Path planning is no substitute for intelligent behavior

David L. Jung[*] and Lynne E. Parker[**]
Center for Engineering Science Advanced Research (CESAR),
Computer Science and Mathematics Division (CSMD), Oak Ridge National Laboratory

ABSTRACT

This paper describes our experience implementing navigation behavior for two different autonomous multi-robot systems using two very different approaches. We describe the problems encountered and their solutions and the extensions necessary to support planning for multiple robots in our application domains. We conclude that there are many applications of path-planning that would be well served by the introduction of a little domain specific intelligent behavior as a substitute for brute force path planning over unnecessarily large configuration spaces.

Keywords: behavior, behavior-planning, path-planning, navigation, mobile robot

1. INTRODUCTION

This paper describes our experience implementing navigation behavior for two different autonomous multi-robot systems using two very different approaches. Although the robot systems and applications differ, there are many overlapping requirements for successful navigation. The first, classical path-planning approach, uses an algorithm similar to A^* (A-Star)[1] to search for a path through discrete states representing regions of physical space. The second approach, which we'll call behavior planning, also searches over discrete states, but the states represent both behavioral states and regions of space. We encountered a number of problems during implementation and we describe how they were overcome. The two approaches are compared and we discuss the application characteristics for which we believe each approach is best suited. We argue the case that there are many applications in which planning over large configurations spaces is often utilized, that would be better served by the introduction of some domain specific 'intelligent' behavior.

2. APPLICATION REQUIREMENTS

The first application, to which we applied the classical path planning approach, is a cooperative multi-vehicle mining task – developed in collaboration with an industry partner[†]. Although implementation of the system was planned for full-scale production vehicles, the current research was performed using a complex 3D distributed simulation system. The vehicles are tracked and hence utilize skid steering. The vehicles are almost twice as long as wide and frequently need to navigate 'corridors' in the outdoor terrain that are barely wider than the vehicles themselves. The terrain is highly dynamic – as the vehicles modify it during operation. The task places fuel usage and time at a premium, so ideally path lengths need to be minimized. The problem can be stated as finding a set of shortest path trajectories from each vehicle's current position to a specified (per vehicle) goal position, that avoid vehicle collisions. Each of the vehicles has onboard computation and can communicate with all others over a moderate bandwidth wireless network. The vehicles have an array of sensors, including a ranging device that provides information about the terrain in the vicinity of the vehicle. This information is broadcast piecemeal to the other vehicles, giving each an approximate knowledge of the terrain within the work area. Each vehicle also knows the location (via DGPS) and current behavioral state of the others.

The second application requires navigation around an indoor office environment for cleaning (the system has been previously reported[2,3]). The system consists of two mobile robots approximately 30x50cm and 50cm high. Each has two wheels centered along its length, using differential steering, and passive casters at the front and rear (see Figure 1).

[*]david.jung@pobox.com; phone:1 865 2412985; [**]parkerle@ornl.gov; phone: 1 865 2414959; http://cesar.ornl.gov; Oak Ridge National Laboratory, Oak Ridge, TN, USA, 37831

[†] Unfortunately, due to a non-disclosure agreement is it only possible to describe the project in the most general terms.

As for the mining task, the cleaning task requires avoiding collisions and operating in tight places where turning is restricted. Although shortest paths are preferred when navigating to a goal position, it is not as critical as in the mining task. The office environment is largely static, with some mobile objects such as people and chairs. This contrasts with the dynamic terrain of the mining environment.

