Improved Hierarchical Planner Performance Using Local Path Equivalence

Ross A. Knepper and Matthew T. Mason

***Abstract*— We propose a motion planning algorithm that reasons about tradeoffs between the discrete decision problems and continuous optimization problems faced by a mobile robot navigating through a cluttered environment. Discrete decisions typically involve transient options as the robot selects corridors to traverse. By contrast, optimization occurs within open spaces among homotopic paths. We utilize properties of local path sets to detect decisions of immediate importance and select routes that maximize the chance of future success.**

I. Introduction

A mobile robot navigating through the world faces a variety of kinds of choices. Where corridors split from each other, there is a discrete decision problem. A robot approaching a decision point at constant velocity has a limited time in which to freely select a corridor without the penalty of backtracking. Such decisions trade off among length, complexity, and safety of routes.

A different sort of choice is the continuous optimization of selecting a path while traveling within a free space, such as in a corridor. In this case, the robot primarily trades off risk of collision with shortness of path. We illustrate these two types of choice in Fig. 1.

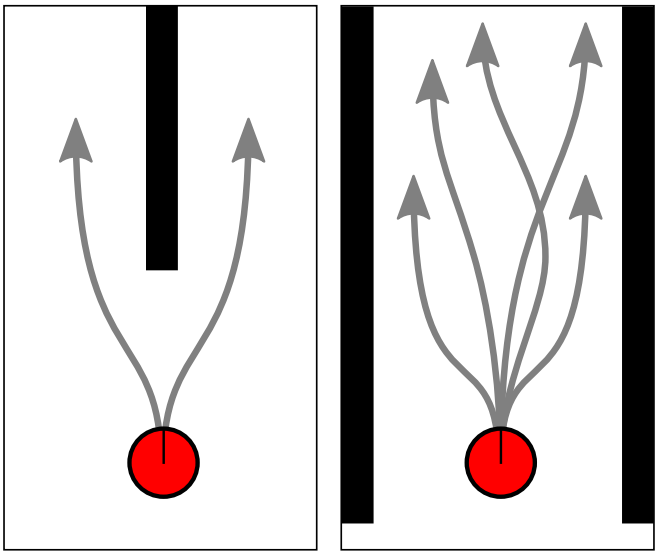

Fig. 1. A navigating robot faces both discrete decisions (left) about which corridor to follow and continuous optimization (right) over where in the corridor to drive. A planner or controller should be able to consider these choices separately.

These two qualitatively distinct choices are reflected in the variety of approaches to motion planning. Many planners decompose the continuum search space into a graph, thus transforming the planning problem into a graph search (a discrete decision process). Early examples deterministically decompose space into a regular grid, including Barraquand and Latombe [2], who discretize the configuration space and action space to generate graph nodes and edges. Other deterministic decision-based methods construct a graph adaptively based on the geometric shape of obstacles [6].

In more recent years, probabilistic planners have become popular. Algorithms like probabilistic roadmaps (PRMs) [8], rapidly exploring random trees [13], and lazy PRMs [17] take advantage of the asymptotic low dispersion of random sequences to sample uniformly throughout complex and high-dimensional configuration spaces without prescribing a fixed density a priori.

Another set of algorithms directly considers motion in the continuum. Khatib's potential fields [10] produce a smooth path in the C-Space by following the gradient of a function combining weighted penalty terms for obstacle proximity with a reward term for progress toward the goal. Through tuning the weights on these terms, a robot may achieve reasonable goal-directed behavior, but the formulation is subject to local minima in of the potential field so that the robot may fail to reach its goal. Rimon and Koditschek proposed the navigation function [16], which assures a potential function free of local minima under certain geometric assumptions on obstacles. Ratliff, et al. introduce CHOMP [15], which performs functional gradient descent on a C-Space path to improve on an initial naive straight-line path by minimizing depth of penetration into obstacles.

