Using State Dominance for Path Planning in Dynamic Environments with Moving Obstacles

Juan P. Gonzalez, Andrew Dornbush, and Maxim Likhachev

***Abstract*— Path planning in dynamic environments with moving obstacles is computationally complex since it requires modeling time as an additional dimension. While in other domains there are state dominance relationships that can significantly reduce the complexity of the search, in dynamic environments such relationships do not exist. This paper presents a novel state dominance relationship tailored specifically for dynamic environments, and presents a planner that uses that property to plan paths over ten times faster than without using state dominance.**

I. Introduction

In many path planning domains it is possible to identify states that cannot possibly contribute to the optimal solution and can therefore be pruned from the search without affecting the optimality of the solution. One such problem is planning a path for a battery powered mobile robot where we want to optimize a path that considers cost and battery level. Because the available battery power is a finite resource, we can say that a state s_i is always better than another state s_j at the same position, if it has more battery power available. We say that state s_i *dominates* state s_j.

This property, called *state dominance* [1] [2], can produce significant reductions in the size of the search space, which corresponds to significant reductions in space and time complexity.

Domains with dynamic obstacles require modeling time as an additional dimension, and are usually considered not to have state dominance. Although time would seem to induce a natural dominance relationship in which states with smaller time would dominate states with higher time, in the presence of dynamic obstacles this is no longer true. Sometimes it is necessary to wait in place for a dynamic obstacle to pass in order to arrive at the destination faster.

Recent results in path planning with *safe intervals*[3], prove that *when the cost function to be optimized is time*, it is possible to substitute intervals for time in the state space without affecting the optimality of the result. A safe interval is a contiguous period of time for a given spatial configuration during which there are no collisions, and it is in collision one timestep prior and one timestep after the period. Since there are usually fewer safe intervals than time discretizations, the resulting algorithm is much faster than explicitly modeling time as a dimension.

However, having time as the cost function to be optimized requires a binary environment where cells are either free or obstacles. This is a very limiting requirement that does not apply to outdoor environments and other types of environments where the cost function to be optimized is better described by a greater range of cost values.

In this paper we extend the results from planning with safe intervals to derive a state dominance relationship for dynamic environments that can be applied to continuous cost domains. We also present a planner called GSIPP that uses this dominance relationship to obtain up to ten times faster planning times in dynamic environments than without using state dominance.

To the best of our knowledge this is the first time a state dominance relationship has been derived and used for path planning in dynamic environments.

II. Related Work

A. State Dominance

In path planning domains, state dominance has been used to achieve significant improvements in memory requirements and planning speed. The ISE framework [4] uses state dominance to reduce the dimensionality of a graph search problem by removing the cost function from the set of state variables under certain conditions. DD* Lite [5] on the other hand does not remove any dimensions but prunes dominated states as it discovers them during the search. Similarly, Gonzalez and Stentz [6][7] also use state dominance to prune states within the context of planning with uncertainty in position.

B. Planning in Dynamic Environments

Domains with dynamic obstacles require modeling time as an additional dimension, which significantly increases computational complexity.

Some approaches model dynamic obstacles as static obstacles in order to avoid adding time as a dimension [8]. This is efficient but incomplete, as it can only model dynamic obstacles as a snapshot in time, or as a larger

Manuscript received September 16, 2011. This work was supported by the U.S. Army Research Laboratory under the Robotics Collaborative Technology Alliance program, Cooperative Agreement W911NF-10-2-0016. The views and conclusions contained in this document do not represent the official policies or endorsements of the U.S. Government.

J.P. Gonzalez is with the Autonomous Perception Research Lab, General Dynamics Robotic Systems. Pittsburgh, PA 15221 USA (Phone 412-473-2164; e-mail: jpgonzal@gdrs.com).

M. Likhachev and A. Dornbush are with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: maxim@cs.cmu.edu).

dynamic obstacle that includes a series of future positions.

Local approaches tend to be more accurate [9], as they model time correctly. However, they are prone to local minima and generally cannot solve problems that require waiting.

Other approaches limit the environments to be represented as binary cost values: either free space or obstacles. Binary environments allow using roadmaps and sampling based planners to speed up planning [10][11]. Safe Interval Path Planning (SIPP)[3] also exploits the properties of binary environments to compress the dimensionality of time-based path planning when the objective function to be optimized is time. This approach produces very quickly paths of higher quality than most roadmap-based or sampling-based approaches. Planning in binary environments, however, has significant limitations. In many domains it is important to consider a range of costs that represents traversal cost, risk, fuel consumption, etc. Binary environments are unable to model such cost functions.

