Conference Paper

On the Application of the D* Search Algorithm to Time-Based Planning on Lattice Graphs

Author(s):
Rufli, Martin; Siegwart, Roland

Publication Date:
2009

Permanent Link:
https://doi.org/10.3929/ethz-a-010035816

Rights / License:
In Copyright - Non-Commercial Use Permitted
On the Application of the D* Search Algorithm to Time-Based Planning on Lattice Graphs

Martin Rufli
Autonomous Systems Lab, ETH Zurich, Tannenstrasse 3, CH-8092 Zurich, Switzerland
martin.rufli@mavt.ethz.ch

Roland Siegwart
Autonomous Systems Lab, ETH Zurich, Tannenstrasse 3, CH-8092 Zurich, Switzerland
rsiegwart@ethz.ch

Abstract—In this paper we present a multi-resolution state lattice, which operates in four dimensions, namely 2D position, heading, and velocity. The generation of such a lattice is described, resulting in an efficient (in terms of branching factor) and feasible (i.e. directly executable) set of edges, which can be searched on using any standard graph based planner.

Furthermore, we introduce a novel heuristic, the time-viable heuristic with horizon \(T_m\), which exploits the limited (but nonetheless extremely large) number of feasible motion combinations in a state lattice of bounded time and stores them in a look-up table. This heuristic then enables recently developed incremental planning algorithms, which typically start node expansion at the goal state (such as the various D* variants [1]) to be employed in time-based search, where the time of arrival is generally unknown a priori. We show, that by employing this technique, on average a comparable number of expanded states are to be expected for a given initial planning problem as when using forward searching algorithms (such as A* and variants [2, 3]), thus speeding up re-planning by up to two orders of magnitude as reported in [4].

Index Terms—Non-holonomic, time-based, motion planning, state lattice, time-viable heuristic

I. INTRODUCTION

State lattices (applied to motion planning) have recently seen much attention in scenarios, where a preferable motion cannot be easily inferred from the environment (such as in planetary exploration [5, 6], off-road navigation, or in large-scale parking lots [7, 8, 9]). They are typically constructed by pruning an extremely large set of trajectory segments (generated with a suitably complex vehicle model via input or state-space sampling), down to a manageable subset. The underlying assumption is that through a sensible pruning policy, every single original trajectory segment is reconstructable by pruned trajectories in the state lattice, which operates in four dimensions, namely 2D position, heading, steering, and feasible (i.e. directly executable) set of edges, which can be searched on using any standard graph based planner.

Most literature on planning algorithms incorporating a state lattice is restricted to static, or quasi-static environments, where the lattice typically operates in a three-dimensional state-space (2D position and heading). Dynamic obstacles are typically treated in the same way as static obstacles, except for an inflated outline. Dynamic obstacle avoidance is then guaranteed via frequent re-planning (see i.e. [9]). These approaches work well in relatively slow, or open environments, but face issues in scenarios where some participants have a much higher maximal velocity, or where dynamic obstacles may block entire routes to the goal.

In general, more information than just the current pose of a dynamic obstacle can be extracted, however. Depending on the environment, predictions of variable quality can be deduced from an object’s present and past motion. Approaches, which model the physics of the obstacle (see i.e. [12, 13]) typically achieve very precise short term predictions. On the other hand, approaches which model the intention of an agent fare better in the long term [14, 15, 16]. This additional information can be implemented into the planning stage, but requires the augmentation of the lattice with a velocity dimension (as was i.e. demonstrated in outdoor scenarios [9] and in indoor environments [11] to produce time-optimal as opposed to path length optimal solutions) and the addition of time to the state space (see [17]).

Full-range planning in nD (2D position, heading, steering angle, velocities, time, ...) is computationally intensive, however, and neither justified due to the uncertainty in motion prediction growing unbounded over time. Kushleyev et al. [17] recently presented a solution to this problem by introducing the time bounded lattice, a multi-resolution lattice, where all but two dimensions (2D position) are dropped as soon as motion prediction fidelity decreases below a certain threshold. In this paper we extend on this idea and enable the application of incremental re-planning algorithms to time-bounded lattices.