All of the above motion planning approaches view the problem as either purely discrete or purely continuous. By contrast, there has been a limited amount of work on hybrid planners that reason about both decision problems and optimization problems. Quinlan's elastic bands [14] optimize an initial collision-free C-Space path generated by a discretizing grid planner—using a functional gradient technique similar to CHOMP—to keep the path safely away from obstacles, even if the obstacles move during execution. Brock's elastic strips [4] operate similarly but in the robot's workspace, thereby alleviating the problem of high-dimensional search for highly articulated systems. Since elastic bands, elastic strips, and CHOMP all operate by decoupling global planning and local optimization, they each tend to find solutions within a single homotopy class. If the chosen class

R. A. Knepper and M. T. Mason are with the Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, USA. rak@ri.cmu.edu, mason@cs.cmu.edu

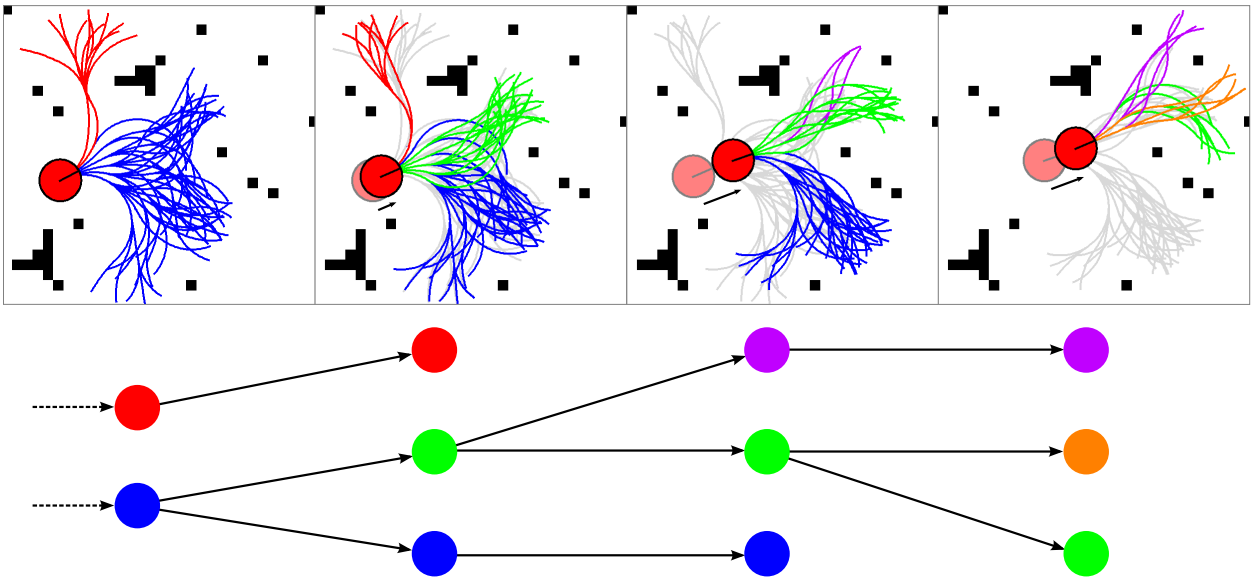

Fig. 2. Between replan cycles, safe paths are associated by a logical succession of equivalence classes. This strict partial ordering relation is represented by a directed acyclic graph. Between cycles, we may detect a termination, split, merge, or mere continuation of the previous equivalence classes. By preferring the logical successor of the previously-commanded path, subsequent path selections give better performance.

later degrades in quality, these algorithms have difficulty weighing the tradeoffs between further local optimization and switching to a new homotopy class.

An alternative hybrid approach by Howard [7] first generates a graph with regular discretization in the C-Space, then optimizes both nodes and edges in the graph to improve the quality of potential paths. Only after optimization completes is the discrete graph search performed. This approach elegantly balances the dual continuous/discrete choice, but this elegance comes at the cost of expensive optimization on many paths that will never be executed by the robot, thus potentially incurring significant up-front computation time prior to the onset of execution.

