of the robots between the subgraphs and then resolve these abstract plans into sequences of more concrete steps. If the subgraphs we use have well-chosen structure then this abstraction will significantly reduce the size of the search space, and the resolution process can be done deterministically without the need for further search.

The subgraphs that I have previously described, stacks and cliques, work well for illustrating this idea but are seldom found in real planning problems. In this paper I introduce a more complex kind of subgraph called a *hall* which is much more common in real planning problems. I explain its structure and its formal properties below and I shall demonstrate its value by using it to do multi-robot planning on a real problem: moving cargo-handler robots around a map of the Patrick container yard at Port Brisbane.

2 Problem Formulation

We assume for this work that we are provided with a roadmap in the form of a graph $G = (V, E)$ representing the connectivity of free space for a single robot moving around the world. We consider our robots to be homogeneous so a single map suffices for them all.

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

With appropriate levels of underlying control these assumptions should not be too difficult to achieve or approximate for most problems.

We also have a set of robots $R = \{r_1, \ldots, r_k\}$, and two mappings $S_0, S^+ : R \to V$ representing the initial and goal positions of the robots respectively, with $S[r_i] \neq S[r_j]$ iff $i \neq$

Algorithm 1 Planning on the reduced graph

```
function PLAN(G, X, R, S_0, S^+)
  P_r ← ∅, ∀r ∈ R
  ≺ ← ∅
  for r ∈ R do
    G_0[r] ← G ∈ G, s.t. S_0[r] ∈ G
  end for
  P ← PLAN(G, X, R, P, G_0, S^+)
  for G ∈ G do
    P ← G.RESOLVE(P)
  end for
  return P
end function

function PLAN(G, X, R, P, G, S^+)
  if ∀r : S^+[r] ∈ G[r] then
    for r ∈ R do
      P ← G[r].TERMINATE(P, r, S^+[r])
    end for
    return P
  end if
  choose r ∈ R
  G_f ← G[r]
  choose G_t ∈ {G' | (G_f, G') ∈ X}
  choose (v_f, v_t) ∈ {(x, y) ∈ G | x ∈ G_f, y ∈ G_t}
  P ← G_f.EXIT(P, r, v_f)
  P ← G_t.ENTER(P, r, v_t)
  add (v_f, v_t) to P_r
  G[r] ← G_t
  return PLAN(G, X, R, P, G, S^+)
end function
```

j. Given this information we wish to construct a set of plans $\mathcal{P} = \{P_r \mid r \in R\}$ for each robot to reach its goal. We also construct a partial-ordering $\prec$ between plan steps so that two robots can never occupy the same the vertex of the map.

2.1 Subgraph Planning

At this stage a naive centralised planner would proceed as follows: First, initialise every robot at its starting position, then select a robot and move it to a neighbouring vertex, checking first that no other robot is currently occupying that vertex. Continue in this fashion selecting and moving one of the robots at each step until all robots are at their goals. Of course, each choice presents multiple possibilities and all alternatives have to be searched in some systematic fashion, usually by breadth-first or A* search.

Subgraph planning proceeds in a similar fashion, as illustrated in Algorithm 1. Before we begin the map G is partitioned into a collection of subgraphs $\mathcal{G} = \{G_0, G_1, \ldots, G_m\}$ and then a minor X of G is constructed by contracting each subgraph to a single vertex;

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

Given this partitioning the planning algorithm proceeds as before. We initialise the robots in their starting positions and then search through a sequence of steps. Each step consists of selecting and moving one of the robots, but in this case we consider how the robot might be moved from the subgraph it currently occupies to one of its neighbours in X. This is more complex than in the naive planner. When we attempt to move a robot from one subgraph to another we must first check that it is possible to do this without forcing any of the other robots within the subgraph to exit. Similarly, we must check that the robot can validly enter the neighbouring subgraph. Rearranging robots *within* the subgraph is permissible but movement *between* subgraphs must only occur as a deliberate plan step.

Once a robot has reached the target subgraph in which its goal vertex lies it must also check that is is possible for it to terminate there. It may be that other robots within the subgraph prevent it from reaching its goal location without any robots exiting the subgraph.

Thus for each subgraph structure we use, we need to implement three test methods used in the search progress:

- ENTER$(\mathcal{P}, r, v)$ Test if robot r can enter a vertex v in the subgraph.
- EXIT$(\mathcal{P}, r, v)$ Test if robot r can exit the subgraph via vertex v.
- TERMINATE$(\mathcal{P}, r, v)$ Test is robot r in the subgraph can move to its goal vertex v.