The remainder of this paper is organized as follows: section II reviews heuristic search applied to time-based planning on lattice graphs with a special emphasis on the introduction of our new time-viable heuristic. In Section III we demonstrate the effectiveness of our approach both by theoretical considerations and in simulated increasingly populated (and thus complex) scenarios. Finally, in Section IV we draw conclusions and sketch future work.
II. APPLYING D* IN COMBINATION WITH MOTION PREDICTIONS

The efficiency by which D* repairs previously generated solutions (in many cases one to two orders of magnitude faster than re-planning from scratch [4]) renders it highly desirable for many deterministic graph search applications; especially so, as ever increasing computing power allows for higher-dimensional (and thus more complex) searches which better cope with various system and environmental constraints. By adding time to the state-space, the resulting motion is time-parametrized, and thus renders it impossible to initialize the goal state with the correct time of arrival, as this property is only known after computing the solution. A naive implementation would therefore estimate a time of arrival, plan, update the estimate, and continue until the estimate is confirmed. Such an implementation looses the speed boost over A* however. In this section, we present a solution to this problem in the form of a novel heuristic, the time-viable heuristic. We also describe the various components of our planner, with a special emphasis on adaptations due to the addition of motion predictions.

A. Graph

In [11], we developed a 4D multi-resolution lattice, where the resolution is adapted based on environmental (i.e. narrowness) and task characteristics (i.e. distance from the robot). It is generated by employing Catmul-Rom parametrized cubic splines and therefore allows for the control of end position, start and end heading, and, to some extent, maximal curvature along each segment. In the future we would like to incorporate quintic splines in order to specify curvature at edge boundaries and add rotational velocity to the state space. The lattice operates on a 0.1 m (high-res.) to 0.2 m (low-res.) grid, at 16 directions and with 6 velocities (0.0, 0.15, 0.5, 1.0 m/s, two rotational velocities). It has an average outdegree of close to 40. All successor edges of a single heading-velocity configuration are depicted in Fig. 1.

B. Heuristics

Heuristic based planner rely on a heuristic to guide them towards the goal state of the search. Well informed heuristics reduce planning time, as fewer states need to be expanded. The development of accurate heuristics is generally harder for higher-dimensional search spaces. We employ three complimentary heuristics, the combination of which is provably also a heuristic and generally very well informed.

1) 2D Heuristic: For a robot with nonholonomic constraints, a simple 2D Dijkstra search [18] on the lattice’s underlying 2D grid yields accurate heuristic values far away from the robot position and in cases where environmental constraints severely limit robot mobility. As it relies on the environment, it cannot be precomputed offline. Depending on the environment size, such a computation can take several seconds, but only needs to be performed once per fixed goal state. For D* searches, this goal state (namely the current robot pose) changes continuously as the vehicle moves, however. We thus limit computation to a tunnel around a 2D A* search.
2) Freespace Heuristic: The freespace heuristic returns the actual constrained cost (i.e., time of traversal) of moving between two states in absence of any obstacles, and is generated by performing a Dijkstra search on the state lattice to a finite time horizon. It is thus often highly accurate in the robot vicinity, but becomes less useful further away where obstacles are likely to invalidate it. We precompute it offline and store it in a look-up table. For memory space considerations, the notion of a trim ratio was introduced in [19], which specifies the cost ratio between the 2D heuristic and the freespace heuristic. This ratio should be selected as close to 1.0 as memory allows.