We introduce a planning algorithm that can simultaneously consider both decision-type and optimization-type choices and even reason about tradeoffs between the two types. The planner considers a set of locally-distinct routes, defined by an equivalence relation, that capture the notion of decision points. It selects among those routes and also selects paths within a route to optimize for safety, goal-directedness, and future flexibility in decision-making.

II. Equivalence Relation

In earlier work [12], we relate discrete and continuum search through an equivalence relation on local paths. In this formulation, two paths are considered equivalent if there exists a continuous deformation between them. This notion differs from homotopy in that we relax the endpoint constraint and instead impose constraints on path shape, such as bounded curvature and fixed length. By recognizing this equivalence, the planner may employ a finite set of paths safely traversing a corridor to represent the continuum of all such paths. Various equivalence classes represent different possible routes amongst obstacles to reach the goal. To rapidly detect equivalence, we employ the Hausdorff metric:

$$\mu_H(p_i, p_j) = \inf\{\varepsilon \colon p_i \subset (p_j)_\varepsilon \text{ and } p_j \subset (p_i)_\varepsilon\},$$

where $(p)_\varepsilon$ describes a dilation of p by radius ε. This metric predicts path equivalence when two paths are sufficiently proximal that no obstacle could fit between them:

$$\mu_H(p_i, p_j) \leq d \implies p_i \sim p_j, \tag{1}$$

where d is the diameter of the robot's swept area or volume (swath) in the workspace. Under this condition, the swaths of two collision-free paths overlap, thus ensuring that a continuum of paths between these two bounds is also collision-free.

We construct from this relation an equivalence graph in which nodes correspond to collision-free paths. An edge connects two nodes if the corresponding paths satisfy (1). Thus, the number of connected components in the graph corresponds to the number of locally-distinct (non-equivalent) routes available to the robot. Equivalence classes are separated from each other by obstacles. When the robot chooses to follow one equivalence class, the others will soon become unavailable to it due to shape constraints on paths. These decisions are therefore important to prioritize.

III. Hierarchical Planning Algorithm

We employ a hierarchical planner with two levels. A global planner provides low-fidelity guidance on the global topology of space, thus instructing the robot which direction is towards the goal. We use D* in an 8-connected grid due to its speed and ability to handle changing environments.

Meanwhile, a local planner, running the Local Planner Algorithm (Alg. 1), runs at high frequency and models robot motions at high fidelity near the current position of the robot. Rapid replanning provides reactiveness to changes in the environment at the price of limited look-ahead for each path

Algorithm 1 $p \leftarrow$Local_Planner_Algorithm(w, x, h, $\mathcal{P}$)

Input: w – a costmap object; x – initial state; h – a heuristic function for selecting a path to execute;
$\mathcal{P}$ – a fixed set of paths
Output: p – path to execute

```
// Called repeatedly while not at goal and time not expired
P_free ← ∅
while time not expired and untested paths remain do
    p ← Get_Next_Path(P)
    collision ← w.Test_Path(p)
    if not collision then
        P_free ← P_free ∪ {p}
return p ← h.Best_Path(x, P_free)
```

tested. Typically, only a small fraction of a local path is executed before the next command is issued. To accelerate path testing throughput, we precompute a low-dispersion set of paths that produce diverse actions.

Since the local planner has a limited horizon, the resulting planned route is a concatenation of paths from the local and global planners. Since global paths are infeasible to execute directly on the robot, the local planner replans at regular intervals to allow continued progress. Thus, Alg. 1 outputs a sequence of paths. During each replan cycle, the planner executes the planned route representing the shortest time to the goal (sum of local and global paths), a heuristic known as *Best_Path*. Traditional hierarchical planners incorporating such a heuristic function produce strongly goal-directed behavior that comes with two drawbacks: temporal incoherence and excessive obstacle proximity.

A. The Temporal Incoherence Problem

Temporal incoherence arises from the fact that *Best_Path* does not ensure consistent behavior between replan cycles. Often in hierarchical planning [1, 9], the ultimate route executed by the Local Planner Algorithm is an emergent behavior because the planner lacks any continuity of intent between consecutive replan cycles. We propose local path equivalence as a means of representing such continuity. In producing a sequence of local paths, these local planners tacitly also select a sequence of equivalence classes. Path equivalence means that, based on information available to the local planner during a given replan cycle, the planned routes of all equivalent paths are homotopic. We propose a new algorithm to improve navigation performance by explicitly considering such consistency during each replan cycle.

