Mobile Robot Motion Planning
in Multi-Resolution Lattices
with Hybrid Dimensionality

Janko Petereit, Thomas Emter and Christian W. Frey

Fraunhofer Institute of Optronics,
System Technologies and Image Exploitation IOSB
Karlsruhe, Germany
e-mail: {janko.petereit, thomas.emter,
christian.frey}@iosb.fraunhofer.de

Abstract: Safe and efficient path planning for mobile robots in large dynamic environments is still a challenging research topic. In order to plan collision-free trajectories, the time component of the path must be explicitly considered during the search. Furthermore, a precise planning near obstacles and in the vicinity of the robot is important. This results in a high computational burden of the trajectory planning algorithms. However, in large open areas and in the far future of the path, the planning can be performed more coarsely.

In this paper, we present a novel algorithm that uses a hybrid-dimensional multi-resolution state × time lattice to efficiently compute trajectories with an adaptive fidelity according to the environmental requirements. We show how to construct this lattice in a consistent way and define the transitions between regions of different granularity. Finally, we provide some experimental results, which prove the real-time capability of our approach and show its advantages over single-dimensional single-resolution approaches.

Keywords: Path planning, motion planning, variable dimensionality, multi-resolution, state lattice, dynamic environment

1. INTRODUCTION

Mobile robot motion planning in large and dynamic environments is still a challenging research topic. In real-world scenarios with wide unstructured areas, path planning algorithms must be able to quickly generate even long paths that are as optimal as possible with regard to a given optimality criterion like minimal time, distance, or energy consumption. To make things worse, the environment in which the robot is acting will generally not be a static world but will change rapidly over time. For example, after a natural disaster or industrial accident, the robot has to share its workspace with human rescue workers and other vehicles. Thus it is necessary for the robot to reliably avoid any collision with these dynamic obstacles. Early approaches were based on the concept of two separate planning layers: A global path planner found a path to the goal while ignoring all dynamic obstacles and an underlying obstacle avoidance algorithm operated in the vicinity of the robot in order to modify the global path if necessary due to an imminent collision. Clearly, such a planning strategy leads to a sub-optimal overall quality of the resulting trajectory.

In order to guarantee a globally optimal trajectory (with respect to the search space discretization and the quality of the prediction of the future motion of the dynamic obstacles), a path planning algorithm has to incorporate all dynamic obstacles directly into its planning. This results in a complex motion planning problem with a potentially high dimensional search space. To cope with this computational complexity, a lot of research effort was undertaken during the last decade. The well-known Rapidly Exploring Random Trees (RRTs) have been applied to kinodynamic motion planning by LaValle and Kuffner (2001). Hsu et al. (2002) used Probabilistic Roadmaps (PRMs) for kinodynamic planning in dynamic environments. Both approaches have in common that they only guarantee probabilistic completeness. Furthermore, they generally focus on merely finding any path, however, in most mobile robot navigation tasks, finding the optimal path is desirable. Another approach comes from Knepper et al. (2010), who employ a hierarchical planning concept, in which an approximate global planner provides input to an underlying local and more exact planner. The practical applicability of their algorithm heavily depends on the speed of the global planner, thus allowing only for very simple global planning strategies (e.g. simple grid search), which sacrifices global optimality in favor of speed.

To guarantee global optimality (with respect to the quantization), we decided to build upon the successful history of state lattices planners (cf. Pivtoraiko et al., 2009), which allow for an efficient formulation and solution of the motion planning problem. For this purpose, we propose a novel algorithm which is capable of adaptively changing the dimensionality and resolution of the lattice during the graph search depending on the required planning accuracy.
1.1 Problem Statement

A path planning that is as optimal as possible with respect to the available knowledge of the environment, is essential for the successful application of mobile robots. Furthermore, the path planning algorithm must be able to quickly generate paths in order to cope with dynamic obstacles and reliably avoid collisions. Especially for the planning of long paths, these are two conflicting requirements due to the immense computational complexity. However, in a practical application, the motion of the robot and its perception of the environment (especially in distant areas) is subject to noise. Therefore, it is reasonable to relax the requirements imposed on the accuracy of the path with increasing distance and time. If these different requirements are already considered during the planning phase, the computational complexity of the path planning problem can be significantly reduced without sacrificing a high solution quality in the local vicinity of the robot.