The key to efficient subgraph planning is to carefully constrain the allowed structure of the subgraphs in our partition, so that the ENTER, EXIT and TERMINATE methods are simple to implement, and do not require expensive search. The advantage of this approach is that each of these functions can always be computed based only on the arrangement of other robots within that particular subgraph and do not rely on the positions of robots elsewhere.

If these three methods are correctly implemented then once a subgraph-based plan has been found, we can guarantee that there is a concrete resolution of this plan into edge transitions in G which fits this abstraction. Furthermore, if no subgraph plan can be found then no concrete plan exists either, as every concrete plan has an abstract representation[1] and our search algorithm is complete.

When subgraph planning is complete, we need an additional method:

- RESOLVE$(\mathcal{P})$ To resolve an abstract plan of enter/exit/terminate events into a concrete sequence of movements between vertices.

Under our assumptions stated earlier, we can resolve the movements of robots within one subgraph without reference to robots elsewhere. So we can implement a RESOLVE method for each subgraph type which performs this operation. Once again, the key is to choose our subgraph structure so that this method can be implemented efficiently without the need for additional search.

[1] The proof of this statement is rather long and will be published in a forthcoming journal paper. The fundamental idea is that a concrete plan can be divided into sections of subgraph rearrangement interleaved with transition actions whenever a robot moves from one subgraph to another. Each transition becomes an step in the abstract plan.

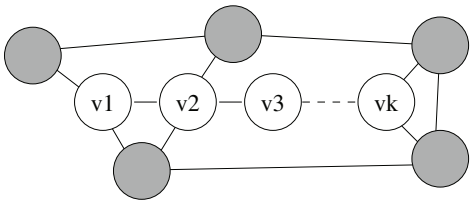

Figure 1: A graph containing a hall subgraph.

3 Subgraph structures

What, then, are these 'subgraph structures' that allow efficient subgraph planning? Previously I have described the use of *stacks* and *cliques* as subgraphs. A stack is a chain of vertices with only a single exit at one end. Robots can only enter and leave a stack in a last-in-first-out order. Cliques are fully-connected subgraphs and may have many exits to other subgraphs from different vertices. Because every vertex in a clique is connected to every other, robots can move in and out of a clique in arbitrary order as long as the clique never becomes full.

The ENTER and EXIT methods for these types of subgraphs are so simple because they place very strong constraints on the structure of the subgraph and its connections to other subgraphs. Unfortunately, for this reason, these kinds of subgraphs rarely occur in realistic planning problems. We therefore need to find other structures that are complex enough to represent real problems, yet still simple enough to implement efficiently. The *hall* is such a structure.

3.1 The Hall Subgraph

A hall (Figure 1) is a generalisation of a stack to include multiple entrance and exit points along its length. Formally it is an induced subgraph $H \subseteq G$ consisting of a chain of vertices $V(H) = \{v_1, \ldots, v_k\}$ each linked to its two neighbours without any 'shortcuts', ie:

$$(v_i, v_j) \in E(H), \text{ iff } |i - j| = 1, \forall v_i, v_j \in V(H)$$

As its name suggests, we may imagine a hall as a long narrow passageway with multiple exits along its length. The passage is too narrow for robots to pass one another, so the sequence of robots within the hall will constrain which exits can be used by a given occupant at a particular time.

To implement the four planning methods ENTER, EXIT and TERMINATE, RESOLVE for halls, we need to keep track of the sequence of robots within the hall at any given moment, including not just their names but also their order (but not, importantly, their actual vertices). I shall denote this sequence as $S = \langle r_{(1)}, r_{(2)}, \ldots, r_{(n)} \rangle$. For each method, I shall now describe how it depends on and affects S.