In general, we expect that each replan cycle should select a new path that closely resembles the old path. In prior work [11], we proposed increasing the chance of such an outcome with the *Best_Path* heuristic by preserving the unexecuted remainder of the old path as a *continuation*, which is considered along with the ordinary path set during subsequent cycles. Even so, on some occasions, consecutive replan cycles may switch equivalence classes, thus selecting a new planned route. *Best_Path* does not distinguish between classes, so such switches may happen arbitrarily often. Frequent switching often arises from perception noise. In noisy systems where two planned routes are about equally costly, the planner may rapidly alternate between them, thus effectively following an unplanned and undesirable path directly towards the obstacle separating the two routes.

B. The Obstacle Proximity Problem

Obstacle proximity, the second drawback of *Best_Path*, risks the robot's safety in cases of outside disturbance or internal prediction error. From a planning perspective, nearby obstacles substantially reduce the quantity and diversity of safe paths available in subsequent replan cycles.

Two related approaches to the problem of decreasing robot proximity to obstacles have been in use for many years. The first approach involves "growing" the obstacles using a hard buffer [5], which runs the risk of closing off narrow openings. This problem is partially ameliorated by making the obstacle growth-radius vary in proportion to robot speed.

The second approach involves placing a soft buffer around each obstacle in the form of a gradient of elevated cost [18], such that cost varies inversely with obstacle proximity. While this approach does not eliminate options from consideration, it is difficult to predict how a given cost function will affect decisions between corridors.

The drawback of both approaches is that their tunable parameters couple the two distinct types of choice: which route to follow, and how to proceed along that route. These decisions are of qualitatively different character because continual fine-tuning is possible throughout the traverse of a corridor, but the choice of corridor to be traversed requires a discrete decision (Fig. 1).

C. Improved Hierarchical Planner

We introduce a new multi-stage path selection algorithm that separates these two decisions, thus allowing them to be weighed separately and traded off against one another. This process in turn improves planning and control flexibility, increasing continuity of plans, and retaining goal-directedness. Through application of a set of rules based on path equivalence (applied both within and across replan time steps), the algorithm selects paths for execution that guide the robot sufficiently far from obstacles while moving consistently towards the goal.

IV. Logical Succession Path Relation

We now introduce a relation on path equivalence classes to detect logical succession across multiple replan cycles.

Definition 1 (logical succession): Logical succession is a strict partial ordering among equivalence classes $\mathcal{A} \rhd \mathcal{B}$ such that some paths $p_A \in \mathcal{A}$ and $p_B \in \mathcal{B}$ exist for which $\mu_H(p_A, p_B) \leq d$ and $\mathcal{A}$ was generated in an earlier replan time step than $\mathcal{B}$. □

This definition establishes that two paths covering largely the same terrain during different replan cycles can be said to follow the same route. The definition assumes a small time increment between replan cycles, such that little ground is covered in the interim.

In considering a new path for execution, logical succession provides a powerful tool for a planner to distinguish between

paths that represent major and minor alterations to the prior plan. Suppose the planner just executed path p_i at time step $t-1$. We may initially choose to consider at time t only those paths p_j such that $[p_i] \rhd [p_j]$, where $[p]$ describes the equivalence class containing p. This restriction provides continuity of plan. Often, each equivalence class has only one logical successor at the following replan time step. However, merges and splits may occur at critical points along the robot's traverse (Fig. 2). When the planner detects a split, it is important to select the branch that maximizes success, given the locally-available information.

