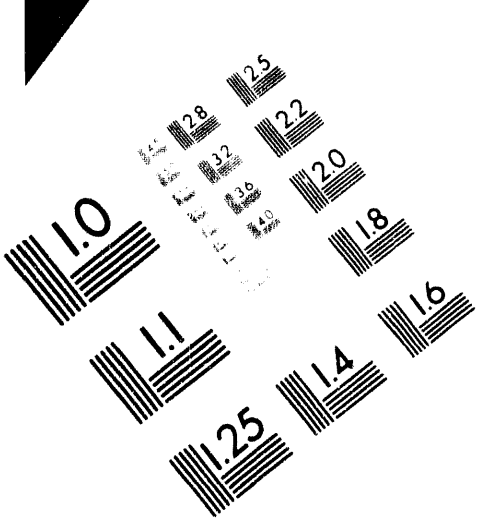
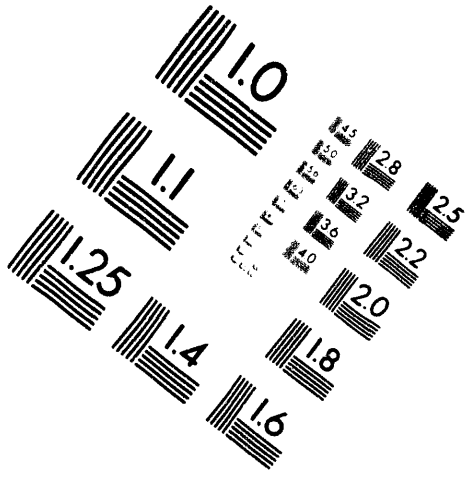
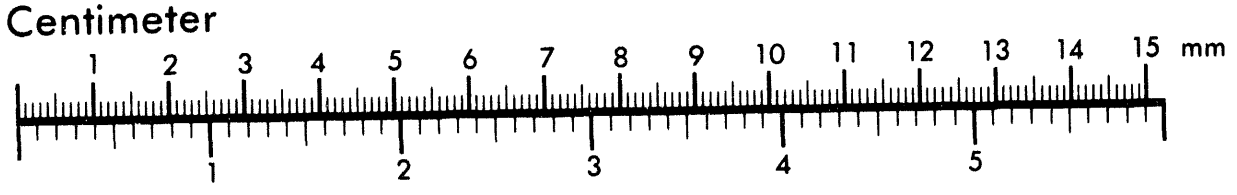


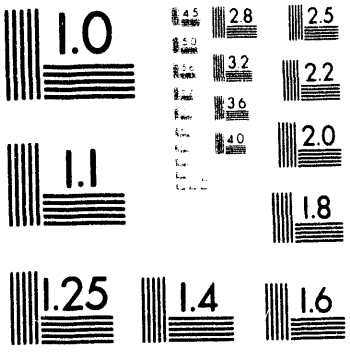
Association for Information and Image Management

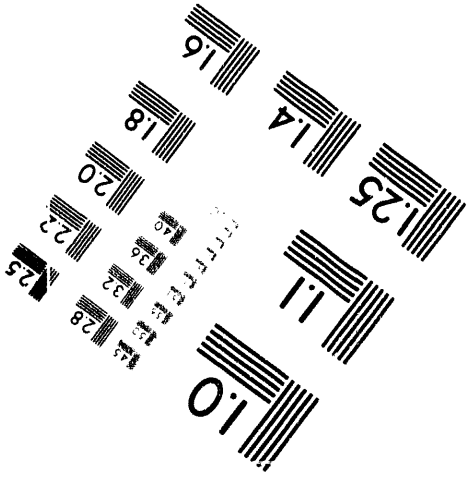
1100 Wayne Avenue, Suite 1100
Silver Spring, Maryland 20910

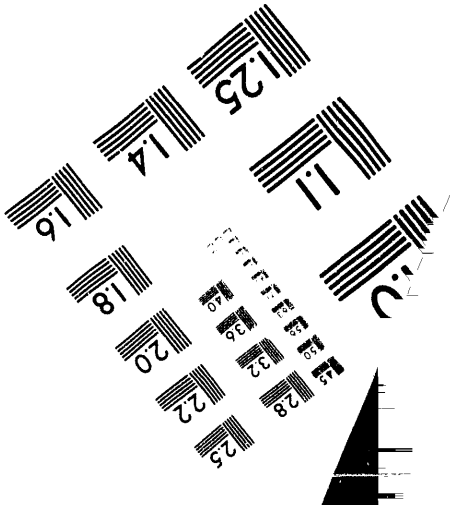
301/587-8202











MANUFACTURED TO AIIM STANDARDS
BY APPLIED IMAGE, INC.

