Learning-Assisted Multi-Step Planning

Kris Hauser, Tim Bretl, Jean-Claude Latombe
Stanford University
Stanford, CA 94307, USA
khauser@cs.stanford.edu, *tbretl@stanford.edu*, *latombe@cs.stanford.edu*

***Abstract*— Probabilistic sampling-based motion planners are unable to detect when no feasible path exists. A common heuristic is to declare a query infeasible if a path is not found in a fixed amount of time. In applications where many queries must be processed – for instance, robotic manipulation, multi-limbed locomotion, and contact motion – a critical question arises: what should this time limit be? This paper presents a machine-learning approach to deal with this question. In an off-line learning phase, a classifier is trained to quickly predict the feasibility of a query. Then, an improved multi-step motion planning algorithm uses this classifier to avoid wasting time on infeasible queries. This approach has been successfully demonstrated in simulation on a four-limbed, free-climbing robot.**

***Index Terms*— Motion planning, multi-step planning, machine learning, climbing robot.**

I. Introduction

Probabilistic sampling-based motion planners, such as the Probabilistic Roadmap (PRM) approach [1], are popular in robotics because they can handle many degrees of freedom and diverse feasibility constraints while remaining fast and easy to implement. Under assumptions that are usually satisfied in practice, the probability that such a planner finds a feasible path between two configurations (whenever one exists) approaches 1 quickly as more time is allowed [2].

However, these planners are generally unable to detect when no feasible path exists. A common heuristic is to declare a query infeasible if a path is not found in a fixed amount of time – of course, this result may be incorrect. In applications where many queries must be processed, a large fraction of which have no solution, a critical question arises: what should this time limit be? Too low, and some feasible queries may not be solved; too high, and infeasible queries dominate total running time.

Multi-step motion planning, which computes paths traversing multiple configuration spaces, faces this dilemma [3]. It is encountered in robotic manipulation [4], [5], multi-limbed locomotion [3], [6], and contact motion [7]. These applications usually require searching large graphs where nodes are configuration spaces associated with different states of contact between the robot and its environment, and edges are feasible transitions between spaces. Checking if an edge connects two nodes corresponds to solving a one-step motion planning query, in a single configuration space. If probabilistic sampling is used, the importance of the time limit is clear. With a low limit, some critical transitions may be incorrectly declared infeasible. With a high limit, most of the time will be spent attempting infeasible transitions.

If we knew in advance which transitions were infeasible, we could prune them from the graph. Then, the PRM planner would be used only to construct feasible one-step motions, not to test their existence. Since all queries would be feasible, the time limit for the PRM planner could be set high. In this paper, we apply a machine-learning approach to create a classifier that quickly provides this feasibility information. We present a probabilistic multi-step planner that uses this classifier but is robust to classification errors. We demonstrate our planner in simulation for the four-limbed, free-climbing robot introduced in [8].

Our approach is applicable to other multi-step planning problems, as long as the structure of the robot's feasible space can be captured sufficiently using machine learning – parameterization and dimensionality are common limitations. For example, the feasible space of the climbing robot is shaped by constraints that depend on a relatively small number of parameters. Non-gaited locomotion of humanoid robots on rough terrain has a similar characteristic.

II. Related Work

The idea of using experimental data to statistically model problem difficulty has attracted attention in computational theory. The distribution of hard-vs.-easy instances of the NP-complete 3-SAT problem was studied in [9]. It was found that problems whose ratios of clauses to variables are either high or low are easily solvable with either depth-first or local search techniques, with exponentially difficult problems lying in between. In [10], the running time of the NP-complete combinatorial auction winner determination problem was predicted using regression on a database of problem examples, solved by a branch-and-bound search. In these previous works, the most important issue was finding the few parameters that best capture algorithm behavior. Here, we seek to predict whether or not a problem admits a solution.

Closer to our work, models of problem difficulty have been used to speed up multi-step planning. For example, the "fuzzy PRM" approach assigns probabilities of success to the edges of the graph searched by a multi-step manipulation planner [5]. Each probability depends on the amount of time spent so far by a PRM planner on the corresponding query. If a path has been found, the probability of the edge is set to 1; otherwise, it decreases toward 0 as more time is spent. One-step planning time is adaptively allocated to edges with higher probability. However, this method has

several drawbacks. First, lowering the probability of an edge requires spending time trying to solve the corresponding one-step query. If the query is actually infeasible, that much time is still wasted. Second, solving some feasible one-step queries might take a long time – either by "bad luck" in PRM sampling or because they are difficult (e.g., involve a "narrow passage" [2]). Such a query might be critical to connect a multi-step path. In this case, many other infeasible queries will be tried before more time is allocated to solve the critical one.