When planning with continuous cost functions, existing approaches have to plan in the full state space. State dominance would provide much needed reduction in the size of the search space, but dynamic environments are usually considered not to have state dominance relationships. Although time would seem to induce a natural dominance relationship in which states with smaller time would dominate states with higher time, in the presence of dynamic obstacles this is no longer true. Sometimes it is necessary to wait in place for a dynamic obstacle to pass in order to arrive at the destination faster.

III. Generalized Safe Interval Path Planning (GSIPP)

A. Safe Interval Path Planning (SIPP)

Recently, Phillips and Likhachev introduced SIPP[3], a planner that uses *safe intervals* to compress the dimensionality of time-based path planning when the objective function to be optimized is time. They define a safe interval as a contiguous period of time for a configuration, during which there are no collisions and it is in collision one timestep prior and one timestep after the period. The obvious exception to this is that the last safe interval for a configuration may go until infinity, if a dynamic obstacle never again is predicted to pass through this configuration.

SIPP assumes that the cost function to be optimized is the time to reach the goal. Under this assumption, they prove that arriving at a state at the earliest possible time within a safe interval maintains the optimality and completeness of the planner. SIPP, however cannot maintain these guarantees if the cost function is not time, as is usually the case in non-binary environments where different cost values are used to represent traversal cost, fuel spent, or other non-binary functions.

SIPP's key insight is the realization that arriving at a state at the earliest possible time within a safe interval maintains the optimality and completeness of the planner. SIPP uses this property of the search space to prune states that are dominated and to remove intermediate wait actions. In this paper we introduce *Generalized Safe Interval Path Planning* (GSIPP), which extends SIPP through the following changes in order to guarantee optimal planning in non-binary environments.

B. State Dominance for Dynamic Environments Using Safe Intervals

Formally, given two states in a search algorithm, s_i and s_j, a dominance relation $\trianglerighteq$ is defined as a binary relation such that $s_i \trianglerighteq s_j$, that is, s_i dominates s_j, implies that s_j cannot be part of a solution better than the best solution obtainable from s_i [2]. Dominated states may be deleted without expansion in the search, thus eliminating entire branches of the search tree.

Algorithms based on dynamic programming such as A* apply state dominance to states with the same state-space coordinates, by only keeping the best path from the start to any given state. If a state s_i can be reached via two paths of total cost from the start $g(s_i)$ and $g'(s_i)$ and $g(s_i) \le g'(s_i)$, then the path with cost $g(s_i)$ dominates the one with cost $g'(s_i)$ and the latter path is eliminated from the search. Some domains, however, have stronger types of state dominance where two states with different state-space coordinates can have a dominance relationship.

We would like to derive a dominance relationship between two states $s_i = (\mathbf{X}, t_i) = (x_1, x_2, ...x_n, t_i)$ and $s_j = (\mathbf{X}, t_j) = (x_1, x_2, ...x_n, t_j)$ that share the same spatial location $\mathbf{X} = (x_1, x_2, ...x_n)$ but have different time coordinates. SIPP implicitly uses the relationship

$$s_i \trianglerighteq s_j \leftrightarrow \ t_i < t_j \tag{1}$$

which only holds when the objective function to be optimized is time ($C(s_i, s_j) = t_j - t_i$, $g(s) = t$) and there are wait actions available at any state. We define a more general, non-binary transition cost

$$C(s_i, s_j) = C_s(\mathbf{X}_i, \mathbf{X}_j) + C_t(t_j - t_i) \tag{2}$$

where C_s is the transition cost due to the static part of the environment, and C_t is the cost of associated with the time extent of the transition. We extend the dominance relationship from SIPP to include this general cost function as follows.

$$s_i \trianglerighteq s_j \leftrightarrow \ g(s_i) + C_t(t_j - t_i) \le g(s_j) \ \wedge \ t_i < t_j \tag{3}$$

where $g(s)$ is the cost of the lowest cost path from the start to state s and $C_t(t_j - t_i)$ is the cost to wait from t_i to t_j at state s_i (Figure 1). This relationship implies that s_j is reachable from s_i through a wait of $t_j - t_i$ and that the total cost of reaching s_j through s_i ($g(s_i) + C_t(t_j - t_i)$) is less than $g(s_j)$.