ENTER()

When a robot r enters the hall, we can compute the possible sequences this creates. If the robot enters at vertex v_i of a hall of length k which already contains $n < k$ robots in sequence S, then the resulting sequence can be of the form:

$$insert(S, r, j) = \langle r_{(1)}, \ldots, r_{(j)}, r, r_{(j+1)}, \ldots, r_{(n)} \rangle$$

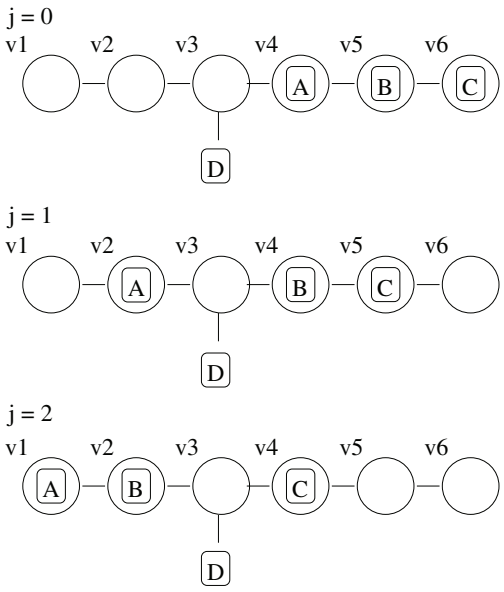

Figure 2: Example of entering a hall subgraph, with $n = 3$, $k = 6$ and $i = 3$. Robot D can enter at three possible sequence positions j = 0, 1 or 2 but not at $j = 3$

for any j satisfying:

$$\begin{array}{rcl} j & \geq & \max(0, n - k + i) \\ j & \leq & \min(n, i - 1) \end{array}$$

If several different values of j are possible then the planner must consider all of these alternatives sequences in the ongoing plan, as they may lead to different outcomes. An example is shown in Figure 2.

EXIT()

Similarly, the position that a robot occupies in the sequence S determines which edges it can use when exiting the hall. If robot $r_{(j)}$ wants to exit from a hall of length k, then it can do so only via edges which connect to a vertex v_i with:

$$\begin{array}{rcl} i & \geq & j \\ i & \leq & k - n + j \end{array}$$

This constraint limits which of the neighbouring subgraphs can be entered by this robot at a particular time.

TERMINATE()

We can terminate the robots in the hall provided that the sequence S matches the sequence termination positions. That is, if $r_{(i)}, r_{(j)} \in S$ with $i < j$ terminate at v_a and v_b respectively then we must have $a < b$.

RESOLVE()

To resolve a subgraph plan which uses a particular hall we need to know three pieces of information:

1. The list of robots which entered and exited the hall,
2. The edges by which they entered and exited,
3. The sequence positions at which they entered.

Given this information we can construct a concrete plan by moving the robots on an as-needs basis. When a robot enters we shuffle the existing occupants left or right as necessary to open up a gap at the appropriate sequence position and we move the newcomer into this gap. When a robot leaves we shuffle the robots appropriately so that it will be at the right vertex from which to depart. None of this requires any kind of search and we are guaranteed that it will succeed because of the checks already done in the ENTER and EXIT methods.

Finally, when we have processed all the entrances and exits the remaining robots should be in the correct sequence to terminate. We can now just move each one to its terminal position, shuffling any that are in the way further up the hall (close to their own terminating positions).

3.2 Complexity

The ENTER and EXIT methods for a hall are both $O(m)$ where m is the number of different sequence positions permitted by the formulae above, which is guaranteed to be less than or equal to the size of the hall $|H|$. TERMINATE is $O(1)$, amortised over all robots in the hall and the RESOLVE method adds a further $O(h)$ term, where h is the number of visits to the hall. Since each subgraph must be resolved in this fashion, this post-processing takes extra time equal to the sum of the lengths of the robots' plans.

4 Experiments

To evaluate the advantage of planning with the hall subgraph structure, I applied it to a large realistic planning task. A map of the Patrick port facility at Port Brisbane in Queensland was provided by the company (Figure 3). This map is used by their computer system to plan the movements of automated straddle carriers - large vehicles which transport shipping containers around their yards. Efficient planning of coordinated paths for these vehicles is important for the smooth running of the facility.

The map is an undirected graph of 1808 vertices and 3029 edges. The vertices are naturally connected in long straight chains representing the roads around the facility. These roads mean that the vertices can be partitioned into 40 hall subgraphs, with only 2 vertices left over, which must be treated as singletons (Figure 4). The reduced graph has 187 edges connecting neighbouring subgraphs.

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

Using this map, I ran the following experiment: N robots were placed in the graph at randomly selected vertices. To force some amount of interaction, each robot was then required to move to the starting location of the next robot in the list, with the final robot moving to the location of the first. Two plans were constructed, one using a naive centralised planner and one using the subgraph planner. Both planners performed a best-first search of the plan space using a distance heuristic based on the all-shortest-paths matrix for a single robot in the map.