3) The Time-Viable Heuristic: The concept of the state lattice postulates that through combination of a (small) finite set of motion primitives, essentially the whole space of feasible motions is described. This implies that the robot can only reach a finite set of states via a limited number of transitions, in bounded time. The time-viable heuristic stores this finite set of states and their predecessor edges in a look-up table. Let us denote the branching factor (a state’s average number of successor edges) with \( BF \), and the execution time of the shortest edge (0.1 m) at highest velocity (1.0 m/s) with \( t_{\text{shortest}} \). The combinatorial number of non-identical trajectories is then computed to be of complexity \( BF^{t_{\text{max}}/t_{\text{shortest}}} \). It thus becomes intractable for large time horizons \( t_{\text{max}} \). The limited fidelity of motion predictions bounds this horizon, however, and allows thus for the implementation of such a heuristic in practice. In particular, we precompute offline a 5D Dijkstra search (2D position, heading, velocity, time) on our 4D lattice for every initial state (resulting in queries for heading-velocity combinations only, due to complete invariance in 2D translation, and partial invariance due to rotation). Edges surpassing a fixed time horizon \( t_{\text{max}} \), linked to the fidelity of motion predictions) are discarded. Expansion continues until there are no valid edges left on the queue. As a post-processing step, we perform an optimization to reduce memory usage: due to the lattice’s discretization in position (0.1 m) and the robot’s maximal velocity (1.0 m/s), a minimal uncertainty in time of 0.1 s results. We exploit this fact and a posteriori discretize the time dimension in 0.1 s intervals in order to merge states with identical rounded arrival times and identical successor edges together. We then arrive at a reduced size 6D look-up table of time-edge distinct states (see Fig. 2 for an illustration with \( t_{\text{max}} = 4.0 \) s). This heuristic look-up table enables the planning algorithm to determine during search, whether a given expanded state is possibly within range of \( t_{\text{max}} \) of the search goal state, and if so, with which time instances it needs to be initialized so that it can reach the goal state at \( t = 0.0 \) s.

C. Motion Predictions of Dynamic Obstacles

Predictions concerning the future motion of dynamic obstacles are challenging, as errors in the applied model unfavorably propagate through time. Nonetheless, in the future we would like to add prediction functionality to our model-based pedestrian detection and tracking module [20]. For the remainder of this paper we will assume perfect knowledge of the future motion of dynamic objects (up to \( t_{\text{max}} \)), however. This allows for a performance analysis in exclusion of artifacts due to motion prediction.
D. Selected D* Function Annotations

The D* planning algorithm does not rely on any assumptions concerning the underlying graph but is commonly employed on a 2D grid. In this section we therefore describe adaptations to key functions necessary when planning with motion predictions on a state lattice (see Table II).

1) getTraversalCost(state pred, state succ): In our implementation, the total cost of motion between two states is comprised of a combination of *time of traversal* and *risk of collision* with dynamic obstacles (defined as impact likelihood of occupation). Online, risk is stored in a three-dimensional cost map (2D position, time), and updated by combining motion prediction with object labeling.

2) getHeuristicCost(state current): As described in Section II-B, we employ three complimentary heuristics. The heuristic estimate from a state \( \text{current} \) to the goal is then computed as the maximal value of the three individual heuristics.

3) getPred(state current): This function returns a list of predecessors of the current state, where predecessors are defined in a graph of directed edges between start and goal location. The introduction of bounded time into the state-space results in two classes of states: these with no time assigned (which signifies that they are not yet within \( t_{\text{max}} \) of the start configuration), and these associated with a time \( \varepsilon \in [0, t_{\text{max}}] \). Predecessor expansion is thus divided into three cases:
   - Case 1: the current state has no time associated with it, and has no predecessors in the range of the time-viable heuristic. In this case, the number of predecessors is equal to the branching factor of the current state. Predecessors are assigned no time.
   - Case 2: the current state has no time associated with it, but some predecessors are in range of the time-viable heuristic, and thus of \( t_{\text{max}} \). This is the most expensive case, as every predecessor in range of the time-viable heuristic is expanded with every stored arrival time at that state. The max. number of arrival times for different configurations is shown in Fig. 2. Case 3: the current state has a time associated with it. This is the cheapest case, as then at most as many edges as the branching factor of the current state are returned.