In dynamic environments with moving obstacles we cannot guarantee that s_j is reachable from s_i as it is possible that waiting in place can lead to a collision.

The safe intervals introduced in SIPP decompose the space into collision free regions, such that if a safe interval *k* at configuration $x_1, x_2, ..., x_n$ spans from time $t_{k_{\min}}$ to time $t_{k_{\max}}$, then all times from $t_{k_{\min}}$ to $t_{k_{\max}}$ are guaranteed to be collision free. Taking advantage of this property, it is possible to guarantee reachability as long as this is limited to states *within the same safe interval*.

Additionally, we need to show that no optimal path from start to goal would require s_j (Figure 1). The total cost of reaching state s_l through state s_j is

$$g'(s_l) = g(s_j) + C(s_j, s_l). \tag{4}$$

If we remove s_j from the search, s_l can only be reached through s_k. The cost of reaching s_l through s_k is

$$\begin{aligned} g(s_l) &= g(s_k) + C(s_k, s_l) \\ g(s_l) &= g(s_i) + C(s_i, s_k) + C(s_k, s_l) \end{aligned} \tag{5}$$

We can guarantee that no optimal path would require s_j if

$$\begin{aligned} g(s_l) &\le g'(s_l) \\ g(s_i) + C(s_i, s_k) + C(s_k, s_l) &\le g(s_j) + C(s_j, s_l) \end{aligned} \tag{6}$$

for any s_i, s_j, s_k and s_l. Using the cost decomposition from (2), (6) becomes

$$\begin{aligned} g(s_i) + C_s(\mathbf{X}_i, \mathbf{X}_k) + C_t(t_l - t_k) &\le g(s_j) + C_s(\mathbf{X}_j, \mathbf{X}_l) \\ g(s_i) + C_t(t_j - t_i) &\le g(s_j) \end{aligned} \tag{7}$$

since $\mathbf{X}_i = \mathbf{X}_j$, $\mathbf{X}_k = \mathbf{X}_l$ and $(t_j - t_i) = (t_l - t_k)$, which shows that the dominance relationship from (3) is valid within a safe interval.

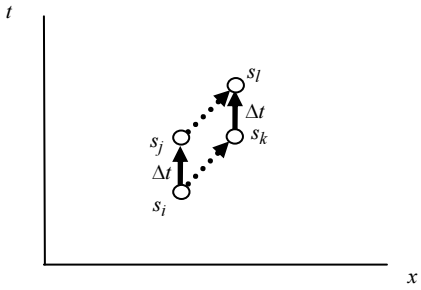

Figure 1. Graphic to illustrate the requirements for state dominance in dynamic environments.

C. Pruning Dominated States

SIPP prunes all dominated states from each configuration $(x_1, x_2, ...x_n)$ within each safe interval $k=[t_{k_{\min}}, t_{k_{\max}}]$ according to (1). This leaves only one time value *t* within each interval. Because of this, SIPP drops the time dimension from the state space, and instead it uses a state space defined by $s_i = (x_1, x_2, ...x_n, k)$.

For non-binary environments, we use the state dominance relationship from (3). However, this relationship now resembles a partial order, instead of a total order. While in binary worlds all states are comparable (either state s_i dominates s_j or state s_j dominates s_i), in non-binary worlds often they will not be comparable (neither state s_i dominates s_j nor state s_j dominates s_i). The result of this is that for a given configuration $(x_1, x_2, ...x_n)$ within a safe interval *k*, there can be multiples states, all with different *t*. For this reason, we cannot drop time as a dimension and we need to define states as $s_i = (x_1, x_2, ...x_n, t)$. We do not explicitly include the intervals as part of the state space.

D. Planning with Generalized Safe Intervals

We use regular A* search (Figure 2), except in the way in which successors are calculated (line 06), and in the state dominance function used (*Dominated*, Figure 3). The search starts with the start state, and GetSuccessors (Figure 4) operates as in regular A* search until there is more than one safe interval to consider. *M*(*s*) (Figure 4, line 02) returns the motions that can be performed from state *s.* These motions indicate how they change the spatial variables of a state and the time that it takes to execute them (lines 03 and 04).