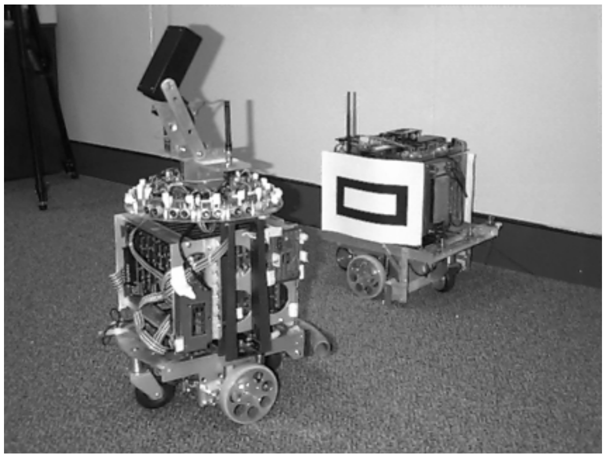
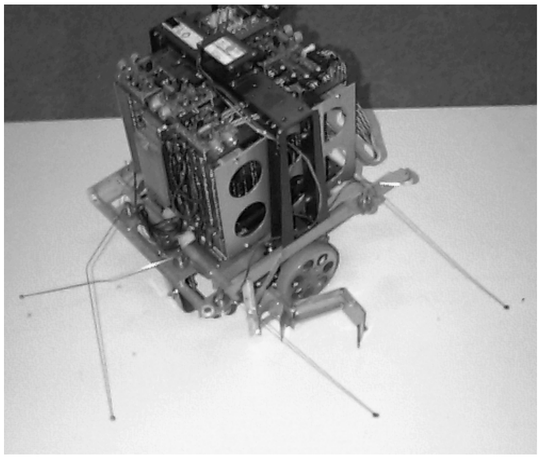


Figure 1 – (a) One of the cleaning robots 'observing' (using vision) the other robot following a wall boundary


(b) The tactile sensors designed specifically for the *Wall-following* basic behavior.

3. PATH PLANNING

Our first thought for a solution to the mining navigation problem was to implement the A^* algorithm[1] – or its dynamic cousin the focused D^* algorithm[4,5]. These algorithms are equivalent to searching for the shortest path though a connected graph using Dijkstra's graph search algorithm[6], but achieve better average performance by using a heuristic to guide the search (typically an estimate of the remaining path length). A^* assumes a static environment during the planning process. A dynamic environment can be accommodated by re-planning from the current location each time the environment changes. Since this is expensive, the D^* algorithm was designed to be equivalent to re-planning but is significantly more efficient in such a dynamic context because information from previous searches is retained.

3.1 Representation

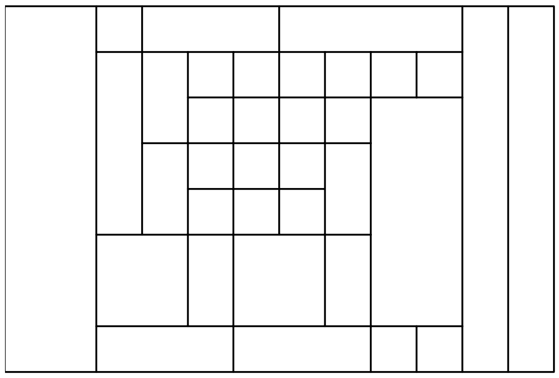
The performance of D^* depends on a number of things, such the size of the configuration state space (C-space), and importantly, its dimension and representation. The algorithm can search a C-space of any dimension. We denote the dimension of the C-space for a single vehicle N. For a single vehicle in the mining application, we used N=3 – we searched for a trajectory over a discretized (x,y,θ) space, representing 2D Cartesian position and angular orientation. The obvious way to plan a set of compatible trajectories for M vehicles, is to multiply the C-space dimensions for each vehicle together to form a high-dimensional joint C-space of dimension $N\times M$. Clearly, such a joint C-space will be prohibitively large except for small vehicle teams with low dimensional and a low resolution C-space for each robot. For this reason we initially focused on planning a trajectory for a single vehicle – and we will restrict discussion to this case until a later section.

Clearly, with the possibility of needing to plan over such a high dimensional C-space we require a discretization that reduces the number of states as much as possible. A common choice is to use an *exact cell decomposition* (in the terminology used by Latombe[1], Chapter 5; a cell meaning a single C-space state). In this representation scheme the free space is divided into states such that the boundary of an obstacle state is comprised of only boundary sections of free space states. The free space states typically have simple geometry and the adjacency relationships between them are used as legal edges between states through which the search can proceed. The assumption for this method is that the environment is mostly static, allowing the expensive decomposition to be performed off-line, saving runtime computation. Unfortunately, for our mining application the terrain is continuously dynamic and has no sharp delineation between obstacle and free states. Consequently, any such decomposition would have to be largely repeated before each trajectory search, rendering the method too expensive and hence unsuitable.

3.2 Grids

The simplest grid representation to implement is the regular, axis-aligned grid – where each dimension is subdivided into equal length parts. Typically, applications require highest resolution near obstacle boundaries, which determines the resolution of the whole regular grid. The result is a large number of states and hence low search performance.