1 of 1

Adaptive Path Planning: Algorithm and Analysis*

(Extended Abstract)

Pang C. Chen
Sandia National Laboratories
Albuquerque, NM 87185

Abstract

Path planning has to be fast to support real-time robot programming. Unfortunately, current planning techniques are still too slow to be effective, as they often require several minutes, if not hours of computation. To alleviate this problem, we present a learning algorithm that uses past experience to enhance future performance. The algorithm relies on an existing path planner to provide solutions to difficult tasks. From these solutions, an evolving sparse network of useful subgoals is learned to support faster planning. The algorithm is suitable for both stationary and incrementally-changing environments. To analyze our algorithm, we use a previously developed stochastic model that quantifies experience utility. Using this model, we characterize the situations in which the adaptive planner is useful, and provide quantitative bounds to predict its behavior. The results are demonstrated with problems in manipulator planning. Our algorithm and analysis are sufficiently general that they may also be applied to task planning or other planning domains in which experience is useful.

1 Introduction

In robotics, path planning refers to finding a short, collision-free path from an initial robot configuration to a desired configuration. It has to be fast to support real-time task-level robot programming. Accordingly, it has received much attention [8, 1, 2, 7, 9, 11], and there are now a number of implemented path planners based on a variety of approaches. Unfortunately, current planning techniques are still too slow to be effective, as they often require several minutes, if not hours of computation.

To remedy this situation, we develop a learning algorithm that uses past experience to increase future performance. More generally, the algorithm is actually a framework in which a slow but effective planner may be improved both cost-wise and capability-wise by a faster but less effective planner coupled with experience. The algorithm has been previously presented for stationary environments [4], and has been recently extended with an on-demand experience repair strategy to cope with incrementally changing environments [5].

In this paper, we present the full adaptive algorithm and provide a deeper analysis for the fundamental stationary case. The analysis is based on a stochastic model, which has also been used to prove the optimality of the on-demand experience repair strategy [5]. We use the model in this work to characterize the situations in which the learning algorithm is useful, and to what degree. We formalize the concept of improvability, and derive the conditions under which a planner can be improved within the framework. We also derive quantitative relationships between training effort, learning rate, planning cost, and planning capability. Finally, we use these analytic performance estimation tools to explain some experimental results. Although our presentation is in the context of motion planning, the algorithm and the analysis are extensible to more general learning. In particular, they may be applied to higher-level task planning or other domains in which experience is useful.

*This work has been performed at Sandia National Laboratories and supported by the U.S. Department of Energy under Contract DE-AC04-76DP00789.

2 Algorithm

Given an arbitrary work environment and an arbitrary task (u, w) of moving the robot from configuration point u to w, we assume that there are initially two path planners available: Reach and Solve. The Reach planner is required to be fast, symmetric, and only locally effective, i.e., it should have a good chance of success if u and w are close to each other. The Solve planner, on the other hand, is required to be much more globally effective than Reach, and hence can be very slow. It is the performance of this planner that we wish to improve.

In our learning scheme, we retain the global effectiveness of Solve by calling it whenever necessary, while reducing the overall time cost by calling Reach whenever possible. To utilize Reach fruitfully, we maintain a history of robot movements in the form of a connected graph, called the *experience graph* $G = (V, E)$ with vertices V and edges E. Set V is a sparse collection of subgoals that the robot can attain and use. Set E indicates the subgoal connections that the robot can follow through the application of Reach. Ideally, G is to be used by Reach to achieve most tasks without the help of Solve. We update G incrementally whenever Reach is incapable of achieving a task through G. In this situation, Solve is called. If Solve is also incapable of finding a solution, then we simply skip to the next task. Otherwise, the solution provided by Solve is abstracted (or compressed) into a chain consisting of a short sequence of subgoals that Reach can use later to achieve the same task. To learn from this experience, we simply generalize the use of the abstracted subgoals by augmenting G with the chain.