When more than one safe interval is available *GetSuccessors* considers all possible safe intervals to transition through wait and move actions (lines 07-11). This means that for each safe interval *k* = [*startTime*, *endTime*], at each successor, we wait the minimal amount of time possible to transition to it and arrive safely. *start_t* is the earliest possible time to arrive at a successor state *s'* and *end_t* is the latest possible time to safely arrive at a successor state *s'*. *end_t* corresponds to waiting until the end of the safe interval of *s* and then transition to the successor state *s'*. Notice that only transitions from [*start_t, end_t*] that overlap with [*startTime*, *endTime*] are safe (line 08).

```
01  g(s_start) = 0 ; OPEN = ∅ ; CLOSED = ∅
02  insert s_start into OPEN with f(s_start) = h(s_start)
03  while s_goal is not expanded
04      remove s with the smallest f value from OPEN
05      insert s into CLOSED
06      successors = GetSuccessors(s)
07      for each s' in successors
08          if s' was not visited before
09              f(s') = g(s') = ∞
10          if g(s') > g(s) + c(s,s')
11              f(s') = g(s) + c(s,s')
12              h(s') = g(s') + h(s')
13              if ¬ Dominated(s')
14                  insert s' into OPEN with f(s')
```

Figure 2. Algorithm 1: A* with generalized safe intervals

Procedure: Dominated(s)

```
01  // s = (x_1, x_2, ..., x_n, t)
02  for each s' = (x'_1, x'_2, ..., x'_n, t') ∈ OPEN ∪ CLOSED such that
03          x'_1 = x_1 ∧ x'_2 = x_2 ... ∧ x'_n = x_n ∧ t' < t ∧
04          interval(s') = interval(s)
05      if g(s') + C_t(t - t') ≤ g(s)
06          return true
07  return false
```

Figure 3. Dominated

Procedure: GetSuccessors(s)

```
01  successors = ∅
02  for each m in M(s)
03      (x'_1, x'_2, ..., x'_n) = motion m applied to s
04      m_time = time to execute m
05      start_t = time(s) + m_time
06      end_t = endTime(interval(s)) + m_time
07      for each safe interval k in (x'_1, x'_2, ..., x'_n)
08          if (startTime(k) ≤ end_t ∧ endTime(k) ≥ start_t)
09              t' = max(start_t, endTime(k))
10              s' = (x'_1, x'_2, ..., x'_n, t')
11              insert s' into succesors
12  return successors;
```

Figure 4. GetSuccesors

The search then progresses over all the valid safe intervals. Within each safe interval the search operates as regular A* search with state dominance, but when different safe intervals become available, transitions to each one of those intervals are again added as possible successors.

E. Theoretical Analysis

As argued in section III B, a state s' that fails the test in line 13 (Figure 2) is dominated by some state s and therefore it can be shown that state s' cannot possibly lie on an optimal path from the start to the goal. The following theorem puts it formally.

Theorem 1. If Dominated(s') in line 13 (Figure 2) *returns true then pruning state s' is guaranteed to preserve at least one optimal path from* s_{start} *to* s_{goal}.

Theorem 2. When the generalized safe interval planner expands a state in the goal configuration, it has found an optimal collision-free path to the goal.

From Theorem 1, it follows that the states that are actually being expanded by the search include all the states necessary to find an optimal path. In optimal A*, when a state is expanded, its g-value is minimal. Therefore when our algorithm expands a state in the goal configuration, its g-value will be minimal. We also know that all states exist within safe intervals, which makes the entire path collision-free, from the start state to the goal state.

IV. EXAMPLE

The example in Figure 5 illustrates the operation of the algorithm and the effect of using state dominance. The robot is trying to move from left to right, while three dynamic obstacles will cross its path. Obstacle #1 (bottom) moves up, while obstacles #2 and #3 move down.

To solve this problem, a naïve planner in (x,y,t) requires a wait action besides the regular successors in x,y such that

$$(x', y', t') = (x, y, t + \Delta t). \quad (8)$$

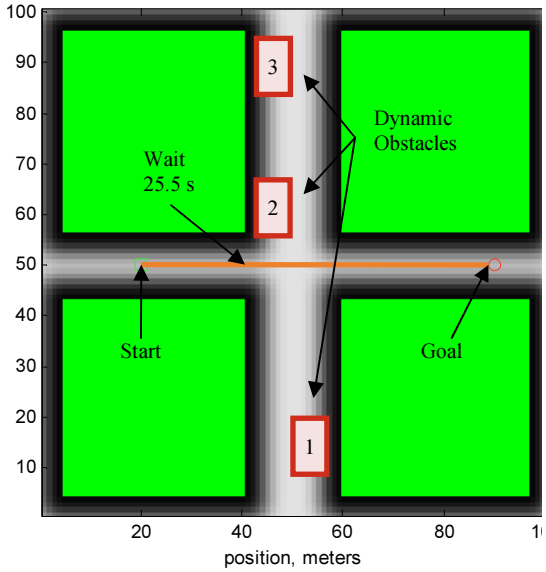

Figure 5. Sample environment showing dynamic obstacles and resulting path when using state dominance.