To strike a balance between the large number of states and cheap decomposition of a regular grid and the reduced number of states but expensive decomposition of an *exact cell decomposition* scheme, we implemented an irregular grid with hyper-rectangle axis-aligned cells (see Figure 2a for an example in 2D). This was implemented via a data-structure that initially contains a single state for the entire C-space and allows arbitrary split and merge operations. A split produces a new boundary at a specific value of a specific dimension over specified intervals in the other dimensions. A merge is the reverse of this, removing boundaries in a specific dimension over specified intervals in all other dimensions. This representation has advantages for planning over the joint C-spaces of multiple vehicles, as will be discussed below.



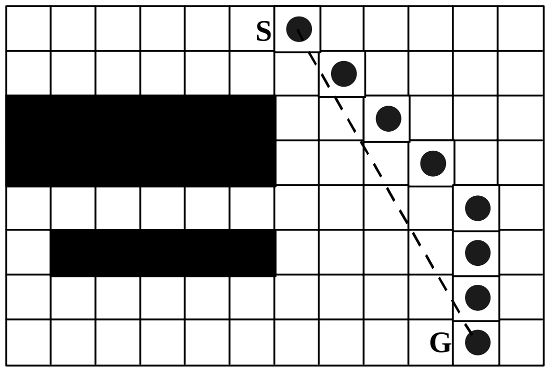


Figure 2 – (a) A 2D irregular grid with rectangular cells (b) A problem with rectangular discretization.

There are a number of problems associated with using a rectangular grid. Consider the regular grid (for simplicity) shown in Figure 2b. The white grid cells represent valid configuration states and the black cells invalid states – for example, those in which the vehicle would occupy the same spatial extent as an obstacle. Note that each C-space element corresponds to a 2D *region* of the real space the vehicle can occupy. Hence, which C-space configurations are marked invalid (black) depends not only on the environment (e.g. the positions and extents of obstacles), but also on the shape of the robot. The task of the planning algorithm is to find a sequence of states from the start configuration (S) to the goal configuration (G) that minimizes a specified cost function – Cartesian distance in this example. One problem is that even though the solution that minimizes the cost function over the continuous space is unique, the minimum-length sequence through C-space may not be. In the figure the dashed line has length ~8.1 units, while one of the shortest C-space sequences (round dots) is ~8.7 units in length. Which of the equally shortest C-space paths is selected depends on the path-planning algorithm implementation – in many implementations it may oscillate between alternatives upon successive searches.

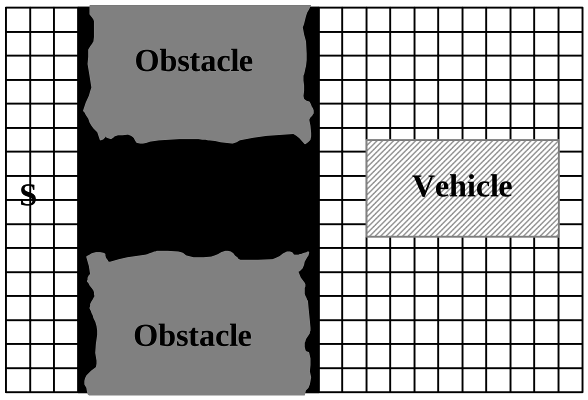


Figure 3 – No path exists if the grid discretization is too coarse.

Another problem was encountered when the trajectory must pass through a corridor just wide enough for the vehicle to pass. In this case, the grid cell size needs to be at least as small as the tolerance on either side of the vehicle or no path will be found. This results in an excessive number of states. The schematic in Figure 3 shows the problem. Even though the vehicle can just fit through the space between the two obstacles, there is no C-space configuration that is valid between them. This is because a C-space configuration is marked invalid if *any* configuration within the real space region it represents is invalid (i.e. the vehicle overlaps the obstacle, in our case). The problem is particularly evident with a long vehicle when the orientation is discretized such that no states exist that are centered on the orientation of the corridor.