When planning in state lattices, there are mainly two options for controlling the quality of the resulting paths. The first possibility is the variation of the dimensionality of the search space. For example, the search can start in a full-dimensional state × time space, continue through a state space which allows dynamically feasible motions, and finally use a state space which represents merely kinematically feasible motions. This procedure results in a successive reduction of the search space dimensionality. The second possibility to control the path quality is the adaption of the state lattice resolution. The resolution of the state lattice can be chosen to whatever is appropriate for a particular region independently from the currently used dimensionality. For example, a high resolution could be chosen in the vicinity of the robot or near obstacles, and a low resolution could be chosen in wide open areas.

In this paper, we propose a novel algorithm which combines these two complexity reduction strategies in a single integrated approach to get the best out of two worlds.

1.2 Related Work

Subcomponents of our proposed algorithm have been subject to extensive research in the last few years and constitute a valuable input for our concept. For example, Likhachev and Ferguson (2009) introduced a multi-resolution approach for their state lattice planner which uses a finer discretization in the vicinity of the robot and near the goal. Rufli et al. (2009) extended this approach to narrow regions of the environment.

Gochev et al. (2011, 2012) presented an approach for planning in high-dimensional state lattices, which uses a strategy for adapting the dimensionality of the search space to guarantee a solution that is still feasible with respect to the high-dimensional motion model while nonetheless making fast progress in regions of the search space that exhibit only a low-dimensional structure. Zhang et al. (2012) extended this method in the context of mobile robots.

With the introduction of time-bounded lattices, Kushleyev and Likhachev (2009) have laid the foundation for the efficient consideration of dynamic obstacles for planning in lattices with variable dimensionality, however, their solution does not provide kinematic feasibility for the entire path. In recent work (Petereit et al., 2013) we extended this approach in order to always guarantee at least kinematic feasibility and allow for a consistent modeling of dynamic and static obstacles. Gonzalez et al. (2012) proposed a method for reducing the computational complexity by exploiting what they refer to as state dominance.

2. STATE × TIME LATTICE CONSTRUCTION

In this section, we will describe the process of constructing a state × time lattice with variable dimensionality and multiple resolutions. It is defined in terms of several motion primitive sets whose design and connections are explained in detail in the following. The construction of the motion primitive sets is entirely performed off-line. The motion primitive sets are cached and are made available for the subsequent online motion planning.

2.1 Robot Motion Model

All motion primitives reflect feasible motions of the associated dynamic system. We will exemplarily demonstrate the efficient generation of motion primitives for a mobile robot with four-wheel steering. It can be described by the following system of differential equations.

\[
\begin{align*}
\dot{x} &= v \cos \theta \\
\dot{y} &= v \sin \theta \\
\dot{\theta} &= c v \tan u_\beta \\
\dot{v} &= u_a
\end{align*}
\]  

The system states are the robot's position \(x\) and \(y\), its heading \(\theta\), and its translational velocity \(v\). The constant \(c\) equals twice the inverse of the wheelbase. The steering angle \(u_\beta\) and the translational acceleration \(u_a\) serve as the input to the dynamic system. For constant inputs \(u_a\) and \(u_\beta\), the system of differential equations (1) can be integrated analytically, which allows for a very efficient simulation of the vehicle dynamics.

2.2 Lattice Construction Basics

In order to generate the motion primitives, first a suitable quantization \(q\) of the state × time space has to be chosen. As the motion due to (1) exhibits “translational invariance” with respect to the position and time, we restrict the admissible values to a regular grid with the corresponding discretization step sizes \(\delta_x^q\), \(\delta_y^q\), and \(\delta_\theta^q\). Thus, the resulting sets of admissible values are simply given by

\[
\begin{align*}
X^q &= \{ n \delta_x^q : n \in \mathbb{Z} \}, \\
Y^q &= \{ n \delta_y^q : n \in \mathbb{Z} \}, \\
T^q &= \{ n \delta_\theta^q : n \in \mathbb{N}_0 \}.
\end{align*}
\]

Similarly, the discretization \(\delta_\theta^q = 2\pi/n_\theta^q\) of the robot heading \(\theta\) is chosen; however, for symmetry reasons, it is beneficial to require the number of discrete heading directions \(n_\theta^q\) to be a multiple of 4.