Alternatively, a method of "disconnection proof" has been proposed to directly check edge infeasibility [3]. This technique uses computational real algebra and semidefinite programming to construct a disconnection manifold – an explicit proof of infeasibility – if it exists. However, the computational cost of this technique is prohibitive. It is only practical to search for "simple" disconnection manifolds (i.e., those defined by low-order polynomials).

Our classifier also checks edge infeasibility directly, but is faster to compute (at the expense of a lengthy learning phase). We borrow from [5] the idea of labeling edges of the multi-step planning graph by probabilities and using these probabilities to invest PRM planning time. However, since our classifier predicts infeasibility rather than difficulty, our method does not suffer from the drawbacks of this prior work.

III. Multi-Step Planning

A. Model and Problem

We consider the multi-step planning problem for a free-climbing robot, LEMUR IIB, shown in Fig. 1. This robot climbs an artificial rock surface by making contact at its limb endpoints, which we call *hands*. It moves on the terrain much like a human rock climber, by moving one hand at a time, from one contact to another, only using friction to maintain equilibrium. We model each contact as a single point, called a *hold*, parameterized by its position, normal, and Coulomb friction coefficient. In this paper, holds are presurveyed; in a practical setting, they would be extracted along protrusions, holes, cracks, or ledges, using visual and tactile sensors.

Each limb of LEMUR IIB has three revolute joints, providing two in-plane and one out-of-plane degrees of freedom (DOF). The in-plane DOF move the links of each limb parallel to the terrain. The out-of-plane DOF are used only to make or break contact with holds. Ignoring the out-of-plane joints, the dimensionality of the robot's configuration space C is 11 (including three parameters defining the planar position and orientation of the chassis).

We refer to each set of hand-terrain contacts used by the robot as a *stance*. We consider stances in which three or four limbs are contacting holds (in a 3-hold stance, the fourth limb is said to be *free*). For simplicity, we ignore 1- and 2-hold stances, which are rarely useful in practice. The closed-chain kinematic constraints imposed by these contacts at a 4-hold (resp., 3-hold) stance σ restrict the set of configurations to a 3-D (resp., 5-D) submanifold C_σ of the original configuration space C. Several parameterizations of this manifold are possible; a simple one is specified by the planar position and orientation of the chassis and the joint angles of the free limb [3].

Fig. 1. The LEMUR IIB climbing robot [8].

A configuration in C_σ is feasible if it is not in collision, and if forces can be applied to the contact holds, without slip, that exactly compensate for the gravity force. These conditions are described in detail in [11]. We denote by F_σ the subset of feasible configurations in C_σ, and call it the *feasible space* at σ.

If a 3-hold stance σ and a 4-hold stance σ' share three contacts, then it is possible for a configuration to be feasible at both. Such configurations $q_t \in F_\sigma \cap F_{\sigma'}$ are critical to multi-step motion, since they represent points at which the robot can switch between two different contact states. We say that σ and σ' are *adjacent* and call $F_\sigma \cap F_{\sigma'}$ the *transition* between them.

To reach a distant hold, LEMUR IIB must traverse a number of feasible spaces at 3- and 4-hold adjacent stances. Its motion within each feasible space $F_{\sigma'}$ is a continuous path between points $q_t \in F_\sigma \cap F_{\sigma'}$ and $q_{t'} \in F_{\sigma'} \cap F_{\sigma''}$ in consecutive transitions. We call this continuous path a *one-step motion* through σ'.

The multi-step planning problem for LEMUR IIB is to construct a sequence of feasible one-step motions connecting given start and goal configurations $q_s \in F_{\sigma_s}$ and $q_g \in F_{\sigma_g}$ at initial and final stances σ_s and σ_g.

B. Basic Algorithm

We view multi-step planning as the search through a graph, in which nodes are transitions and edges are one-step motions. It is convenient to index each node by a specific configuration q_t in the corresponding transition.