Figure 6 shows the states expanded for a slice of the state space at y=50. Notice the thick volume of states that are considered at each x coordinate. Also notice that in order to get past the intersection and avoid collisions, a wait is necessary. Since all waits of equal duration have the same cost, the planner chooses to wait at the start of the path.

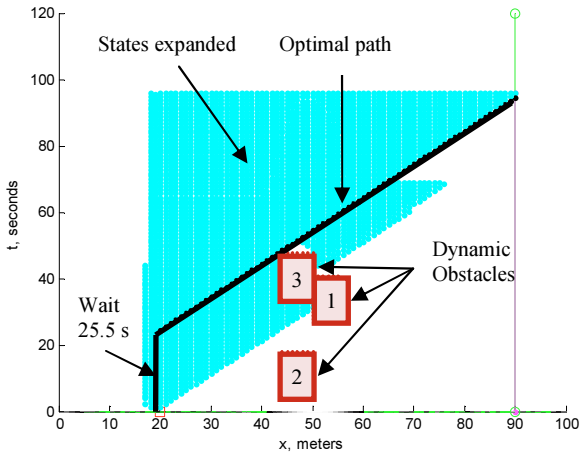

Figure 6. States expanded without state dominance (naïve planner in x,y,t)

A common approach to avoid the computational complexity of modeling time as an additional dimension is to model dynamic obstacles as larger static obstacles. Each obstacle is usually modeled as occupying not only its current location, but also all positions from t to $t + \Delta t$ seconds in the future. This is efficient but incomplete, as it cannot model wait states and other complex interactions. Figure 7 shows the resulting path obtained after modeling the dynamic obstacles from Figure 5 as larger obstacles, and *not* using time as a dimension.

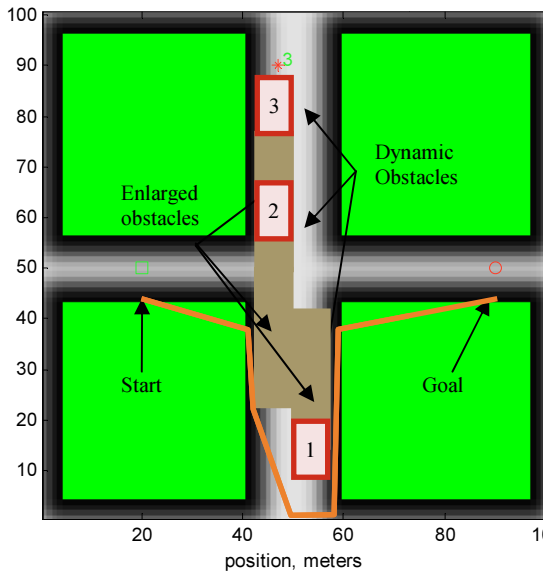

Figure 7. Solution found by modeling dynamic obstacles as static obstacles and not modeling time as an additional dimension.

GSIPP has a computational complexity much lower than that of a naïve *(x,y,t)* planner, yet preserves optimality and models time as a full dimension. Figure 8 shows the states expanded for the same slice of the state space from Figure 6 when using state dominance and safe intervals. Notice that when using safe intervals, the only wait actions that take place are at the boundary of a safe interval, therefore the wait needed to clear the intersection now happens near the intersection. The duration of the wait is the same as before, and so is the total cost of the path.

In general, only a thin slice is needed within each safe interval. However, this is not always the case. After passing the dynamic obstacles there are two thin slices of solutions moving towards the goal. The top one is the lowest cost solution that ultimately leads to the goal. The bottom one is another solution that has shorter time but higher cost.