2.1 Environmental Assumptions

To allow fruitful learning, we require that the environmental change be incremental, i.e., occasional and localized. By occasional, we mean that the interval between workcell changes is large compared to the amount of time spent on each task. By localized, we mean that the workcell change involves only a few objects in a relatively small area of the workspace, and hence is not extensive. Both conditions are prevalent in applications and have their intuitive implications: Occasional implies that old experience may be useful for significant amount of time, and localized implies that old experience may have salvage value.

2.2 Formal Specification

Formally, the learning algorithm Adapt is shown in Figure 1. In the algorithm, u is the current robot configuration, and w is the next goal configuration. To access G, we maintain two pointers: $\hat{u}$ and $\hat{w}$, each of which points to a vertex of G that is known to be reachable with one call of Reach from u and w, respectively. The algorithm is based on two planners: $\mathcal{R}$ and $\mathcal{S}$, which are in turn based on Reach and Solve, respectively. Both $\mathcal{R}$ and $\mathcal{S}$ have task (u, w) as arguments, and graph G and a heuristic vertex ordering function h as parameters. For planner $\mathcal{R}$, we use $\mathcal{R}(\cdot)$ to denote the predicate that $\mathcal{R}$ is successful, and $\mathcal{R}[\cdot]$ to denote the path planned when $\mathcal{R}$ succeeds, and similarly for $\mathcal{S}$.

Planner $\mathcal{R}$ searches for ways to achieve task (u, w) using only Reach and G as guideline. The algorithm for $\mathcal{R}(\cdot)$ is:

1. Search the vertices of G in order according to heuristic h, and find a vertex v satisfying Reach(v, w).
2. If v exists, then set $\hat{w} \leftarrow v$, and return success.
3. Else return failure.

To generate $\mathcal{R}[\cdot]$, we require the success of $\mathcal{T}(\cdot)$, which guarantees that there is a connected sequence of vertices Γ in G from $\Gamma_1 = \hat{u}$ to $\Gamma_k = \hat{w}$ for some $k \geq 1$. Once $\mathcal{T}(\cdot)$ succeeds, a simple solution for $\mathcal{R}[\cdot]$ would be the concatenation of Reach$[\Gamma_j, \Gamma_{j+1}]$ for j going from 0 to k. However, we can also improve the quality of this solution locally as we shall see in Subsection 2.5.

Planner $\mathcal{S}$ sets $\hat{w}$ for the future augmentation of G. The algorithm for $\mathcal{S}(\cdot)$ is:

1. Set $\hat{w}$ to be the best vertex in G according to h.
2. Return(Solve$(\hat{w}, w)$).

```
Algorithm Adapt(R, S; T)
  u ← current position; v ← u; G ← ({v}, ∅);
  do forever
    w ← goal();
    if (not R(u, w; G, h)) then
      if (not S(u, w; G, h)) then continue;
      ρ ← Abstract(S[u, w; G, h]);
      G ← Augment(G, ρ);
      ŵ ← last vertex of ρ;
    endif
    if T(u, w; G, h) then Execute(R[u, w; G, h]); u ← w;
  enddo
end.
```

Figure 1: A speedup learning algorithm for path planning in incrementally-changing environment

To generate $\mathcal{S}[\cdot]$ once $\mathcal{S}(\cdot)$ returns success, simply output Solve$[\hat{w}, w]$.

2.3 Object-Attached Experience Abstraction

To abstract a solution path from v to w with $v \in G$, we assume that there is an efficient Abstract$(\cdot)$ function available that returns a short chain from v to $v' = w$, traversable by Reach. We assume that the size of the chains abstracted from solutions of $\mathcal{S}$ are all boundable by a constant. In practice, this is a reasonable assumption, since a typical task consists of only 3 smooth motions: departure, traversal, and approach.

To increase the flexibility of the subgoals, we require the vertices returned by Abstract$(\cdot)$ to be relative robot positions associated with nearby objects. That is, we remember each of them as an offset from some nearby object serving as a landmark. Under this object-attached experience abstraction scheme, we can adjust to any minor environmental change without expensive experience repair.

2.4 On-Demand Experience Repair

Of course, if the environment changes significantly, the validity of G will deteriorate. How much deterioration will G suffer depends on how drastically the environment changes. If the change is major and extensive, then it may be better to start over with no experience (G reinitialized), rather than to work with the old impaired experience. In the more interesting case where the change may be major (e.g., introducing a new object) but not extensive (e.g., the rest of the workcell is undisturbed), the right choice is not as clear. Thus, we introduce an on-demand repair scheme to retain those experiences that remain valid and useful.