At each step of the search, we select a node

$q_t \in F_\sigma \cap F_{\sigma'}$ to expand. We would like to determine whether a one-step motion is possible from q_t through σ' to each adjacent stance σ''. Two conditions must be satisfied for this one-step motion to exist. First, the transition $F_{\sigma'} \cap F_{\sigma''}$ between stances σ' and σ'' must be nonempty. Second, both q_t and some configuration $q_{t'} \in F_{\sigma'} \cap F_{\sigma''}$ must be in the same connected component of $F_{\sigma'}$. The resulting node, indexed by $q_{t'}$, corresponds to the transition $F_{\sigma'} \cap F_{\sigma''}$.

It is computationally impractical to test each of these conditions exactly. Instead, we use probabilistic sampling. The transition $F_{\sigma'} \cap F_{\sigma''}$ is declared nonempty if a single feasible configuration $q_{t'}$ is sampled from $C_{\sigma'} \cap C_{\sigma''}$ (as described in [11]). Likewise, q_t and $q_{t'}$ are in the same connected component of $F_{\sigma'}$ if a continuous path is constructed between them by the PRM planner described in [8].

However, this approach raises the dilemma discussed in Section I. Since probabilistic sampling can not prove the emptiness of a space, nor can a PRM planner demonstrate that two configurations are disconnected, how much time should be spent checking each condition before it is assumed to be unsatisfied? Too little, and critical one-step motions may be missed; too much, and testing infeasible conditions dominates computation time.

One way to alleviate this problem is to use a "lazy" search technique that postpones costly computations [12]. Our algorithm proceeds in two stages. First, we construct a multi-step path via breadth-first search, assuming that each one-step motion from a transition $F_\sigma \cap F_{\sigma'}$ to a transition $F_{\sigma'} \cap F_{\sigma''}$ is feasible only if we can sample a configuration $q_{t'}$ in $F_{\sigma'} \cap F_{\sigma''}$ (disregarding the second condition). Then, we use our PRM planner to compute one-step motions between each consecutive $q_t, q_{t'}$. If at any stage the time limit is exceeded, we discard the corresponding edge and search for another multi-step path. Henceforth, we refer to this algorithm as Basic-MSP.

There are several details of our implementation that we do not discuss here. In particular, since transitions may have multiple components, our algorithm may introduce several nodes in the search graph corresponding to the same transition.

Basic-MSP gives reasonably good results for the LEMUR IIB robot. However, experiments reported in [3] show that total planning time still increases quickly with the time limit allocated to the probabilistic sampler, indicating that much time is wasted trying to sample empty feasible spaces.

IV. Learning a Classifier

A. Overview

Our goal is to construct a classifier that quickly rejects empty transitions, so they do not have to be sampled during multi-step search. Our approach is to learn this classifier offline, using training data from many runs of Basic-MSP in randomly generated terrain. We call the classifier Θ, and define it as a function on adjacent stances:

$$\Theta : \sigma, \sigma' \mapsto \begin{cases} 0 \text{ if } F_\sigma \cap F_{\sigma'} \text{ is empty,} \\ 1 \text{ otherwise.} \end{cases}$$

In the following sections, we describe supervised learning of machine learning models to represent Θ. Some commonly used techniques are least-squares regression, k-nearest neighbors, Bayesian networks, neural networks [13], and support vector machines [14], [15]. For this problem we have experimented with several, and have had the best results with the latter two. Critical issues in the design of our models are the parameterization and sampling of training data; these issues are discussed below.

It may also be useful to learn a *path classifier* to avoid calling the PRM planner on infeasible queries. But, learning this classifier seems much more difficult than learning Θ. Moreover, as will be shown later, calculating one-step queries does not dominate the computation time of Basic-MSP. So, we leave the construction of this classifier for future research, if needed.

B. Parameterization

Parameterizing the domain of Θ is straightforward. The function Θ classifies the transition $F_\sigma \cap F_{\sigma'}$ between a pair of adjacent 3-hold and 4-hold stances, σ and σ', which have three holds in common. This transition is uniquely defined by the four holds in σ'. Each hold is defined by four parameters: two coordinates, the normal orientation, and the Coulomb friction coefficient. Since the feasibility of a stance is translation-invariant, we factor out two position parameters. Also, the current one-step planner assumes constant friction (set conservatively to a low value), so we eliminate friction from the parameter space. Finally, the normal orientation of the hold that σ and σ' do not have in common is irrelevant to the shape of the feasible space, since the force exerted at this hold during transition must be zero. Consequently, the domain of Θ has a 9-dimensional parameterization.