V. Multistage Path Selection Algorithm

We introduce the Multistage Local Planner Algorithm (Alg. 4) to make principled path selections that trade off among the issues of logical succession, safety, and estimated path length to the goal. At a high level, the algorithm consists of two stages. Stage one selects for consideration a subset of $\mathcal{P}_{free}$ (the set of collision-free local paths) comprising one or more equivalence classes in order to ensure progress, safety, and consistency. Stage two selects from among the chosen subset one path for execution that trades off safety and length (or cost) of the path.

A. Stage One: Solving the Decision Problem

In generally preferring to select a new path from a logical successor class to the previously chosen equivalence class, we largely eliminate sensitivity to noisy perception data.

Two exceptions arise in which the algorithm will not execute a logical successor path. First, if a non-successor equivalence class predicts a significantly shorter distance (or lower cost) to the goal, then the planner switches classes on the assumption that the magnitude of the change exceeds that of likely perception noise. Second, we allow the algorithm to consider broader alternatives (either more or less costly) if all logical successor classes terminate or become *narrow*.

Definition 2 (narrow, wide): A *narrow* equivalence class contains few paths. We employ path count as a proxy for the measure of a corridor in path space. Thus, a low path count indicates little space to maneuver the robot through a narrow corridor. Non-narrow classes are called *wide*. □

We define a constant fraction, PATH_THRESH, which adaptively selects the cutoff in corridor width as a percent of the number of paths in $\mathcal{P}_{free}$. Thus, the more densely we sample the space of paths, proportionately more paths are required to constitute a wide corridor. But since we only consider safe paths, a highly cluttered space with few safe paths permits a relatively narrow passage to be considered "wide" in comparison to others.

This concept of wide and narrow corridors closely resembles that of Borenstein and Koren [3]. Their vector field histogram represents obstacle density projected down to one dimension corresponding to heading. Sparse regions of the histogram indicate corridors, but due to the projection, only the component of corridor width perpendicular to that projection is recorded. Our notion of corridor detection and width estimation is more general since it closely approximates the full capabilities of the robot and is not limited to a particular obstacle configuration or observational perspective.

Algorithm 2 $(\mathcal{W}, \mathcal{N}) \leftarrow$Divide_Wide_Narrow($C$, t)

Input: C – candidate set of classes; t – threshold size of class
Output: $\mathcal{W}$ – set of paths in wide classes; $\mathcal{N}$ – set of paths in narrow classes

$\mathcal{W} \leftarrow \emptyset$
$\mathcal{N} \leftarrow \emptyset$
for all $\mathcal{C} \in C$ **do**
  if $|\mathcal{C}| > t$ **then**
    $\mathcal{W} \leftarrow \mathcal{W} \cup \mathcal{C}$ // Paths in wide classes
  else
    $\mathcal{N} \leftarrow \mathcal{N} \cup \mathcal{C}$ // Paths in narrow classes
$\mathcal{W} \leftarrow Cull\_Nonprogressing\_Paths(\mathcal{W})$
$\mathcal{N} \leftarrow Cull\_Nonprogressing\_Paths(\mathcal{N})$
return $(\mathcal{W}, \mathcal{N})$

Algorithm 3 $p \leftarrow$Optimize_Path(x, h, e, p)

Input: x – initial state;
  h – a heuristic function for selecting a path to execute;
  e – equivalence object
  p – seed path for optimization
Output: Return a path similar to p but safer

repeat
  $\mathcal{N} \leftarrow e.Get\_Neighbors(p) \cup \{p\}$
  $p \leftarrow h.Farthest\_Obstacle\_Path(x, \mathcal{N})$
until p converges or $p.obstacle\_proximity > \frac{3}{2}d$
return p

While Alg. 4 displays a preference for wide logical successor classes, it strictly selects only progressing paths within a class for consideration.

Definition 3 (progressing): A *progressing* path is one for which both of the following two points are nearer to the goal by a lower-bounded distance than is the current robot position, according to the global planner:

1) the point one replan time step in the future, and
2) the end point of the local path. □

The progressing property is often shared by all paths in an equivalence class, but certain large classes can have mixed progressivity. By executing only progressing paths, we ensure that the robot monotonically approaches the goal, thus guaranteeing that the robot can never become stuck in an infinite loop. Alg. 2 is used to eliminate nonprogressing paths and divide the rest according to the narrow/wide dichotomy.