In this scheme, we plan as if G is connected, until $\mathcal{R}(\cdot)$ succeeds and we actually need to produce a path. Then, to generate $\mathcal{R}[\cdot]$, we require the success of $\mathcal{T}(\cdot)$ to provide a connected sequence from $\hat{u}$ to $\hat{w}$. As $\mathcal{T}(\cdot)$ searches for and verifies such a sequence, it may come across invalid edges, which it simply deletes. If $\hat{u}$ is already connected to $\hat{w}$ in G, then no repair need take place. If, however, $\hat{u}$ and $\hat{w}$ do not belong to the same (connected) component due to the deterioration of G, then Solve is called to reestablish their connectivity. It is of course possible that connectivity cannot be reestablished due to the environmental change. In this case, the portion of G connected to $\hat{w}$ is deemed useless, and hence discarded. The procedure for $\mathcal{T}(\cdot)$ is:

1. While there exists a sequence Γ of vertices in G connecting $\hat{u} = \Gamma_1$ to $\hat{w} = \Gamma_k$ for some $k \geq 1$ do
   (a) If Reach(Γ_i, Γ_{i+1}) for all $1 \leq i < k$ then return success;
   (b) Else remove edge (Γ_i, Γ_{i+1}) with smallest i such that $\neg$Reach(Γ_i, Γ_{i+1}).
2. If Solve$(\hat{u}, \hat{w})$ then augment G with Abstract(Solve$[\hat{u}, \hat{w}]$); return success;

3. Else remove the (connected) component of $\hat{w}$ from G, and return failure.

2.5 Solution Quality and Redundancy

So far we have focused on task solvability but not solution quality. If solution quality is not important, then in $\mathcal{R}[\cdot]$, we can simply produce the solution of going through Γ with Reach. In this situation, the experience graph will always be a tree. However, if solution quality is important, then it may be worthwhile to locally optimize Γ by "cutting corners" whenever possible.

3 Analysis

To analyze our algorithm in more detail, we have developed an analytic model that is simple enough for probabilistic treatment, yet general enough to capture the key aspects of the learning process. This model has been used to analyze the algorithm under incrementally-changing environments, and in particular, the optimality of the on-demand experience repair strategy [5]. The development of the model is in the same spirit as that in developing search tree models for analyzing heuristics [12]. Using this model, we now further the analysis of our algorithm under stationary environments. (For this extended abstract, the proofs are in the Appendix.)

As in the framework of PAC-learning [10], we assume that the goals are drawn randomly and independently from a distribution. We do not require Solve to be complete; we do require that it have a success probability σ in solving a random task. We make the simplistic assumption that only Solve, Reach, and Abstract have costs, each being a constant. To normalize, let 1, r, and c be the respective costs of Solve, Reach, and Abstract. (Both r and c are typically $\ll 1$.) Let G_n be the experience graph G after Adapt has been trained with n tasks. We are interested in both the speedup that Adapt has over the plain iterations of Solve, and the capability of Adapt as it increases with training. Thus, we analyze the relationship between the following random variables:

A_n	The probability that Adapt will need to call Solve in solving task $n+1$, i.e., the probability that task $n+1$ will not be $\mathcal{R}$-reachable via G_n.
K_n	The number of times that Solve has been called after Adapt has been trained with n tasks.
L_n	The probability that a random task not $\mathcal{R}$-reachable via G_n will now be $\mathcal{R}$-reachable via G_{n+1}.

As pointed out in [4], the learning rate L_n and the expected learning rate $\mathbf{E}(L_n \mid A_n)$ are key quantities in determining the success of Adapt. To continue the analysis, we model the experience graph G_n by the following random tree T_n with the notion of utility μ defined.