Symmetries exist in this parameterization, since the robot's limbs are identical and evenly distributed around a circular chassis. We use these symmetries to further reduce the volume (but not the dimensionality) of the domain.

C. Sampling

We create a set of labeled training *examples* as follows: first, we generate points (σ, σ') in the domain of Θ by picking values of the parameters at random; then, we run the probabilistic sampler used in Basic-MSP to determine the desired value of Θ at each point. That is, if a configuration is successfully sampled in $F_\sigma \cap F_{\sigma'}$, the point is labeled 1; otherwise it is labeled 0. We reduce labeling errors by allowing much more time for the sampler than is permitted in Basic-MSP.

Unfortunately, the set of points that are feasible is typically a small fraction of the domain (about 2-5% for the LEMUR IIB robot). In particular, it is difficult to sample points that are close to the "decision boundary" separating nonempty transitions from empty ones. Consequently, a very large number of sampled points would be necessary

to fully characterize Θ if they were generated entirely at random.

Therefore, we use an alternative two-step approach. First, we generate a number of points in the domain of Θ uniformly at random, retaining only those that are labeled 1. Then, we select several of these with probability inversely proportional to the density of surrounding points, and generate a number of additional samples through perturbation. By iterating, we produce a database of points densely sampled near the decision boundary, about 35-65% of which are labeled 1.

D. Training

Using the methods above, we generated a database of 100,000 labelled examples, taking 1-2 days of computation on a dual 1 GHz Pentium II computer. From these we create an SVM classifier Θ_{SVM} and a neural network Θ_{NN}.

For SVM training, we used the package *SVMlight* (http://svmlight.joachims.org/) [16], [17]. We optimized SVM parameters using a pattern search technique [18] on 10,000 randomly chosen examples from the database, taking 7 hours. Using the optimal parameters, we then trained Θ_{SVM} from a set of 20,000 randomly picked examples, taking 4 hours.

Θ_{NN} was created by training a three layer backpropagation neural network with 9 input nodes, 100 hidden nodes, and one output node [13]. Training was performed on 50,000 examples. This process took 3 days of computation to achieve sufficient convergence. When complete, we composed the network with a threshold to map the output to $\{0, 1\}$.

We measured classifier accuracy with cross-validation, designating accuracy on feasible and infeasible examples as ϵ_+ and ϵ_-. We found that Θ_{SVM} has accuracy $\epsilon_+ = 84.62\%$ and $\epsilon_- = 83.98\%$, and can be evaluated in 1.40ms. Θ_{NN} has accuracy $\epsilon_+ = 78.34\%$ and $\epsilon_- = 76.94\%$, and can be evaluated in 0.003ms.

V. Planning with Classifiers

A. Graph Pruning

The simplest way to use Θ is to help Basic-MSP prune its search graph faster. Recall that the main drawback of Basic-MSP is the use of costly probabilistic sampling to test whether transitions are empty (Section III-B). We change Basic-MSP in two respects.

First, when considering a new transition $F_\sigma \cap F_{\sigma'}$, we replace the sampler with Θ. If $\Theta(\sigma, \sigma') = 0$ (predicting empty $F_\sigma \cap F_{\sigma'}$), then the transition is pruned. Otherwise, $F_\sigma \cap F_{\sigma'}$ is added as a node of the graph. Second, after the search yields a multi-step path to the goal, we call the probabilistic sampler to find a feasible configuration q_t in each transition $F_\sigma \cap F_{\sigma'}$ along the path. Since $F_\sigma \cap F_{\sigma'}$ is likely to be feasible, the sampler can be allowed a high time limit. If no configuration is found, $F_\sigma \cap F_{\sigma'}$ is removed from the graph and the search resumes. If all transitions have been successfully sampled, one-step PRM planning proceeds as in Basic-MSP. We call this new planner Pruned-MSP.

Unfortunately, Θ is not perfect and occasionally does misclassify transitions. These errors affect Pruned-MSP differently depending on whether or not the transitions are actually empty. For example, it is more or less harmless to misclassify an empty transition as nonempty. If this transition lies on a candidate path to the goal, then probabilistic sampling will later reject it. Thus, the error will merely slow down the planner. On the other hand, misclassifying a nonempty transition as empty is critical, and might cause Pruned-MSP to fail. If many different feasible multi-step paths exist, a few such errors may not hurt the search. Some feasible path will likely still exist in the pruned graph and be found. But if some transitions are critical (e.g., if all feasible multi-step paths eventually pass through them) then a small number of classification errors could disconnect the start and goal node.