\[
\Theta^q = \{ m \frac{2\pi}{n_\theta^q} : m \in \mathbb{N}_0 \land m < n_\theta^q \}
\]

The set of discrete velocities is obtained by directly specifying the individual values. In order to allow the robot to perform a “wait” action, the zero velocity has to be a member of this set.

\[
V^q = \{ 0, v_1^q, v_2^q, \ldots \}
\]
After having determined an appropriate state \( x \times t \) space discretization, the actual motion primitive sampling has to be performed. This process is described in detail in Peteriet et al. (2013), however, for the sake of completeness, we restate a short outline of the procedure below. Firstly, we generate a huge number of motion primitive candidates by integrating the system model (1) using randomly sampled piecewise constant inputs over a time span that is a multiple of the temporal discretization step size \( \delta t \). If the end state of such a candidate is too far away from a lattice point, the candidate is dropped. For all motion primitive candidates that end at a state which belongs to the same point in the lattice, only the motion primitive which minimizes the quantization error is preserved. Finally, all motion primitives which can be reconstructed by concatenating shorter motion primitives are dropped. We call the resulting set of motion primitives, that all share the same start state, which only depends on the start heading \( \theta_0 \) and the start velocity \( v_0 \), a bunch denoted by \( B^0(\theta_0, v_0) \).

As the motion due to (1) is only “translationally invariant” with regard to the position \( x \) and \( y \) and the time \( t \), this procedure of sampling motion primitives has to be performed for all the start states resulting from all possible combinations of \( \theta \in \Theta^0 \) and \( v \in V^0 \). This results in a total of \( |\Theta^0| \times |V^0| \) separate motion primitive bunches. The combined set

\[
M^0 = \bigcup_{\theta \in \Theta^0} \bigcup_{v \in V^0} B^0(\theta, v) \tag{7}
\]

of all bunches completely defines the admissible motion through the associated state \( x \times t \) space lattice \( L^0 \) with its particular quantization \( q \).

### 2.3 Low-Resolution Lattice

As already stated in the introduction, we will employ multiple resolutions in our path planning algorithm. In this section, we will describe the construction of the low-resolution lattice \( L^1 \). We want the path planning algorithm to explicitly take into account dynamic obstacles during the planning. Thus, in order to correctly respect the future motion of the dynamic obstacle, the search space must contain a time dimension. To create the full-dimensional lattice

\[
L^1_0 = X^l \times Y^l \times \Theta^l \times V^l \times T^l, \tag{8}
\]

we perform the steps described in section 2.2, which generate the corresponding set of motion primitives

\[
M^0_1 = \bigcup_{\theta \in \Theta^l} \bigcup_{v \in V^l} B^0_1(\theta, v). \tag{9}
\]

Planning a complete trajectory in the full-dimensional state \( x \times t \) space would be computationally intractable for real-time applications. Therefore, we use the full-dimensional lattice only until a particular time \( t_0 \). For planning the parts of the path that exceed this time, we use the low-dimensional state lattice

\[
L^1_1 = X^l \times Y^l \times \Theta^l, \tag{10}
\]

which comprises a reduced number of dimensions. The corresponding low-dimensional motion primitive set

\[
M^1_1 = \bigcup_{\theta \in \Theta^l} B^1_1(\theta) \tag{11}
\]
is obtained by projecting \( M^0_1 \) onto \( L^1_1 \), which is essentially the same as stripping the \( v \) and \( t \) dimensions from each motion primitive. The index “1” denotes that this is the first projection of the full-dimensional search space.

Due to this projection step, it is very likely that two motion primitives of the same bunch \( B^1_1(\theta) \) will end at the same state. Whenever this is the case, we assess these motion primitives by the same cost function as the subsequent graph search uses, and keep only those motion primitive with the least cost, the other motion primitives are discarded. In addition, the “wait” action is removed from the set as it has no meaning any more. The proposed approach guarantees that the part of the path starting at \( t_0 \) is still kinematically feasible. This distinguishes our approach of dimensionality reduction from the one presented by Kushleyev and Likhachev (2009), which reduces to a simple grid search after having reached \( t_0 \).