3.3 Planning for multiple vehicles

As mentioned previously, an obvious way to search for a set of trajectories for multiple vehicles is to multiply the C-spaces of each vehicle together into a higher dimension joint C-space. This clearly leads to an exponential growth of the number of states in M (the number of vehicles) – and correspondingly unacceptable search performance. Instead we chose to trade optimality for performance. The multi-vehicle search is conducted in a number of steps. It is quite common for the set of independently shortest paths not to intersect – hence, in the first step each vehicle plans a path without regard for the other vehicles. Each vehicle communicates its planned trajectory to the others and also does a pair-wise intersection test between its own trajectory and those of the other vehicles. If an intersection between a pair of trajectories is detected it implies a *possible* future collision. For the mining application, the three most common cases of independently planned intersecting trajectories can be characterized by Figure 4. The start and goal positions of vehicles 1 and 2 in each case are, respectively, S1, G1, S2 and G2.

In Figure 4a, the independently planned trajectories (not shown) are the straight lines passing through the corresponding start and goal positions – which partially overlay each other. The solution is to find a new set of trajectories that do not intersect at all (shown). This is accomplished by re-planning while treating the other vehicle as an obstacle. Such solutions may not be globally optimal. Figure 4b shows a case where the solution involves the same trajectories, but one of the vehicles must stop and wait during the execution of the trajectory, for the other vehicle to pass. This solution can be obtained by searching for a path through a 2D C-space where each dimension represents a parameterization of the distance along the existing independently planned trajectories of each vehicle. The invalid states of the C-space correspond to where the existing trajectories intersect. These two cases are, by far, the most common situations encountered in the mining application. Which one is the best solution depends on task context. Hence, task context is used to choose which to try first.

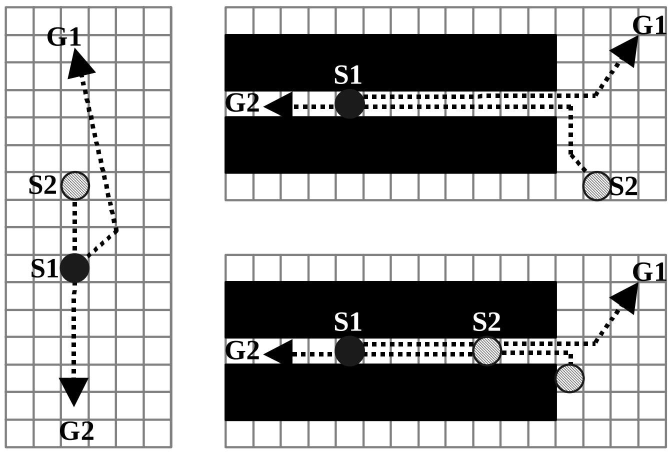


Figure 4 – Solutions to the three most common cases of independently planned intersecting trajectories (clockwise from left): (a) Find non-intersecting trajectories. (b) Vehicle 2 must wait before proceeding into the 'corridor'. (c) Vehicle 2 must back-out of the 'corridor' and allow vehicle 1 to pass before continuing in to its goal.

If the solution method for cases a and b both fail to find a valid path, the vehicles could be in the situation depicted in Figure 4c. Here, the solution involves both waiting and the generation of a new trajectory. Our method is to combine the C-space of the two vehicles into a joint C-space, and search for a set of two trajectories over that C-space. This is an expensive computation. Although the result is a path optimal with respect to the cost function, it is not really optimal as the vehicles hesitate for a reasonable time before moving at all (while performing the computation).

Our irregular grid representation is well suited to plan over the joint C-space of the two vehicles. Notice in Figure 2a that the left and right ends of the horizontal dimension have a vertical extent that spans the entire space in the vertical dimension. This means that the C-space is effectively 1-dimensional in these regions. When planning the independent trajectory for a single vehicle a C-space of dimension $N\times2$ is actually used (the dimension of a 2-vehicle joint C-space). However, analogously with the 2D example, the extents of the states in every dimension representing the second vehicle span the entire dimension – effectively eliminating the second vehicle from consideration. When the two vehicles must be considered jointly, a series of split operations are performed to expand the number of states to equally represent both vehicles. After planning is complete, merge operations restore the C-space to the reduced number of states adequate for a single vehicle again.

Finally, there are also instances where even this search may fail – for example, if three or more vehicles have trajectories that intersect in particular ways. However, we have found these cases to be rare. The only solution is to create even higher dimensional C-spaces by combining the C-space of all the vehicles involved – which is prohibitively expensive. Our ad hoc approach in these cases is to start the vehicles in motion along their planned paths and try to re-plan at a later time. If this also fails, a human operator is notified (and can tele-operate the vehicles to resolve the deadlock).