One method to reduce errors on feasible transitions is to bias the classifier to have higher accuracy ϵ_+. This is accompanied by a drop in ϵ_-, causing more harmless errors, but hence a slower planner. In addition, unless 100% feasible accuracy is obtained, Pruned-MSP may still fail in the presence of critical transitions.

B. Fuzzy Search

The above shortcomings led us to develop a fuzzy search algorithm, inspired by [5] (Section II). We call this planner Fuzzy-MSP. It operates very much like Basic-MSP, but rather than breadth-first search, it performs uniform-cost [13] search prioritized by probability of feasibility.

When a new transition $T = F_\sigma \cap F_{\sigma'}$ is encountered, it is added to the nodes of the graph. It is then assigned a probability of feasibility $P(T)$ given the result of classification $\Theta(\sigma, \sigma')$, using the posterior probabilities

$$P(T|\Theta(\sigma, \sigma') = 1) = \frac{P_+\epsilon_+}{P_+\epsilon_+ + P_-(1-\epsilon_-)} \quad (1)$$

$$P(T|\Theta(\sigma, \sigma') = 0) = \frac{P_+(1-\epsilon_+)}{P_-\epsilon_- + P_+(1-\epsilon_+)} \quad (2)$$

where P_+ is the prior probability of encountering a feasible transition, $P_- = 1 - P_+$, and ϵ_+ and ϵ_- are the classifier accuracies as measured above.

We define the cost of a path in the search graph as $-\sum_T \ln(P(T))$, where the sum is taken over all the nodes in the path. Uniform-cost search always expands the node that has the lowest path cost from the start node.

Like Pruned-MSP, once Fuzzy-MSP finds a multi-step path, probabilistic sampling is used to pick a configuration in each transition along this path. If such a configuration is found, the probability of the corresponding node is set to 1. On failure, the corresponding node is removed from the graph, and graph search resumes. As in the other planners, once all transitions have been successfully sampled, the PRM planner is queried to connect consecutive configurations. Failure causes an edge to be removed from the graph. During all graph changes, the lowest-cost path

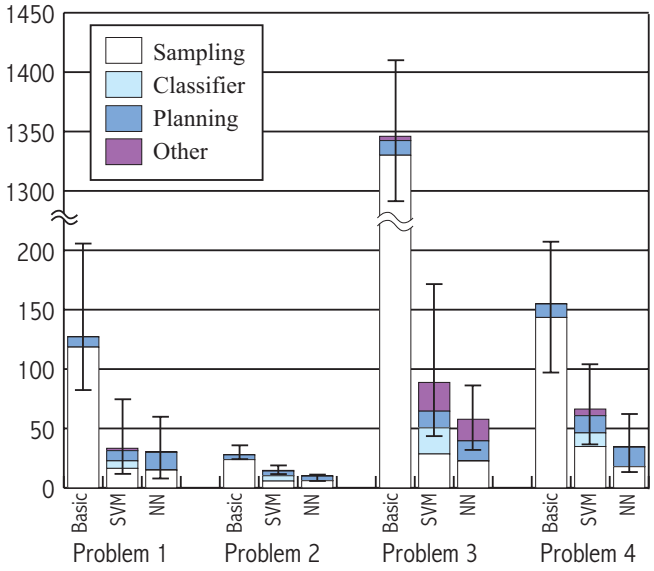

Fig. 2. Comparison of planning times of different algorithms on four test problems. Times are in seconds. Columns indicate average times, extended lines indicate minimum and maximum.

of each node from the start node is dynamically maintained by updating the shortest-path predecessor of each node.

Fuzzy-MSP is as robust as Basic-MSP, since the classifications of Θ are never completely trusted. But it is much faster, because Θ is both correct most of the time and very fast. On the other hand, Fuzzy-MSP is more robust than Pruned-MSP, but only slightly slower, as it creates more nodes and maintains lowest-cost paths. We therefore suggest Fuzzy-MSP should be used for any imperfect classifier.

VI. Results

We now provide experimental results that empirically validate our approach. These results were obtained on four example problems for LEMUR IIB. We denote a problem as "m-step" if the shortest multi-step path from start to goal requires m one-step motions (m corresponds to the minimum depth of the multi-step search graph). Likewise, we denote a problem as "n-hold" if it occurs in a terrain containing n holds (n is roughly correlated to the branching factor of the graph). Problem 1 (see Fig. 3) is a 34-hold, 10-step problem. Problem 2 is a 14-hold, 24-step problem. Problem 3 is a 34-hold, 20-step problem. Problem 4 is a 17-hold, 10-step problem.