Definition 1 *Let $\|X\|$ for any graph X be the number of edges in X. Let T_0 be a one-vertex tree consisting of only the root v_0 (home position). For $n > 0$, T_n is a random tree obtained from T_{n-1} by extending from a uniformly chosen vertex of T_{n-1} a branch ρ_n consisting of an alternating sequence of edges and vertices. The length (number of edges) of ρ_n is random variable $\|\rho\| \geq 1$ that is independent of n.*

Under model $\mathcal{M}$, the experience graph of Adapt *is $G_n = T_n$. The utility of any subgraph of T_n is the probability that the particular subgraph can be used to solve the next random task. For each non-root vertex v of T_n, utility $\mu(v)$ is an independent, identically distributed, nonnegative random variable with mean $\bar{\mu}$.*

Thus, each branch ρ in T_n corresponds to the abstracted solution of a call to Solve. $\mathcal{M}$ is designed to model Adapt without local optimization (Subsection 2.5), since the experience graph is always a tree.

We begin our analysis with the following results that express the planning costs of Adapt in terms of its failure probability A_n.

Lemma 1 *Suppose that* Adapt *has accrued N vertices during its training. Then on average under $\mathcal{M}$ and without calling* Solve *again, the probability that* Adapt *will succeed in solving the next random task is $1 - Q_N$*

where

$$Q_N \stackrel{\text{def}}{=} (1-\bar{\mu})^N, \tag{1}$$

and the cost of Adapt *in solving this task is* $(1-Q_N)r/\bar{\mu}$.

Theorem 2 *Suppose that* Adapt *has been trained with n tasks. Then under* $\mathcal{M}$, *if* Solve *is not called, the average cost* E_n *of* Adapt *to solve the next task is*

$$E_n = \frac{r}{\bar{\mu}}(1-\mathbf{E}A_n). \tag{2}$$

Theorem 3 *Under* $\mathcal{M}$, *the average cumulative cost* F_n *of* Adapt *after training with n tasks is*

$$F_n = rn/\bar{\mu} + (1+\sigma c - r/\bar{\mu})\mathbf{E}K_n, \tag{3}$$

or equivalently,

$$\Delta F_n \stackrel{\text{def}}{=} F_{n+1} - F_n = r/\bar{\mu} + (1+\sigma c - r/\bar{\mu})\mathbf{E}A_n. \tag{4}$$

The relationship between training effort n, experience utility μ, and failure probability A_n is derived in the following via the learning rate L_n.

Lemma 4 *Suppose that* Adapt *has an expected learning rate of* $\mathbf{E}(L_n \mid A_n) = \alpha A_n$ *for some positive* $\alpha \leq 1$. *Then the average* $\mathcal{R}$*-failure probability of* Adapt *after* $n > 0$ *tasks of training has an upper bound of*

$$\mathbf{E}A_n \leq \frac{1}{\alpha(n+1)} \tag{5}$$

for all $n \geq 0$.

With the lemma above, we can show that the reliance of Adapt on Solve is at most inversely proportional to the number of training tasks.

Theorem 5 *Under* $\mathcal{M}$,

$$(n+1)\mathbf{E}A_n \leq \frac{1}{\sigma \mathbf{E}\mu(\rho)}, \tag{6}$$

where

$$\mathbf{E}\mu(\rho) \stackrel{\text{def}}{=} \mathbf{E}(1-\bar{\mu})^{\|\rho\|} \tag{7}$$

denotes the average utility of a branch.

Using the following definition of improvability, we determine the conditions under which Solve is improvable and the amount of training required.

Definition 2 *Let* $\mathcal{A}$ *be a learning algorithm designed to improve* $\mathcal{A}'$. *We say that* $\mathcal{A}$ *can improve* $\mathcal{A}'$ *with failure probability* A *iff* $\mathcal{A}$ *can solve the same task as* $\mathcal{A}'$ *with failure probability at most* A, *while costing less on average. In particular, we say that* $\mathcal{A}$ *can effectively improve* $\mathcal{A}'$ *iff* $\mathcal{A}$ *can improve* $\mathcal{A}'$ *with failure probability 0.*

Theorem 6 *Under* $\mathcal{M}$, Adapt *can effectively improve* Solve *with sufficient training iff* $r < \bar{\mu}$. *An upper bound on* $n_{\min}$, *the minimum number of training tasks required, is*

$$n_{\min} \leq \frac{1}{\sigma \mathbf{E}\mu(\rho)}\left(1 - \frac{\sigma c}{1 - r/\bar{\mu}}\right). \tag{8}$$