In stage one, the algorithm establishes an order of preference in selecting $\mathcal{S}$, the set of paths for consideration.

1) All wide, progressing, logical successor classes
2) Any wide, progressing class
3) All narrow, progressing, logical successor classes
4) Any narrow, progressing class
5) Return failure

The ordering of choices 2 and 3 expresses a willingness to trade off some extra path length for added safety. To check if the planner has found a highly suboptimal subset of paths, the algorithm then compares the best path in $\mathcal{S}$ against the best path in $\mathcal{P}_{free}$. A difference over a certain SCORE_THRESH provokes a mid-course correction. Such a

Algorithm 4 $(p, \mathcal{L}) \leftarrow$Multistage_Local_Planner_Algorithm(w, x, h, e, $\mathcal{P}$, $\mathcal{L}$)

Input: w – a costmap object;
x – initial state;
h – a heuristic function for selecting a path to execute;
e – equivalence object;
$\mathcal{P}$ – a fixed set of paths;
$\mathcal{L}$ – equivalence class of path selected in prior call (initially $\emptyset$)
Output: p – a path progressing safely toward the goal;
$\mathcal{L}$ – equivalence class of p

$\mathcal{P}_{free} \leftarrow Get\_Safe\_Progressing\_Paths(w, x, \mathcal{P})$
$thresh \leftarrow$ PATH_THRESH $\times |\mathcal{P}_{free}|$
$b \leftarrow h.Best\_Path(x, \mathcal{P}_{free})$ // Greedy shortest path
// Stage 1: select equivalence classes for consideration
if $\mathcal{L} \neq \emptyset$ **then** // Compute successor path candidates
  $C \leftarrow e.Get\_Successor\_Classes(\mathcal{P}_{free}, \mathcal{L})$
  $(\mathcal{W}_s, \mathcal{N}_s) \leftarrow Divide\_Wide\_Narrow(C, thresh)$
else
  $(\mathcal{W}_s, \mathcal{N}_s) \leftarrow (\emptyset, \emptyset)$
$E \leftarrow e.Compute\_Equivalence\_Classes(\mathcal{P}_{free})$
$(\mathcal{W}, \mathcal{N}) \leftarrow Divide\_Wide\_Narrow(E, thresh)$
if $\mathcal{W}_s \neq \emptyset$ **then**
  $\mathcal{S} \leftarrow \mathcal{W}_s$ // Wide successor classes
else if $\mathcal{W} \neq \emptyset$ **then**
  $\mathcal{S} \leftarrow \mathcal{W}$ // Any wide, progressing class
else if $\mathcal{N}_s \neq \emptyset$ **then**
  $\mathcal{S} \leftarrow \mathcal{N}_s$ // Narrow successors
else if $\mathcal{N} \neq \emptyset$ **then**
  $\mathcal{S} \leftarrow \mathcal{N}$ // Last resort: take any path
else
  return failure
$p \leftarrow h.Best\_Path(x, \mathcal{S})$
if $p.score - b.score >$ SCORE_THRESH **then**
  $\mathcal{S} \leftarrow \mathcal{P}_{free}$ // Jump equivalence classes
  // Stage 2: Select one path from the set
$p \leftarrow h.Best\_Path(x, \mathcal{S})$
$p \leftarrow Optimize\_Path(x, h, e, p)$
$\mathcal{L} \leftarrow e.Class\_Of(p)$
return $(p, \mathcal{L})$

switch of equivalence class should be a rare event.

Note that we are making a choice with global implications based on unreliable information from a low-fidelity global planner. Lacking detailed knowledge of the complete path to the goal, we instead consider a calculation based solely on average obstacle density, which predicts that a narrow corridor "pinch point" should occur periodically at some frequency during traversal. Given a distance remaining to reach the goal, an expected number of risky narrow corridors can be estimated. SCORE_THRESH should be chosen so that the decreased risk (stemming from the shorter path length) of getting stuck in a future narrow corridor outweighs the immediate risk involved in the current route change, which may itself jump to a narrower corridor.