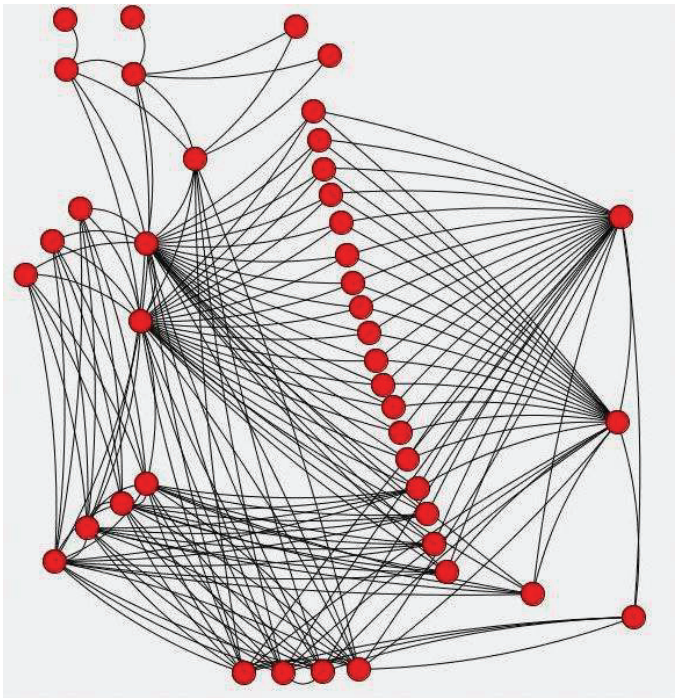

Figure 4: The reduced map. Each vertex represents a single subgraph.

The experiment was repeated 200 times each for values of N from 2 to 10 and the median running time was calculated[2]. These values are plotted in Figure 5. Medians are shown rather than means, as in some cases the planners failed due to lack of memory. Such instances are treated as having an infinite running time. Figure 6 shows the percentage of cases which ran out of memory for each planner, exceeding the 1Gb limit placed on the heap.

As can be seen from these two graphs, the naive planner works more efficiently on small problems (due to the extra overhead of subgraph planning) but quickly blows up as the number of robots is increased, taking much more time and memory. Subgraph planning is able to handle significantly larger problems with many more robots before its search space also becomes too large.

With regard to plan length, the use of best-first search meant neither planner was designed to achieve optimality and outcomes were mixed. In some cases by keeping to the halls the subgraph planner produced much shorter plans. In other cases poorly chosen sequencing decisions meant that these plans were much longer. Work is in progress to construct sensible measures of the length of an abstract plan and heuristics to estimate the cost to completion so that A* search can be applied to the abstract planning problem, to optimise the lengths of plans. Nevertheless we feel that it is better to be producing a sub-optimal plan quickly than to search for an optimal plan and fail.

[2]Running times were measured on a 3.20GHz Intel(R) Xeon(TM) CPU running Sun JDK 5.0 with 1Gb of heap.