A. Planning Time

In Fig. 2 we compare the performance of Basic-MSP, Fuzzy-MSP using Θ_{SVM}, and Fuzzy-MSP using Θ_{NN}. The algorithms are labeled as Basic, SVM, and NN, respectively. Each algorithm was run 10 times on each example problem. The average planning times are plotted in seconds as solid columns, with the minimum/maximum planning time plotted as extending lines. The average times are broken into transition configuration sampling (Sampling), running the classifier Θ_{SVM} or Θ_{NN} (Classifier), PRM planning (Planning), and other calculations such as initialization and graph operations (Other).

Clearly, the bulk of computation time of Basic-MSP is spent in the configuration sampler during graph exploration. Fuzzy-MSP is several times faster than Basic-MSP in each problem, primarily because our method reduces the number of transitions sampled.

In these four tests, neural networks have a slight advantage over support vector machines, although we have encountered problems where the reverse is true. We note that the choice of classifier is fairly unimportant; a significant benefit is achieved by any reasonably efficient and accurate classifier.

TABLE I
CHARACTERISTICS OF BASIC-MSP

Transitions feasible / total sampled	423 / 2865
PRM failures	4.3
Path length	15.2

TABLE II
CHARACTERISTICS OF FUZZY-MSP USING Θ_{SVM}

Transitions predicted feasible / total classified	1353 / 5892
Sampling failures / total	219 / 316
PRM failures	3.7
Path length	26.0

B. Search Characteristics

Table I and Table II compare other characteristics of Basic-MSP and Fuzzy-MSP using Θ_{SVM}, averaged over the same 10 runs of Problem 1. For simplicity we omit the neural-network algorithm.

Table I lists the number of transitions in which the probabilistic sampler has been run. The great majority of transitions are infeasible, causing Basic-MSP to waste much time attempting to sample infeasible transitions. Table II lists the number of transitions classified by Θ. Even though Θ is executed for many more transitions than the number sampled in Basic-MSP, this operation is done at a small fraction of the time. Below that, Table II lists the number of transitions for which Fuzzy-MSP calls the configuration sampler. Recall that sampling occurs once Fuzzy-MSP finds a candidate multi-step path to the goal. Since transitions along the paths produced by Fuzzy-MSP are likely to be feasible, probabilistic sampling is reduced to about a tenth of the number of transitions as Basic-MSP.

For both algorithms, 'PRM failures' lists the number of times the PRM planner failed to connect two transitions along a candidate multi-step path. These numbers are quite low, and are characteristic of all problems that we have encountered. This justifies our decision to omit a path classifier.

Finally, 'Path length' lists the number of one-step motions in the resulting motion plan. On average, Fuzzy-MSP returns a path through 26 stances, rather than the minimum 10 stances. In fact, in our trials, it never finds the shortest path. This occurs because Fuzzy-MSP misclassifies

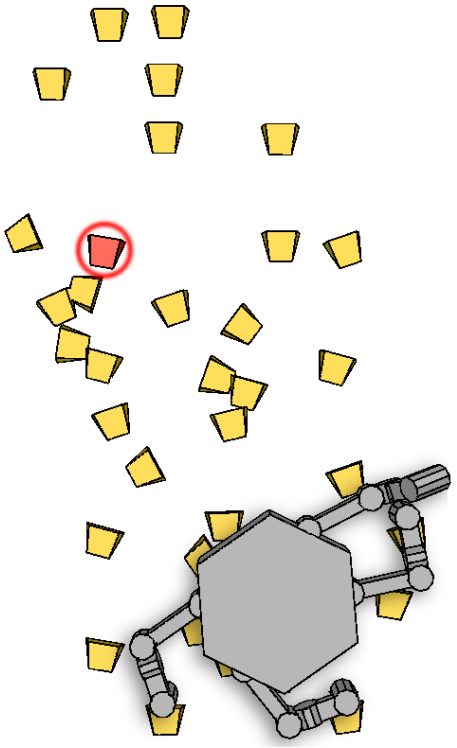

Fig. 3. Terrain of Problem 1. The LEMUR IIB robot is schematically depicted in grey at the start stance. The trapezoidal "ledge" icons depict the holds. Hold positions are at the center of the long end of each icon. Normals point outward from the ledges. The goal hold is circled and colored in red. The climbing plane is inclined at 60 degrees.

one or more feasible transitions along the shortest path. This causes the search to explore higher probability paths before returning to examine the misclassified transitions. Misclassifications happen most often near the decision boundary between feasible and infeasible transitions, where "difficult" transitions are hard to distinguish from infeasible ones. One might speculate that because of this, the paths returned by Fuzzy-MSP could be easier to plan than the shortest path. Whether this is true or not should be investigated in future research.