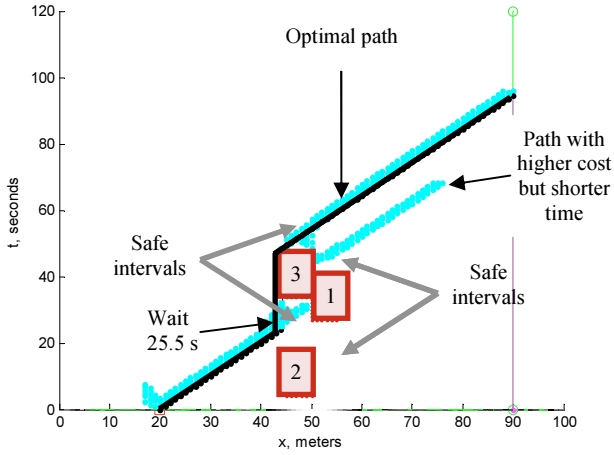

Figure 8. States expanded with state dominance and safe intervals (GSIPP).

V. EXPERIMENTAL RESULTS

In order to evaluate the experimental performance of GSIPP we performed two sets of simulations and one set of experiments in a segbot running ROS.

A. Performance with respect to different environment sizes

In order to evaluate performance with respect to the size of the environment we generated several fractal worlds with different random seeds and sizes varying from 100x100 to 500x500 cells. The density of movers was kept constant in such a way that the number of movers at 500x500 cells was 200. The start location was fixed near the lower left corner and the goal location was fixed near the top right corner. 50% of the dynamic obstacles were 1x1 cell big, and 50% were 5x5 cells big. Dynamic obstacles started at random locations and followed paths with random variations in heading, with the small movers changing heading every four time steps, following a uniform distribution from -45 to 45 degrees; and the large movers following a uniform distribution from -10 to 10 degrees. The total number of trials evaluated was 500. Figure 9 shows one of these environments.

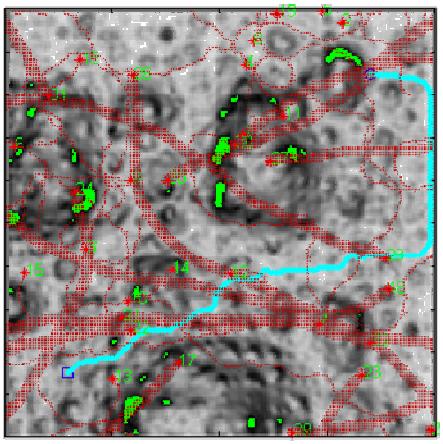

Figure 9.Sample environment with dynamic obstacles and their trajectories, as well as the resulting path.

We compared the performance of a naïve *(x,y,t)* planner with GSIPP running in *(x,y,t)*, with state dominance applied to states at the same *(x,y)* location. Figure 10 shows the results of these experiments. There is over an order of magnitude improvement in the number of expansions, independently of the size of the world. Processing time also shows over an order of magnitude improvement with a slight decrease as the size of the environment increases. This increase is caused by the overhead in the calculation of state dominance, which increases with the size of the environment.

B. Performance with respect to number of movers

In order to evaluate performance with number of movers, we again generated several fractal worlds of 500x500 cells,

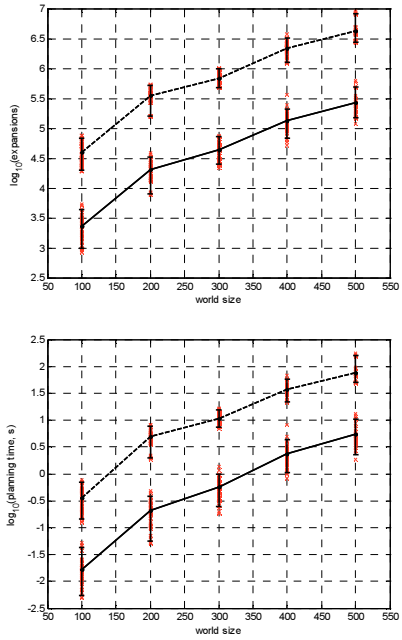

Figure 10. Expansions (top) and planning time (bottom) in logarithmic scale for the naïve (x,y,t) planner (dashed line) and GSIPP (solid line), for various world sizes in a fractal environment with constant dynamic obstacle density. Bars indicate 95% confidence intervals.