B. Stage Two: Solving the Optimization Problem

After establishing a final set of candidate paths $\mathcal{S}$, the algorithm moves on to stage two. Initially, it finds the greedy *Best_Path* option in $\mathcal{S}$, but this path may come unsafely close to an obstacle. Within the equivalence class containing the shortest path, the subroutine *Optimize_Path* performs a local, gradient-descent-type optimization in path space by traversing the equivalence graph. This optimization, which generates a soft safety buffer around obstacles, seeks to maximize the distance to the one nearest obstacle.

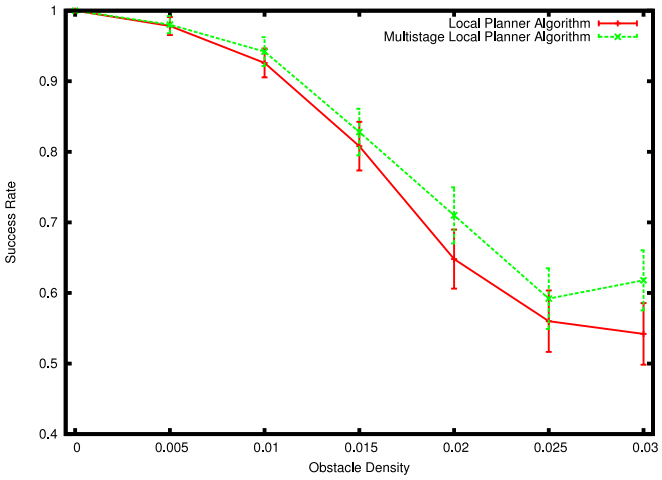

Fig. 3. Improvement in overall planner success rate. At high clutter, Alg. 4 significantly outperforms Alg. 1. Note that 0.03 obstacle density in the workspace corresponds to approximately 67% C-space density.

In especially wide corridors, the robot should be free to follow a reasonably short path, so the obstacle proximity penalty decays to zero beyond 1.5 robot diameters. The proximity penalty function is only defined with respect to the one nearest obstacle, so in a narrow corridor the penalty is locally minimized by the path most nearly following the center of the corridor, thus maximizing both safety and future planning options. Note that the algorithm will follow even an extremely narrow corridor, provided that the route represents the best means to progress towards the goal.

Ultimately, the algorithm we describe here improves on the original Local Planner Algorithm by executing an action that is safe, maximizes future planning/control options, and remains consistent across replan cycles, all while retaining goal-directedness.

VI. Results

We tested the Multistage Local Planner Algorithm (Alg. 4) in simulation over a variety of environments at a range of obstacle densities in order to evaluate the benefits of local equivalence class awareness. All planning problems require the robot to reach a goal 14 m distant within a 20×20 m environment containing randomly scattered point obstacles. Failure occurs if the local planner finds no valid options during a planning cycle. Each data point in the plots represents an average over 500 different problems.

Fig. 3 shows success rate for the Local Planner Algorithm (greedy) and Multistage Local Planner Algorithm (equivalence aware). The latter produces a statistically significant improvement at solving planning problems in dense clutter. Path length increases only negligibly, and despite the extra path length, we find a decreased path cost, expressed as

$$c(p) = \int_p \frac{ds}{od(s)},$$

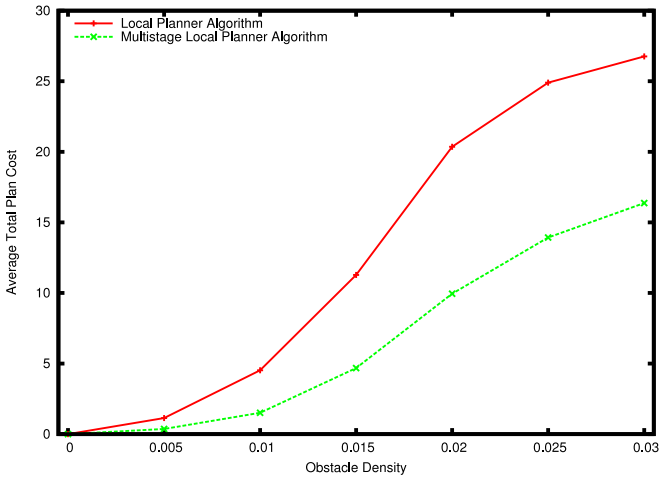

Fig. 4. Cost function penalizing obstacle proximity and path length.