In conclusion, we found the above path planning algorithms and data-structures moderately difficult and time consuming to develop – especially having to find solutions to work around the several problems encountered. We were also always struggling to keep the computation time within acceptable limits.

4. BEHAVIOR PLANNING

The following section describes a different approach to a similar navigation problem. We turn now to the solution within the second application domain – cooperative cleaning. As mentioned, the task involves two mobile robots that must navigate around a typical office building environment while cooperating to clean the floor. The details of the cooperation and cleaning are not important here and have been reported elsewhere[2]. The major qualitative difference from the mining task is that the environment is mostly static – with only intermittently moving obstacles, such as people, moved chairs and other furniture and the other robot.

4.1 Basic navigation behavior

For reasons incidental to navigation, the cleaning system was implementing using a behavior-based architecture. The architecture was layered, with path-planning level navigation competence only appearing in a upper layer. The lower layers of the system consist of various behaviors that allow the robots to accomplish their task, although very inefficiently. Specifically, some of the behaviors are wall following, turning a corner, passing though a door, visual servoing toward a target, moving in a straight line for a short distance, collision avoidance and others. The robots could 'navigate' throughout the office using only basic behaviors by essentially invoking them randomly. Obviously the robots have no capability to purposively navigate from one arbitrary location to another in this mode of operation (or to represent an arbitrary location at all).

Many of the basic behaviors involve motion of the robot. Hence they can be considered limited 'navigation' in some sense. For example, the wall following behavior is invoked when the robot senses it is against a 'wall' – if suitably obstacle free, the robot will then successfully 'navigate' along the length of the wall. Similarly, the visual servoing behavior can successfully 'navigate' from the robot's current location to that of a visually recognizable target within range.

The nature of the cleaning task is such that performance can be greatly enhanced if the robots can build, maintain and navigate via a map of their environment. It is upon the basic behaviors that a path-planning style navigation capability is layered. Before describing how the layering is achieved, the following section first discusses an alternative algorithm for path planning over discrete C-spaces – which forms the basis for the behavior-planning scheme detailed subsequently.

4.2 Spatial path planning

Path planning can be visualized by analogy with the solution of the 2D heat-equation. Imagine a 2D Cartesian configuration space over which we wish to plan a shortest path. Represent this space with a flat sheet of metal. Attach a heat source at the goal location and heat sinks everywhere that corresponds to obstacle configurations. If initially the sheet is cold, then over time heat will dissipate from the source until the sheet reaches an equilibrium state. Now, starting from *any* position on the metal sheet, the shortest distance to the goal can be computed by following the steepest temperature gradient to the goal (the hottest point).

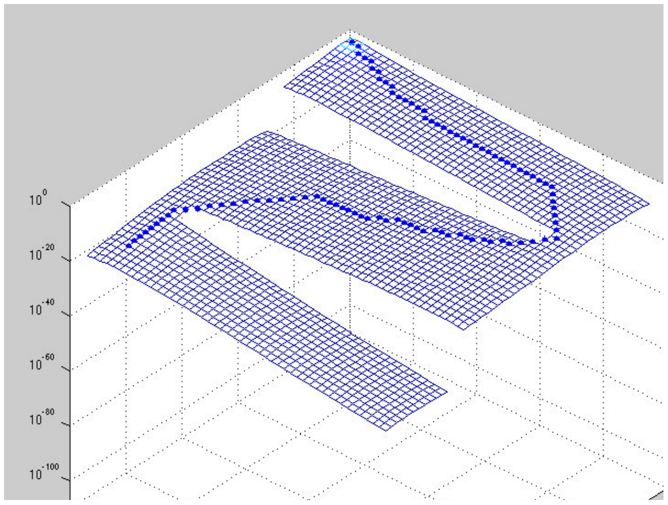
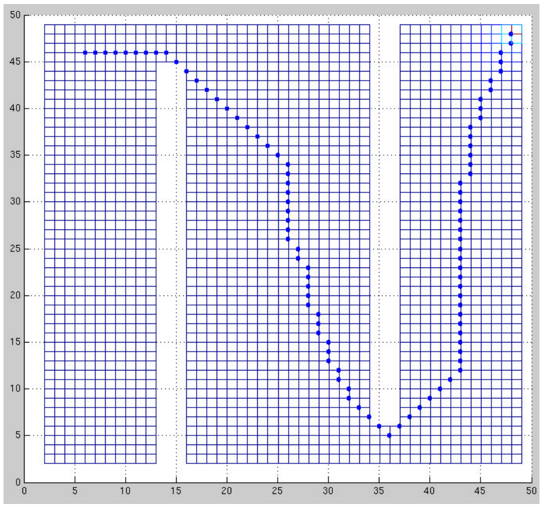


Figure 5 – (a) Potential function generated for optimal path computation (with path overlaid). The function values are analogous to temperature – hottest at the top



(b) Path computed by optimal potential field method over a 50×50 regular grid (white areas without grid-lines denote obstacle configurations).