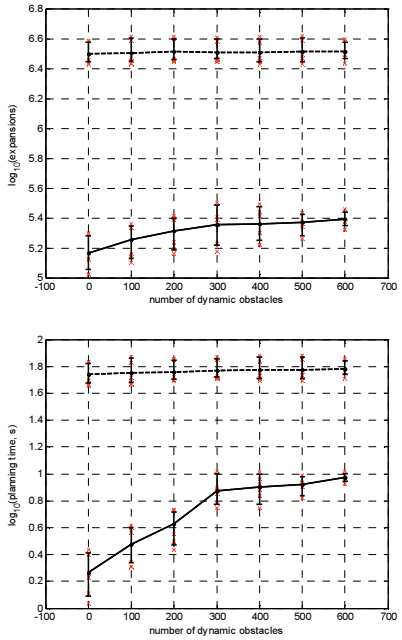

Figure 11. Expansions (top) and planning time (bottom) in logarithmic scale for the naïve (x,y,t) planner (dashed line) and GSIPP (solid line) for different number of dynamic obstacles in a 500x500 fractal environment.Bars indicate 95% confidence intervals.

with varying number of dynamic obstacles from 0 to 600. The distribution and motion of movers follow the same criteria as in the previous section. The total number of trials evaluated was 300.

We compared the performance of the same planners, a naïve (x,y,t) planner with GSIPP running in (x,y,t), with state dominance applied to states at the same (x,y) location. Figure 11 shows the results of these experiments. There is over an order of magnitude improvement in the number of expansions, with a slight decrease as the number of dynamic obstacles increases. The planning time shows an improvement of about one order of magnitude, but there is a greater difference between the improvement without movers and the improvement with 600 movers. Most of this difference is caused by the overhead in calculating state dominance.

C. *Tests on a Segbot*

We implemented GSIPP as a planner in the *sbpl_dynamic_env* stack in ROS. This stack contains a global planner that plugs in to ROS' navigation stack. The navigation stack provides the planner with the robot's pose and a map of the environment and expects a path in return.

We ran on a Segbot at the Search Based Planning lab at Carnegie Mellon University. This robot is a Segway base with a Hokuyo lidar, using monte-carlo localization for position estimation. We used the tracker node from SIPP, which uses the lidar to predict the trajectory of dynamic obstacles. The tracker clusters lidar points from each scan and then matches the clusters from this scan to clusters from the last iteration. It then uses a linear regression fit to predict the velocity vector of the dynamic obstacle, and this vector is used to predict the future position of the obstacle.

The planner implemented uses a lattice in (x,y,θ,t) , and we apply time-based state dominance to states with the same (x,y,θ) coordinates.

We use ROS' *costmap_2d* package to generate obstacle expansions and an exponentially decaying buffer around obstacles. We also penalize backward motions by applying a higher cost with respect to forward motion. Both of these options are possible because of the ability of GSIPP to optimize non-binary cost functions. Figure 12 shows the cost map for one of the experiments, in which the robot is moving from left to right in a narrow corridor and detects a dynamic obstacle coming towards it. GSIPP plans a path that turns to the left and waits for 5 seconds for the dynamic obstacle to pass. The planner was able to find a solution in less than 1 second on each cycle, with a maximum planning time of 0.9 seconds. Figure 13 shows a succession of snapshots from *rviz* and from the accompanying video for this paper.

VI. Conclusions and Future Work

We introduced a state dominance relationship that uses safe intervals and can be used for continuous-cost dynamic environments. We also introduced a planner that uses this dominance relationship to obtain about an order of magnitude fewer expansions and faster planning times in dynamic environments than without using state dominance.

To the best of our knowledge this is the first time a state dominance relationship has been derived and used for path

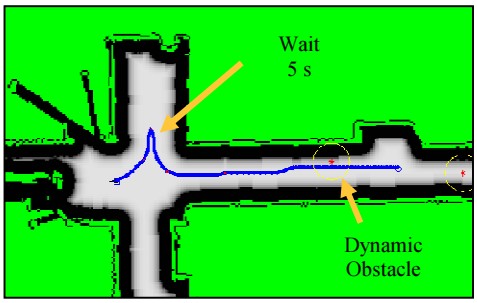

Figure 12. Cost function generated by costmap_2d and used by GSIPP to calculate optimal paths. The yellow circles are dynamic obstacles detected. Light gray areas are low cost and darker gray areas are higher cost. Green areas non-traversable (expanded obstacles). The blue (solid) line shows the path found by GSIPP

planning in dynamic environments.