The admissible transitions from \( L^1_1 \) to \( L^1_1 \) are defined as follows: All states \( s_0 = (x, y, \theta, v, t) \in L^1_0 \) with \( t \geq t_0 \) connect to those states of \( L^1_1 \) that are reachable by a motion primitive \( m \in B^1_1(\theta) \), i.e., the bunch starting at \( t_0 \) by the projection of \( s_0 \).

### 2.4 High-Resolution Lattice

By using the state \( x \times t \) lattice with fixed resolution, constructed according to the last section, an efficient planning of trajectories would already be possible. However, especially in the close vicinity of the robot, a very precise motion planning is desirable. On the other hand, in farther regions, the planning could be done more coarsely due to the increased perception uncertainty. Thus, we build upon the work of Likhachev and Ferguson (2009) and Rufli et al. (2009) and transfer the concept of a multi-resolution state lattice to our approach of a hybrid-dimensional state \( x \times t \) lattice. To the best of our knowledge, this is the first approach that combines these two concepts in an integrated consistent way.

As in the low-resolution case, in the following we will construct two high-resolution lattices with different dimensionality. First, we construct the full-dimensional high-resolution motion primitive set \( M^h_0 \) similar to the low-resolution set \( M^l_0 \) by randomly sampling a large number of motion primitives and subsequent motion primitive selection. Afterward, we merge this newly constructed motion primitive set \( M^h_0 \) with its low-resolution counterpart \( M^l_0 \) in order to obtain the final full-dimensional high-resolution motion primitive set

\[
M^h = M^h_0 \cup M^l_0. \tag{12}
\]

This procedure allows us to provide sub-optimality bounds for planning in the multi-resolution lattice: A path planned in the multi-resolution lattice will be at least as optimal as it would be when planning in the low-dimensional lattice only. Furthermore, we assume that the quantization of the high-resolution lattice is chosen such that the low-resolution lattice is a strict subset of the high-resolution lattice, i.e., the following holds.

\[
L^1_0 \subset L^h_0. \tag{13}
\]

This essentially means that the low-resolution discretization step size has to be a multiple of the high-resolution discretization step size, i.e.,
\[ \delta_x^h = \delta_x^l, \delta_y^h = \delta_y^l, \quad m \in \mathbb{N}, \quad (14) \]
\[ \delta_t^h = \delta_t^l, \quad n \in \mathbb{N}, \quad (15) \]
\[ n_0^h = k n_0^l, \quad k \in \mathbb{N}. \quad (16) \]

The set of discrete velocities in the high-resolution lattice is simply obtained by adding additional velocities to the low-resolution set of discrete velocities \( V^l \).

\[ V^h = V^l \cup \{v_1^h, v_2^h, \ldots \}. \quad (17) \]

Choosing the quantization of the high-resolution lattice in this way offers two advantages. On the one hand, by these means, the transitions between different resolutions during the graph search can be defined very concisely (see section 2.5); on the other hand, it has the positive effect that only a few new motion primitives are added to \( M_0^h \) during the merge (12) as many of them already belong to \( M_0^l \). This keeps the overall number of motion primitives in \( M_0^h \) low, thus resulting in a reduced branching factor in the subsequent graph search.

As in the low-resolution case, we construct the low-dimensional high-resolution motion primitive set \( M_0^h \) by projecting \( M_0^h \) onto the low-dimensional state lattice
\[ L_0^h = \mathbb{X}^h \times \mathbb{Y}^h \times \Theta^h_0. \quad (18) \]

The transitions between the high-dimensional state \( \times \) time lattice \( L_0^h \) and the low-dimensional state lattice \( L_1^l \) are defined similarly to the low-resolution transitions.

### 2.5 Transitions Between Different Resolutions

The transitions from low-resolution planning regions to high-resolution regions are trivial. As \( L_1^l \subset L_0^h, i \in \{h, l\} \) holds, each state of the low-resolution lattice \( L_1^l \) is also a member of the corresponding high-resolution lattice \( L_0^h \). Thus, one can easily switch from a state \( s_i \) in the low-resolution lattice \( L_1^l \) to the high-resolution lattice \( L_0^h \) by simply executing the motion primitives of the corresponding bunch \( B_i^h \) that starts at \( s_i \).