The 2D heat-equation is a simple partial differential equation that can be solved numerically via standard iterative methods. For example, by discretizing into a regular grid, iterating over each element and updating it using the forward Euler formula approximation until equilibrium is reached. This formulation effectively updates each element by averaging its four immediate neighbors. An example of the potential function yielded over a 2D 50×50 regular grid is shown in Figure 5.

Glasius et. al. published a path-planning method formulated in terms of a Hopfield type neural network[7]. On closer inspection the formulation can be seen to be another instance of this dissipation type scheme (in fact the formulas can be made identical with the forward Euler formulation via substitutions and some additional assumptions). Glasius showed it is optimal and that it is not necessary for the elements to be updated in any particular order to reach equilibrium. The update order only affects the convergence rate. Lagoudakis and Maida have suggested other improvements to this method[8]. As the equilibrium state corresponds to the set of shortest paths from *any* position to the goal (in our example), the computation can be terminated as soon as the start position configuration potential value is updated. Also, the pattern of updates over the C-space elements can be specifically designed to speed convergence for likely paths (e.g. those that include the start position).

This algorithm closely resembles part of the spreading activation[9] method of planning that we employed for behavior selection in our behavior-based architecture, as described below. This type of algorithm was chosen because it is amenable to a neural-like implementation, whereas the A^* family of algorithms are typically implemented in a procedural fashion.

4.3 Topological path planning

Notice that both the A^* family of algorithms and the dissipation/spreading-activation algorithms only require a discrete set of states and information about their adjacency. It doesn't matter if the adjacency of two states represents a spatial relationship or something else entirely. It is this property that enables the use of these algorithms to plan behavior sequences and paths through topologically represented maps. To make this concrete, consider the following simple example (illustrated in Figure 6). The figure shows the corner of a room overlaid with a 2D spatial regular grid. The squares represent the C-space states and the lines between them spatial adjacency – all as before. In addition there are lines representing behavioral adjacency. To move between two states that are spatially adjacent, the robot needs to activate a behavior that can move it by a small amount along the appropriate connecting trajectory (ignoring trajectory smoothing for the moment). To move between two states that are behaviorally adjacent, the robot needs to activate the appropriate behavior. So to move from the start to the goal position, in the example, the sequence of behaviors 'Follow Wall', 'Corner', 'Follow Wall' would be valid. A sequence of 10 moves along short trajectories, between spatially adjacent states, would also be valid.

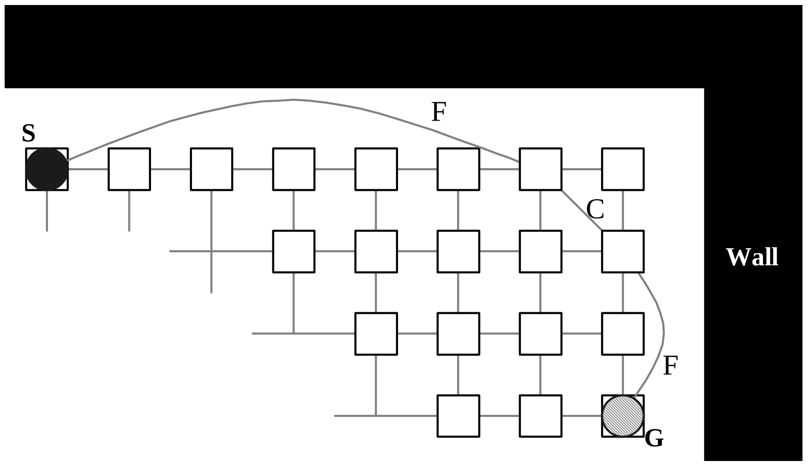


Figure 6 – Exploded view of C-space states showing adjacency (lighter lines). All lighter lines represent spatial adjacency except those labeled **F** for *Follow Wall* behavioral adjacency and **C** for *Cornering* behavioral adjacency.