4) getSucc(state current): This function returns a list of successors of the current state. Successors are assigned an arrival time if and only if the current state also has one assigned and it is lower than \( t_{\text{max}} \).
In the future, we will attempt to approach this issue through parallelization. Second, we will focus on designing more informed, environment aware heuristics in both static and dynamic constrained scenarios.

We have shown in scenarios of increasing complexity, that for our D* implementation the average number of expanded states is comparable to a forward searching A* query, despite the introduction of the time-viable heuristic. On the other hand, the number of states placed on the priority queue (resulting in priority queue percolates and cost checks, both of them costly operations) grows for our implementation substantially faster with environment complexity.

IV. CONCLUSIONS AND FUTURE WORK

In this paper we introduced a novel heuristic, the *time-viable heuristic*. It enables deterministic incremental planning algorithms to be employed in conjunction with motion predictions of dynamic obstacles which are thought to be necessary for planning better solutions in cluttered and dynamic scenes.

We have shown in scenarios of increasing complexity, that for our D* implementation the average number of expanded states is comparable to a forward searching A* query, despite the introduction of the time-viable heuristic. On the other hand, the number of states placed on the priority queue (resulting in

<table>
<thead>
<tr>
<th>$t_{\text{max}}$ [s]</th>
<th>3.0</th>
<th>3.5</th>
<th>4.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>5.2</td>
<td>3.5</td>
<td>16.7</td>
</tr>
<tr>
<td>min</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>max</td>
<td>18</td>
<td>37</td>
<td>62</td>
</tr>
</tbody>
</table>

**TABLE I**
COMPLEXITY ANALYSIS: AVG. NO. OF ADDITIONAL STATES PLACED ON QUEUE PER EXPANDED NODE UNDER FREE-SPACE ASSUMPTION

robot is required to plan several motions from a random start configuration on the left border to a random goal state on the right border. Table IV shows, that for various prediction horizons and scene complexities, our D* implementation expands a similar number of states to reach the solution as A*.

For harder problems, more states are placed on the heap, corresponding to many crossings of the time-viable heuristic’s border. These heap placements are costly, as they go along with cost checks and priority queue percolates. In the future, we will attempt to approach this issue through parallelization.

2) **Scenarios II & III**: Scenarios II and especially III illustrate the need for accurate heuristics when employing high-dimensional heuristic search. Scene II is located in a complex narrow environment, where another (autonomous) vehicle is moving towards the top at a constant velocity of 0.5 m/s (see Fig. 3), and thus blocks the direct path to the goal (green dot). The freespace heuristic is inaccurate due to environmental constraints (walls). The 2D heuristic neglects dynamic obstacles and thus dramatically underestimates the cost to reach the goal. Nonetheless our planner is able to find a solution in fewer than 100 expansion steps.

Scene III is located in the same environment as scene II, except that a dynamic object now moves towards to bottom of the scene at variable speed. The resulting motion requires our vehicle to perform an extra loop to let the obstacle pass, and only then proceed to the goal (see Fig. 3). This extra loop results in many expanded states close to the start position, as the planner believes it is close to the solution, just to discover that at these arrival times, the start state is expected to be blocked by the dynamic object. With our current heuristics, computation of a solution requires > 10000 expansions, and is thus far from real-time. It should be noted that while D* completely fails in such a hard environment, a forward searching algorithm could potentially start executing partial motions, combined with frequent re-planning.

### CONCLUSIONS AND FUTURE WORK

In this paper we introduced a novel heuristic, the *time-viable heuristic*. It enables deterministic incremental planning algorithms to be employed in conjunction with motion predictions of dynamic obstacles which are thought to be necessary for planning better solutions in cluttered and dynamic scenes.

We have shown in scenarios of increasing complexity, that for our D* implementation the average number of expanded states is comparable to a forward searching A* query, despite the introduction of the time-viable heuristic. On the other hand, the number of states placed on the priority queue (resulting in priority queue percolates and cost checks, both of them costly operations) grows for our implementation substantially faster with environment complexity.