where $od(s)$ is the distance to the nearest obstacle from the given point along the path (Fig. 4).

VII. Summary

We have introduced a method for addressing a shortcoming of hierarchical planning: that the future intent of a plan is forgotten between replan cycles. By recognizing both individual paths as well as their corresponding local equivalence class, we define a general notion of logical succession between replan cycles. Through principled choices, our planner weighs the benefits of continuing along the previously planned path versus switching to a dramatically shorter route.

VIII. Acknowledgments

This work is sponsored by the Defense Advanced Research Projects Agency. This work does not necessarily reflect the position or the policy of the Government.

References

[1] T. Allen, J. Underwood, and S. Scheding. A path planning system for autonomous ground vehicles operating in unstructured dynamic environments. In *Proc. Australasian Conference on Robotics and Automation*, 2007.

[2] J. Barraquand and J.-C. Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10(2-3-4):121–155, 1993.

[3] J. Borenstein and Y Koren. The vector field histogram—fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, 1991.

[4] O. Brock and O. Khatib. Elastic strips: A framework for motion generation in human environments. *International Journal of Robotics Research*, 21(12):1031–1052, December 2002.

[5] J. Buhmann, W. Burgard, A. B. Cremers, D. Fox, T. Hofmann, F. E. Schneider, J. Strikos, and S. Thrun. The mobile robot rhino. *AI Magazine*, 16(2):31–38, 1995.

[6] H. Choset and J. Burdick. Sensor based planning, part I: The generalized Voronoi graph. In *Proc. International Conference on Robotics and Automation*, pages 1649–1655, 1995.

[7] T. Howard. *Adaptive Model-Predictive Motion Planning for Navigation in Complex Environments*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 2009.

[8] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In *Proc. International Conference on Robotics and Automation*, pages 566–580, 1996.

[9] A. Kelly, A. Stentz, O. Amidi, M. W. Bode, D. Bradley, A. Diaz-Calderon, M. Happold, H. Herman, R. Mandelbaum, T. Pilarski, P. Rander, S. Thayer, N. M. Vallidis, and R. Warner. Toward reliable off road autonomous vehicles operating in challenging environments. *International Journal of Robotics Research*, 25(1):449–483, May 2006.

[10] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proc. International Conference on Robotics and Automation*, St. Louis, USA, March 1985.

[11] R. A. Knepper and M. T. Mason. Path diversity is only part of the problem. In *Proc. International Conference on Robotics and Automation*, Kobe, Japan, May 2009.

[12] R. A. Knepper, S. S. Srinivasa, and M. T. Mason. An equivalence relation for local path sets. In *The Ninth International Workshop on the Algorithmic Foundations of Robotics*, Singapore, December 2010.

[13] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In B. R. Donald, K. M. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pages 293–308. A K Peters, Wellesley, MA, 2001.

[14] S. Quinlan and O. Khatib. Elastic bands: connecting path planning and control. In *Proc. International Conference on Robotics and Automation*, pages 802–807, Atlanta, USA, May 1993.

[15] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *Proc. International Conference on Robotics and Automation*, May 2009.

[16] E. Rimon and D. E. Koditschek. Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation*, 8(5), October 1992.

[17] G. Sánchez and J.-C. Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In *Proc. International Symposium on Robotics Research*, Lorne, Victoria, Australia, November 2001.

[18] C. E. Thorpe. Path relaxation: Path planning for a mobile robot. In *Proc. National Conference on Artificial Intelligence*, pages 318–321, 1984.