It is evident from the example that a path can consist of transitions between states that require arbitrary basic behaviors to be executed to enact it. This provides great flexibility. For example, a single plan could consist of a sequence whereby the robot navigates to another office, uses manipulation behaviors to grasp a waste paper basket, navigates to a recycling repository, deposits the paper and returns the basket to its original location. If the basic behaviors are well chosen, such a sequence may consist of a small number of states.

It is clear that using behavior planning with basic behaviors appropriate to the application domain can dramatically reduce the number of discrete C-space states needed. The small number of states allowed us to use the Spreading Activation planning algorithm without concerns over performance.

4.4 Grids again

If only a topological map utilizing behavioral adjacency information were used, the robot would never be able to navigate to locations for which no basic behaviors result in being at that location[‡]. Consequently our cleaning system also utilizes an irregular a grid of states distributed over the floor space. Although the grid is initially organized as a regular grid, with 8-way spatial adjacency represented, it is continuously adapted using a Kohonen Self Organizing Map[10].

Fortunately, the problems with using rectangular axis-aligned grid elements, discussed previously in Sec. 3.2, don't appear. As in the irregular grid for the mining implementation, only coarse resolution is necessary within regions of free space. Unlike the mining implementation there is no need for higher resolution spatial decomposition near the boundaries of obstacle states, due to the basic behaviors. Specifically, the spatial component of any planned paths only needs to move the robot to within the approximate vicinity of obstacle boundaries. Once the robot is near the boundary of an obstacle a basic behavior can be triggered. For example, no high resolution spatial planning is necessary to move the robot through a doorway, as moving the robot within visual recognition range of the doorway and triggering the *pass-through-door* behavior robustly guides the robot through the door (using visual servoing). Similarly, navigating to within range of a wall is adequate for the follow-wall behavior to 'navigate' the robot along the length of the wall (using tactile sensors specifically designed for the purpose).

4.5 Multiple robots

No specific techniques for dealing with multi-robot planning were implemented for the cleaning application. Due to the nature of the application, most of the cooperative interactions between the robots are controlled by dedicated basic behaviors when the robots need to interact. Situations in which the trajectories of the two robots interfere are not specifically handled due to lesser importance placed on optimality than for the mining application. The basic behaviors will always get the robots out of any deadlocked situation, but it will not be via an optimal joint path. Since there are only two robots, more complicated situations don't occur.

So, the use of behavior planning using a map that represents both spatial and topological relationships between C-space states dramatically reduces the number of states needed (provided the basic behaviors are appropriate for the application domain). Because of the relatively small search space size we were able to utilize the spreading activation planning algorithm, despite its low performance. Planning takes such a small portion of the computation in the system, any optimizations for performance would be insignificant gains in comparison to the added complexity of the implementation.

5. A COMPARISON

The major difference between the two applications is the premium on optimal paths in the mining application. This was the motivation for selecting an algorithm from the A^* family – because of their superior performance. Unfortunately, our inability to use an exact cell decomposition of the space significantly increased the number of states over which the system searches. Re-planning is performed often, as are modifications to the spatial decomposition and re-evaluating whether states are obstacle states or free states. The number of states required was also dramatically increased through the requirement to commonly navigate the vehicles through tight 'corridors' – which required a higher resolution decomposition at the boundary of the obstacle states. All these factors contributed to path-planning performance always being a limiting factor during development.

[‡] Actually, spatial adjacency is usefully considered just a special case of behavioral adjacency where the behavior is a simple one that moves the robot between the neighboring positions in an appropriate way (for example, smoothly).

To perform multi-vehicle planning, simply combining the C-spaces of multiple vehicles together produces a prohibitively large joint C-space. However, by trading guaranteed optimality for near optimality in most cases, we were able to extend the system to multi-vehicle planning in a reasonably straightforward manner. Planning in the joint C-space for even two vehicles was pushing the limits of computation – and three vehicles or more in this general case was prohibitive.

The use of basic behaviors in the cleaning application, to perform elementary 'navigation' tasks in situations that are the cause of most performance of the problems in the mining application, greatly reduced the necessary number of C-space states. We implemented the system using a planning algorithm that is extremely inefficient in comparison to the A^* family. However, due to the very short state sequences, this did not significantly degrade performance. In addition is it much easier to implement. For applications with larger C-space sizes, D^* could be used for behavior planning with equal facility.