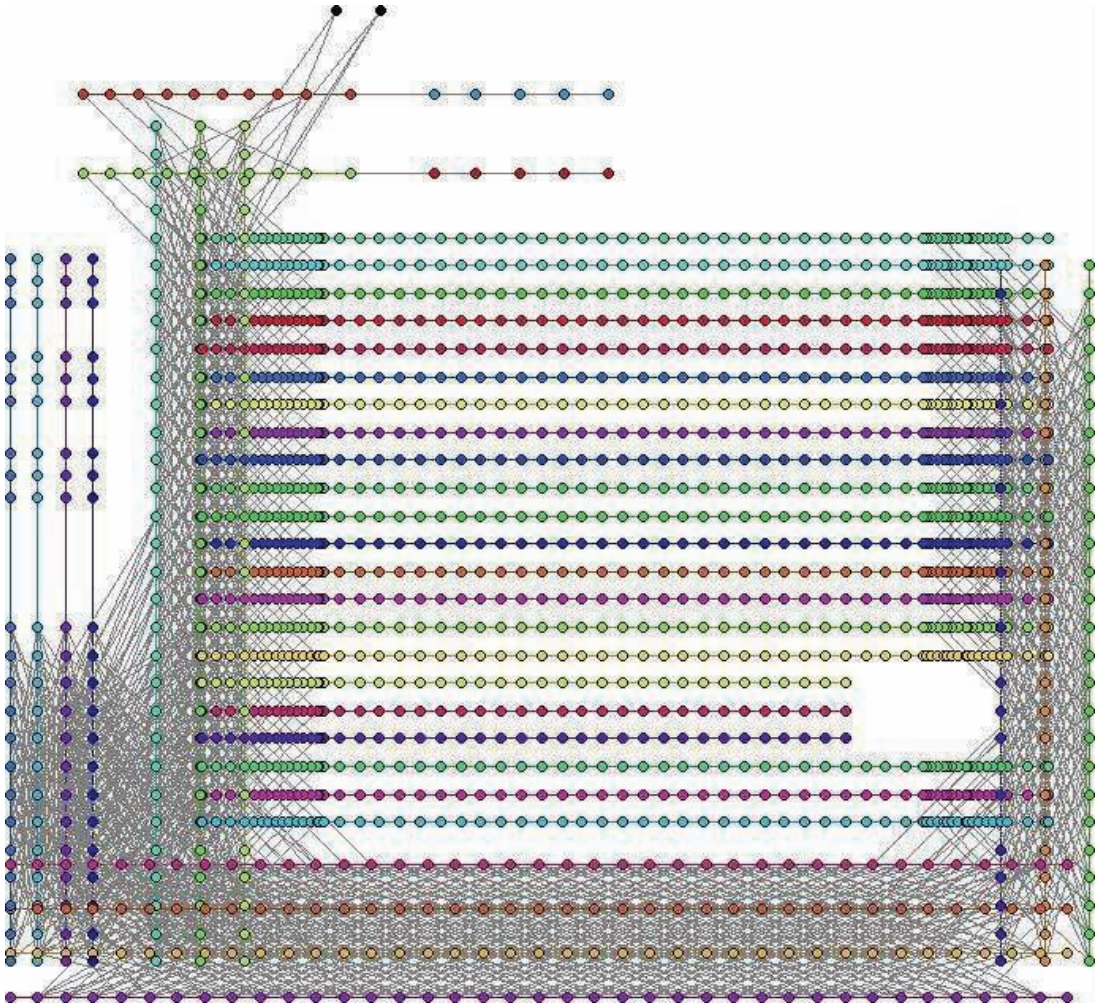

Figure 3: A map of the Patrick port facility at Port Brisbane.

5 Related Work

Hierarchical planning has been applied to path-planning before with abstractions such as approximate cell decomposition (Barbehenn & Hutchinson, 1995), (Conte & Zulli, 1995), generalised Voronoi graphs (Choset & Burdick, 1995), and general ad-hoc hierarchical maps (Zivkovic, Bakker, & Kröse, 2006), but the structures identified in these examples do not carry over well to the multi-robot scenario.

In the domain of task-planning the discovery and use of well-defined subproblems has shown recently popularity (Long & Fox, 2002; McCluskey & Simpson, 2004). The subgraphs I describe here may be considered 'generic types' for the multi-robot planning domain.

6 Conclusion and Further work

Abstraction has always been a valuable tool in improving the efficiency of planning, but the difficulty is in finding a good abstraction that is (i) commonly occurring, (ii) easy to recognise and represent and (iii) easy to reason about. In the domain of multi-agent path planning the hall subgraph is such an abstraction. Halls can be found in many different kinds of graphs and are especially common in maps of man-made domains such as corridors and roads. In domains such as these, it is relatively easy to identify halls by hand but the structure is also simple enough that automatic partitioning should not difficult to implement.

I have shown how this structure allows us to do planning in a large real world domain with many robots, where a naive centralised planner without abstraction rapidly runs out of memory. The key to this improvement is the efficient implementation of the ENTER, EXIT, TERMINATE and RESOLVE methods. This improvement is available without having to sacrifice completeness, unlike other solutions such as prioritisation.