For arbitrary transitions from a state \( s_i^h \) of the high-resolution region to the low-resolution region, very specific motion primitive sets would be needed to make the successors of \( s_i^h \) coincide with points of the low-resolution lattice \( L_1^l \). However, this would require a large number of additional motion primitive sets. To keep the computational complexity low, we chose a rigorous approach: We only allow a transition from \( s_i^h \) to the low-resolution region of the search space if \( s_i^h \) is already an element of the low-resolution lattice \( L_1^l \) as well. Otherwise, if the graph search requests a transition from a state \( s_i^h \notin L_1^l \) to the low-resolution region, this branch of the search is discarded, i.e., in such a case, \( s_i^h \) has no successors at all.

### 3. SEARCHING THE LATTICE

Using the motion primitives and lattice transitions defined in the previous sections, an implicit search graph is constructed in order to convert the motion planning problem to a general shortest path graph search. One can choose from a variety of graph search algorithms to find the optimal trajectory/path through this hybrid-dimensional multi-resolution state \( \times \) time lattice. We decided to use the ARA* algorithm presented by Likhachev et al. (2003) which seeks to quickly find an initial, yet possibly suboptimal solution for the shortest path in a graph. Thereafter, if there is still some computation time left, the solution can be incrementally improved. For each improvement step, a sub-optimality bound can be guaranteed by the inflation factor \( \epsilon \) of the heuristic.

This shortest path computation is performed repeatedly in order to quickly react to dynamic obstacles and to be able to appropriately generate new paths for a changed environment. Due to this receding horizon scheme (with respect to \( t_0 \)) for the generation of open-loop control strategies, the proposed motion planning algorithm can be thought of some kind of high level model predictive control (MPC).

#### 3.1 Regions of High-Resolution Planning

The resolution for planning in a certain region of the search space can be chosen independently of the used dimensionality, i.e., each combination of full-/low-dimensional and high-/low-resolution planning is possible, characterized by its respective lattice \( L_1^i, i \in \{h, l\}, q \in \{h, l\} \). As already described in section 2.3, the dimensionality of the search space is only depending on the time threshold \( t_0 \) for each expanded node. For the choice of the search space resolution, we follow the approach of Rufli et al. (2009) and Likhachev and Ferguson (2009), who differentiate between task and environment characteristics.

The task-based criterion is based on the idea that a high-resolution planning in the vicinity of the robot as well as in the proximity of the goal is desirable to allow for a smooth and precise trajectory planning in these two particular regions. We modify this criterion to not use a fixed radius to describe the vicinity of the robot but instead decide on whether to use a high-resolution planning by taking the distance of the currently considered state from the robot’s start position along the planned path candidate into account.

For the environment-based criterion, Rufli et al. (2009) propose that a high-resolution planning should be done in narrow areas of the environment. We use a more general environment-based criterion: In order to plan safely among obstacles while still having global optimality in mind, we employ a high-resolution planning whenever a state is close to an obstacle. This includes the special case of high-resolution planning in narrow areas. To efficiently determine the high-resolution planning zones, we first compute the Euclidean distance transform of the underlying map using the algorithm proposed by Maurer et al. (2003). For each expanded node of the graph, the corresponding cell of the map is computed. If the distance of that cell to the next obstacle is smaller than a given threshold \( d^* \), the planning will be performed using the high-resolution lattice; otherwise, the low-resolution lattice will be used.

#### 3.2 Dynamic and Static Obstacles

For a safe trajectory planning, that is as optimal as possible, dynamic obstacles in the vicinity of the robot have to be explicitly considered during the planning. In our implementation, we use the approach presented in Peteret...
et al. (2013), that models the collision risk due to dynamic obstacles in a probabilistic way based on the predicted trajectory of the dynamic obstacles. The graph search considers dynamic obstacles only during the full-dimensional planning phase, i.e., until the time $t_0$. The time threshold $t_0$ can either be set to a fixed value (like 4s) or be inferred from the prediction of the dynamic obstacles (e.g., the time when the uncertainty of the obstacles’ position exceeds a certain threshold). Additionally, we calculate a collision risk emerging from static obstacles, which is based on the previously mentioned Euclidean distance transform of the map and decreases with increasing distance from a static obstacle.