In the near future we would like to convert the planner into an anytime planner for faster performance in time critical applications. We also would like to perform further experiments in different robotic platforms and in outdoor environments. We would also like to evaluate the results of using this form of state dominance to planning domains with higher dimensions.

ACKNOWLEDGMENT

We would like to thank Michael Phillips for making available the source code of his implementation of SIPP, and for his help in setting up our planner to run within ROS.

REFERENCES

[1] T. Fujino and H. Fujiwara, "An efficient test generation algorithm based on search state dominance," in *Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers., Twenty-Second International Symposium on*, jul 1992, pp. 246 –253.

[2] E. Horowitz and S. Sahni, "Fundamentals of computer algorithms," 1978.

[3] M. Phillips and M. Likhachev, "Sipp: Safe interval path planning for dynamic environments," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2011.

[4] P. Tompkins, A. Stentz, and D. Wettergreen, "Mission-level path planning and re-planning for rover exploration," *Robotics and Autonomous Systems, Intelligent Autonomous Systems*, vol. 54, no. 1, pp. 174 – 183, February 2006.

[5] G. A. Mills-Tettey, A. Stentz, and M. B. Dias, "Dd* lite: Efficient incremental search with state dominance," in *Twenty-First National Conference on Artificial Intelligence (AAAI-06)*, July 2006, pp. 1032–1038.

[6] J. P. Gonzalez and A. Stentz, "Using linear landmarks for path planning with uncertainty in outdoor environments," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, 2009, pp. 1203 –1210.

[7] ——, "Planning with uncertainty in position: An optimal and efficient planner," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS '05)*, August 2005.

[8] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *Int. J. Rob. Res.*, vol. 28, pp. 933–945, August 2009. [Online]. Available: http://portal.acm.org/-citation.cfm?id=1577179.1577184

[9] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *Robotics & Automation Magazine, IEEE*, vol. 4, no. 1, pp. 23–33, 1997.

[10] J. Van Den Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. IEEE, pp. 2366–2371.

[11] J. Van den Berg and M. Overmars, "Roadmap-based motion planning in dynamic environments," *Robotics, IEEE Transactions on*, vol. 21, no. 5, pp. 885–897, 2005.

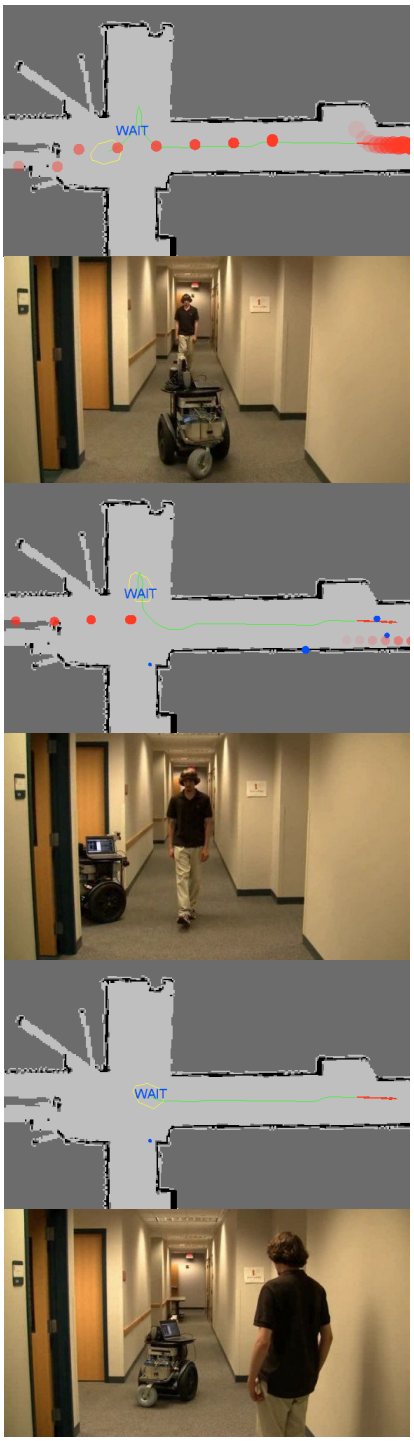

Figure 13.Succession of events of a GSIPP run in ROS. The robot detects a moving obstacle (red circles indicate present and projected path). GSIPP plans a path that avoids it by getting out of the way and waiting for the dynamic obstacle to pass. After the dynamic obstacle has passed, the robot continues its trajectory to the goal.