We have not investigated in detail extension of the behavior planning system to solve some of the multi-robot planning situations encountered in the mining application. However, it seems intuitive that similar methods could be applied. It may even be practical search over joint C-spaces involving more robots, due to the much-reduced size of each robot's C-space (but due to the exponential increase this will only get you so far). The presence of basic behaviors removes the uniformity of spatial-only adjacency relationships between C-space states. This could complicate the combining of multiple robot C-spaces in unforeseen ways.

6. CONCLUSION

We have shown that for many applications, the introduction of a little intelligent behavior into the planning space – in the form of domain specific basic 'navigation' behaviors – can dramatically reduce the storage and computational requirements of a path-planning implementation. It also results in a system that is more flexible – by being able to plan arbitrary sequences of navigation-like behavior interleaved with any other behavior in the robot's repertoire, simpler and easier to implement. In hindsight, we believe our mining application would have benefited significantly if we had applied the behavior planning approach. It is not proven that the behavior planning approach could have completely accommodated the additional needs of the mining application, with regard to optimality of multi-robot plans, but we believe these are issues that can be successfully addressed.

There would also intuitively seem to be a continuous middle ground between the two approaches. Both methods have a clear separation between the search algorithm and the C-space representation. All the algorithms discussed produce optimal paths given a set of states and their adjacencies – they differ only in their performance optimizations. The case of a purely spatial C-space representation, as employed in the mining application, can be considered a special case of more general adjacency relationships between states. Clearly there is a continuum – between large C-space sizes used in conjunction with search algorithms and data structures that are optimized for simpler adjacency structure; and smaller C-space sizes used with search algorithms and data structures that can accommodate more arbitrary adjacency between states.

A browse of the literature reveals a huge number of applications for which the A^* and related search algorithms have been applied. We believe that the majority of implementations would be well served by the introduction of a little domain specific intelligent behavior as a substitute for brute force path planning over unnecessarily large C-spaces.

ACKNOWLEDGMENTS

This research was funded by the Engineering Research Program, Office of Basic Energy Sciences, of the U.S. Department of Energy, under contract DE-AC05-00OR22725 with UT-Battelle, LLC. The submitted manuscript has been authored by a contractor of the U.S. Government under the above contract. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

REFERENCES

1. Latombe, J-C, *Robot Motion Planning*, appendix C, Kluwer Academic Publishers, Boston USA, ISBN 0-7923-9129-2, 1991.
2. Jung, David and Zelinsky, Alexander, "Integrating Spatial and Topological Navigation in a Behavior-Based Multi-Robot Application", *proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS99), pp323-328, Kyongju Korea, October 1999.
3. Jung, David and Zelinsky, Alexander, "Grounded Symbolic Communication between Heterogeneous Cooperating Robots", Autonomous Robots, **8(3)**, pp269-292, Kluwer Academic Publishers, June 2000.
4. Stentz, A., "Optimal and Efficient Path Planning for Partially-Known Environments", *Proceedings of IEEE International Conference on Robotics and Automation*, vol.4, pp3310-3317, IEEE Press, San Diego CA, USA, 1994.
5. Stentz, A., "The Focused D* Algorithm for Real-Time Replanning", *Proceedings of International Joint Conference on Artificial Intelligence*, Morgan Kaufmann Publishers, Montreal Canada, August 1995.
6. Dijkstra, E.W., "A note on two problems in connection with graphs", Numerical Mathematics, **1**, pp269-271, 1959.
7. Glasius, R., Komoda, A. and Gielen, S., "Neural network dynamics for path planning and obstacle avoidance", Neural Networks, **8(1)**, pp125-133, 1995.
8. Lagoudakis, Michail G. and Maida, Anthony S., "Neural Maps for Mobile Robot Navigation", *proceedings of the International Joint Conference on Neural Networks*, Washington DC USA, July 1999.
9. Maes, P., "Situated Agents Can Have Goals", *Designing Autonomous Agents*, Maes, P., MIT-Bradford Press, ISBN 0-262-63135-0, 1991.
10. Kohonen, Teuvo, "The self-organising map", *Proceedings of IEEE*, **78(9)**, pp1464-1479, September 1990.