3.3 Cost Function and Heuristic

For computing the cost of each path candidate during the graph search, we consider a combination of the traveled distance, the needed time, and the estimated collision risk. Of course, the weighting of the individual components has a huge impact on the planning result. Therefore, choosing the weighting parameters is essentially a matter of personal preference. The used heuristic is quite simple. It only takes into account the Euclidean distance to the goal as well as the minimal time needed to arrive at the goal.

$$h(x, y) = \| (x_{goal}, y_{goal}) - (x, y) \| \left( 1 + \frac{\lambda_t}{v_{max}} \right)$$

(19)

The ratio of these two components is controlled by the parameter $\lambda_t$, which corresponds to the duration penalty of the cost function. This basic heuristic function is sufficient to successfully demonstrate the proper and already quite fast operation of our proposed algorithm of a hybrid-dimensional multi-resolution state × time lattice in the context of this paper. However, the performance of our algorithm would certainly benefit from the employment of more sophisticated heuristics like the one proposed by Ziegler et al. (2008), which is based on the Voronoi graph of the free space.

4. RESULTS

We have implemented the presented algorithm in C++ and evaluated it on an Intel® Xeon® E3-1270 CPU using previously recorded map data. In this section, we provide two exemplary results for the planning of hybrid trajectories/paths using the proposed multi-resolution state × time lattice.

We decided for the following two quantizations of the multi-resolution lattice.

Table 1. Quantization of the state × time lattice

<table>
<thead>
<tr>
<th>$L^i_t$</th>
<th>$L^h_t$</th>
<th>unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta_x, \delta_y$</td>
<td>0.6</td>
<td>0.2</td>
</tr>
<tr>
<td>$n_\theta$</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>$V$</td>
<td>${-2, 0, 3}$</td>
<td>${-2, 0, 3}$</td>
</tr>
<tr>
<td>$\delta_t$</td>
<td>0.5</td>
<td>0.5</td>
</tr>
</tbody>
</table>

As can be seen, we used the same discrete velocities and time increments in both the low-resolution and the high-resolution case. This is due to the tradeoff between a fine search space resolution and a small size of the motion primitive sets, which reduces the branching factor.

The following table provides some statistical information on the four sets of motion primitives sampled according to the procedure described in section 2.

Table 2. Motion primitive set characteristics

| $M^i_1$ | 552 | 11.5 | 1.69 m |
| $M^i_2$ | 308 | 19.3 | 2.10 m |
| $M^h_1$ | 912 | 9.5 | 1.39 m |
| $M^h_2$ | 564 | 17.6 | 1.61 m |

4.1 Obstacle Avoidance Maneuver

In the first example (see Fig. 1), the robot faces an oncoming dynamic obstacle. In order to avoid a collision, it has to compute an appropriate avoidance maneuver. Fig. 1a shows the costs that are accumulated during the graph search in the lattice (denoted by $g$). Only the minimum costs for all possible orientations, velocities and points in time are shown for each location. One can clearly see the regions of low-resolution planning (in open areas) and high-resolution planning (in the vicinity of the robot and the goal, and near obstacles). The magenta part of the path corresponds to the dark-green tiles in Fig. 1b and represents the trajectory portion of the solution (i.e., it additionally contains a time and velocity component). The white part of the path corresponds to the light-green tiles and represents the mere path portion of the solution (i.e., without any time or velocity information). The transition between these two segments happens at time $t_0 = 4s$. 

Fig. 1. Planning of an obstacle avoidance maneuver. The upper plot shows the costs $g$ which are the costs accumulated during the graph search, more precisely, the plot depicts $\min_{\theta,v,t} g$.
4.2 Parking Maneuver

The second example (see Fig. 2) is a parking maneuver. The robot starts facing upwards and has to back into a small parking space at the opposite side. After only 11 ms, the ARA* algorithm finds an initial solution using a heuristic inflation factor $\epsilon = 2$, the second one improves the initial solution using $\epsilon = 1$. The upper plots show the costs $g$ which are the costs accumulated during the graph search, more precisely, the plots depict $\min_{\theta,v,t} g$.

5. CONCLUSION

In this paper, we presented a novel approach for efficiently planning high-fidelity robot motions using hybrid-dimensional multi-resolution state $\times$ time lattices. We have shown a way to consistently construct this lattice and define transitions between the various regions of the search space. Our algorithm is real-time capable even for complex maneuvers which is the prerequisite for a reliable avoidance of dynamic obstacles. Our main objective for future research is the development of a more sophisticated heuristic to further reduce the number of nodes expanded during the graph search.

REFERENCES