VII. Conclusion

Using machine-learned classifiers, we have modeled the feasibility of transitions between stances of a four-limbed rock-climbing robot. We demonstrated that this classifier can be combined with a fuzzy search algorithm to greatly increase the efficiency of multi-step planning, with no sacrifice in robustness. Eventually, climbing robots will have to perform multi-step planning on-line, along with sensing and motion control. Learning-assisted planning gets us closer to this goal.

Future work should investigate the effects of classification accuracy and speed on multi-step planning. We are also investigating the use of machine learning to solve other problems faced in motion planning for a rock-climbing robot. One important problem currently being tackled is how to recognize holds from geometric terrain information. Since learned classifiers are quite fast, it might be possible to test thousands of random candidate holds quickly, picking only feasible ones for motion planning. Furthermore, learning a quality measure, such as tolerance to hold measurement error, could prioritize choices that yield safe motions.

Acknowledgment

K. Hauser is supported by a Stanford Graduate Fellowship. The authors would also like to thank the Mechanical and Robotic Technologies Group at JPL and Prof. S. Rock for their continued support of the LEMUR IIB project.

References

[1] L. Kavraki and J. Latombe, "Probabilistic roadmaps for robot path planning," *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, pp. 33–53, 1998.

[2] D. Hsu, J. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," in *Int. J. of Computational Geometry and Applications*, 1999, pp. 495–512.

[3] T. Bretl, S. Lall, J. Latombe, and S. Rock, "Multi-step motion planning for free-climbing robots," in *Sixth International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2004.

[4] R. Alami, J.-P. Laumond, and T. Simeon, "Two manipulation planning algorithms," *Algorithmic Foundations of Robotics*, pp. 109–125, 1995.

[5] C. Nielsen and L. E. Kavraki, "A two-level fuzzy prm for manipulation planning," in *Proceedings of The IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE Press (Refereed), November 2000, pp. 1716–1722.

[6] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue, "Motion planning for humanoid robots," in *International Symposium on Robotics Research*, Sienna, Italy, 2003.

[7] J. Xiao and X. Ji, "A divide-and-merge approach to automatic generation of contact states and planning of contact motion," in *IEEE International Conference on Robotics and Automation*, 2000.

[8] T. Bretl, S. Rock, J. Latombe, B. Kennedy, and H. Aghazarian, "Free-climbing with a multi-use robot," in *9th Int. Symp. on Experimental Robotics*, Singapore, June 2004.

[9] D. Mitchell, B. Selman, and H. Levesque, "Hard and easy distributions of sat problems," in *Proceedings of the Tenth National Conference on Artificial Intelligence*, 1992.

[10] K. Leyton-Brown, E. Nudelman, and Y. Shoham, "Learning the empirical hardness of optimization problems: the case of combinatorial auctions," in *Constraint Programming*, 2002.

[11] T. Bretl, J. Latombe, and S. Rock, "Toward autonomous free-climbing robots," in *International Symposium on Robotics Research*, Sienna, Italy, 2003.

[12] G. Sánchez-Ante and J. Latombe, "On delaying collision checking in prm planning - application to multi-robot coordination," in *Int. J. of Robotics Research*, January 2002, pp. 5–26.

[13] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach. Second edition*. Prentice Hall, 2003.

[14] B. Schölkopf, "Support vector learning," Ph.D. dissertation, Informatik der Technischen Univeritat Berlin, 1997.

[15] C. J. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, pp. 121–167, 1998.

[16] T. Joachims, "Making large-scale svm learning practical," *Advances in Kernel Methods - Support Vector Learning*, 1999.

[17] ——, *SVMlight manual*. [Online]. Available: http://svmlight.joachims.org/

[18] M. Momma and K. P. Bennett, "A pattern search method for model selection of support vector regression," in *Second SIAM International Conference of Data Mining*, 2002.