Future work will thus investigate two directions: first, we will attempt to further increase planning and re-planning speed through parallelization. Second, we will focus on designing more informed, environment aware heuristics in both static and dynamic constrained scenarios.

### ACKNOWLEDGMENT

This work has been partially supported by the Swiss National Science Foundation NCCR MC3 and by the European Project EUROPA under contract number FP7-231888.
R. Rosales and S. Sclaroff. Improved tracking of multiple humans with
M. Rufli, D. Ferguson, and R. Siegwart. Smooth path planning in
M. Pivtoraiko and A. Kelly. Generating near minimal spanning control
K. Han and M. Veloso. Physical model based multi-objects tracking
D. Ferguson, T. Howard, and M. Likhachev. Motion Planning in
D. Ferguson, T. Howard, and M. Likhachev. Motion Planning in Urban
M. Likhachev and D. Ferguson. Planning long dynamically-feasible
R. A. Knepper M. Pivtoraiko and A. Kelly. Differentially constrained
REFERENCES

<table>
<thead>
<tr>
<th>Scene</th>
<th>L00</th>
<th>L02</th>
<th>L04</th>
<th>L06</th>
<th>L08</th>
<th>L10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planner</td>
<td>A*</td>
<td>D*</td>
<td>A*</td>
<td>D*</td>
<td>A*</td>
<td>D*</td>
</tr>
<tr>
<td>Motion Prediction Time Bound: 4.0 s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>expansions</td>
<td>mean</td>
<td>11</td>
<td>17</td>
<td>48</td>
<td>50</td>
<td>61</td>
</tr>
<tr>
<td>min</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>max</td>
<td>14</td>
<td>21</td>
<td>290</td>
<td>237</td>
<td>290</td>
<td>361</td>
</tr>
<tr>
<td>states on heap</td>
<td>mean</td>
<td>248</td>
<td>145</td>
<td>457</td>
<td>426</td>
<td>601</td>
</tr>
<tr>
<td>min</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>max</td>
<td>14</td>
<td>16</td>
<td>270</td>
<td>134</td>
<td>270</td>
<td>134</td>
</tr>
</tbody>
</table>

| Motion Prediction Time Bound: 3.5 s | | | | | | |
| expansions | mean | 11 | 11 | 33 | 24 | 44 | 26 | 78 | 39 | 85 | 51 | 95 | 48 |
| min | 9 | 8 | 9 | 8 | 9 | 8 | 9 | 8 | 9 | 8 | 9 | 8 |
| max | 14 | 16 | 270 | 134 | 270 | 134 | 270 | 134 | 300 | 131 | 300 | 131 |
| states on heap | mean | 248 | 207 | 416 | 267 | 579 | 284 | 917 | 486 | 976 | 512 | 1058 | 506 |
| min | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 |
| max | 300 | 131 | 300 | 131 | 300 | 131 | 300 | 131 | 300 | 131 | 300 | 131 |

| Motion Prediction Time Bound: 3.0 s | | | | | | |
| expansions | mean | 11 | 11 | 30 | 29 | 37 | 29 | 44 | 33 | 50 | 47 | 50 | 47 |
| min | 9 | 8 | 9 | 8 | 9 | 8 | 9 | 8 | 9 | 8 | 9 | 8 |
| max | 14 | 16 | 270 | 261 | 270 | 261 | 270 | 261 | 247 | 233 | 247 | 233 |
| states on heap | mean | 248 | 126 | 405 | 227 | 564 | 243 | 593 | 292 | 628 | 378 | 630 | 381 |
| min | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 |
| max | 14 | 16 | 270 | 261 | 270 | 261 | 270 | 261 | 247 | 233 | 247 | 233 |

TABLE III
SCENARIO I: PERFORMANCE OVERVIEW