That said, there is no reason why this abstraction should be incompatible with prioritised planning. A prioritised subgraph planner would have the advantage that plans could remain abstract until plans were constructed for all robots. By delaying some of the conflict resolution to the resolution step, the could allow the planner to produce a wider range of plans

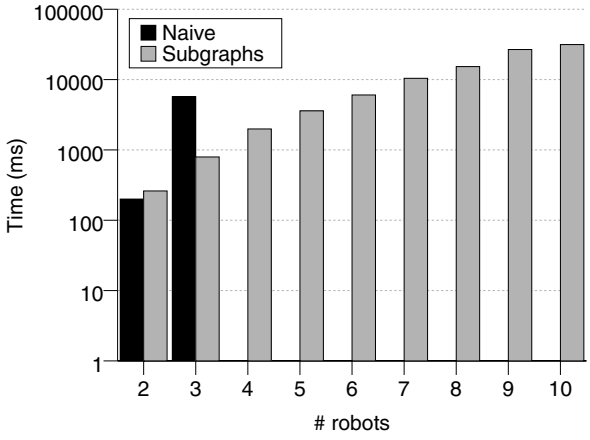

Figure 5: Median running times for planning in the port map. With more than three robots the naive algorithm ran out of memory before completing the task, so no result is shown.

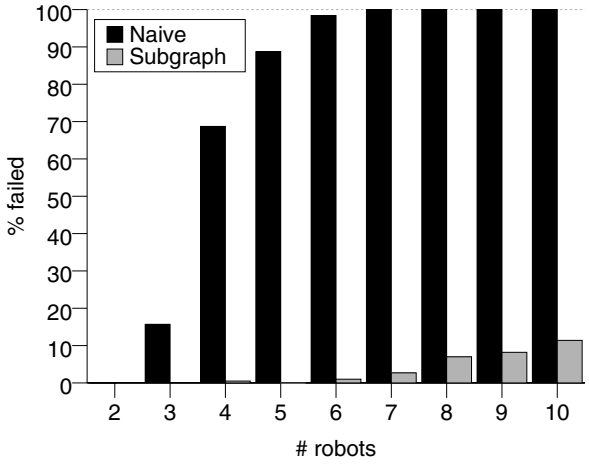

Figure 6: The percentage of planning tasks which exceeded the 1Gb memory limit.

than a naive prioritised planner. I intend to investigate this possibility as future work.

Another avenue for improvement would be to examine the symmetries created by the subgraph representation. Recent work in symbolic task-planning (Porteous, Long, & Fox, 2004) has shown that recognising and exploiting symmetries and almost-symmetries in planning problems can eliminate large amounts of search. Subgraph configurations provide a natural ground for similar work in our problem domain and we expect similar improvements are possible.

Acknowledgements

I wish to thank Daniel Pagac and the people at Patrick for providing me with the map data used in this project.

References

Barbehenn, M., & Hutchinson, S. (1995). Efficient search and hierarchical motion planning by dynamically maintaining single-source shortest paths trees. *IEEE transactions on robotics and automation*, *11*(2), 198-214.

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

Berg, J. van den, & Overmars, M. (2005). Prioritized Motion Planning for Multiple Robots. In *Proc. Int. Conf. on Intelligent Robots and Systems* (p. 430-435).

Choset, H., & Burdick, J. (1995). Sensor based planning. I. The generalized Voronoi graph. *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, *2*.

Conte, G., & Zulli, R. (1995, April). Hierarchical path planning in a multi-robot environment with asimple navigation function. *IEEE Transactions on Systems, Man and Cybernetics*, *25*(4), 651-654.

Erdmann, M., & Lozano-Pérez, T. (1986). *On Multiple Moving Objects* (Tech. Rep. No. 883). M.I.T. AI Lab.

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

LaValle, S. M., & Hutchinson, S. A. (1998). Optimal Motion Planning for Multiple Robots Having Independent Goals. In *IEEE Trans. on Robotics and Automation* (Vol. 14).

Long, D., & Fox, M. (2002). Planning with generic types. In G. Lakemeyer & B. Nebel (Eds.), *Exploring Artificial Intelligence in the New Millenium* (p. 103-138). Morgan Kaufmann.

McCluskey, T. L., & Simpson, R. (2004). Knowledge Formulation for AI Planning. In A. E.Motta N.Shadbolt & N.Gibbins (Eds.), *Engineering knowledge in the age of the semantic web* (p. 449 - 465). Springer.

Porteous, J., Long, D., & Fox, M. (2004). The Identification and Exploitation of Almost Symmetry in Planning Problems. In K. Brown (Ed.), *Proc. of the 23rd UK Planning and Scheduling SIG.*

Ryan, M. R. K. (2006). Multi-robot path planning with subgraphs. In *Proc. of the 19th Australasian Conference Robotics and Automation.*

Zivkovic, Z., Bakker, B., & Kröse, B. (2006). Hierarchical Map Building and Planning based on Graph Partitioning. *IEEE International Conference on Robotics and Automation*, 1.