If $r \geq \bar{\mu}$, *then* Adapt *can still improve* Solve *with some failure probability* $\mathbf{E}A_n \geq 1 - \bar{\mu}/r$, *provided that* Adapt *is not overtrained. An upper bound on* $n_{\max}$, *the maximum number of training tasks that* Adapt *should receive, is*

$$n_{\max} < \frac{1}{\sigma \mathbf{E}\mu(\rho)}\left(\frac{1}{1 - \bar{\mu}/r}\right) - 1. \tag{9}$$

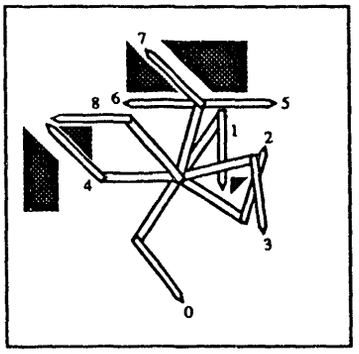


Figure 2: A planar 2-link robot environment

Corollary 7 *Suppose that* Solve *is not a complete planner, i.e.,* $\sigma < 1$. *Then under* $\mathcal{M}$, Adapt *can improve* Solve *both cost-wise and capability-wise iff there is an* n *such that* $\bar{\mu}/r \geq 1 - \mathbf{E}A_n > \sigma$. *In this case, both improvements can be achieved with the number of training tasks being*

$$n = \left\lfloor \frac{1}{\sigma(1-\sigma)\mathbf{E}\mu(\rho)} \right\rfloor. \tag{10}$$

Finally, we have the following asymptotic bound on the performance of Adapt during training.

Theorem 8 *Under* $\mathcal{M}$, *the ratio of the average cost of* Adapt *to that of* Solve *is bounded asymptotically by* $r/\bar{\mu}$ *as the number of training tasks approaches infinity. More globally, the behavior is*

$$\frac{F_n}{n} \leq \frac{r}{\bar{\mu}} + \left(1 + \sigma c - \frac{r}{\bar{\mu}}\right) \begin{cases} \ln(eA_0\alpha n)/(\alpha n) & \text{if } A_0\alpha n > 1; \\ A_0 & \text{otherwise,} \end{cases} \tag{11}$$

where $\alpha = \sigma\mathbf{E}\mu(\rho)$. *Accordingly, the maximum value that the ratio can attain at any* n *is at most*

$$F_n/n \leq r/\bar{\mu} + (1 + \sigma c - r/\bar{\mu})\, A_0. \tag{12}$$

4 Discussion

To gain more insight into our algorithm, we explain some experimental results with the theory developed in the previous section. Figure 2 shows a stationary 2-link planar robot environment in which Adapt is applied. In this experiment, there are 5 polygonal obstacles in the fixed workcell, and a goal set consisting of 9 preselected goal positions. Starting at home position 0, the robot is to go through a sequence of 100 goals randomly selected from the goal set. The result of this experiment is shown in Figure 3. In the left frame, the ratio of the cumulative planning cost of Adapt to that of Solve only is plotted against the task number n. The planning costs are averaged over 100 runs and are measured by the number of robot-to-obstacle distance evaluations, which is the dominating factor in the computing cost of each planner. In the right frame, the ratio is plotted against $(\ln(n+1))/(n+1)$ to show their asymptotic linear relationship, hinted by Theorem 8.

The experiment shows that Adapt is able to learn and speed up its performance relative Solve from 150% slower (ratio $\doteq$ 2.5) to (by extrapolation) 62% faster (ratio $\doteq$ 0.38). If we believe that the upper bound provided by Theorem 8 is also an asymptotic lower bound, then the plot implies that $r/\bar{\mu} \doteq 0.38$. From other empirical observation, we estimate that $r \doteq 0.1$, $c \doteq 1$, $\sigma \doteq 1$, and $A_0 \doteq 0.9$. Hence, $\bar{\mu} \doteq 0.26$. Since the branches are of lengths $\doteq 2$, we also estimate $\mathbf{E}\mu(\rho) \doteq 0.45$. To see how consistent these numbers are, we estimate the number of training tasks required by Adapt to have its cumulative cost first become less than that of Solve. Using the formula in Theorem 8, we have $\alpha \doteq 0.45$ and

$$\frac{\ln(eA_0\alpha n)}{eA_0\alpha n} \doteq \frac{1 + \sigma c - r/\bar{\mu}}{eA_0(1 - r/\bar{\mu})} \doteq 0.156. \tag{13}$$

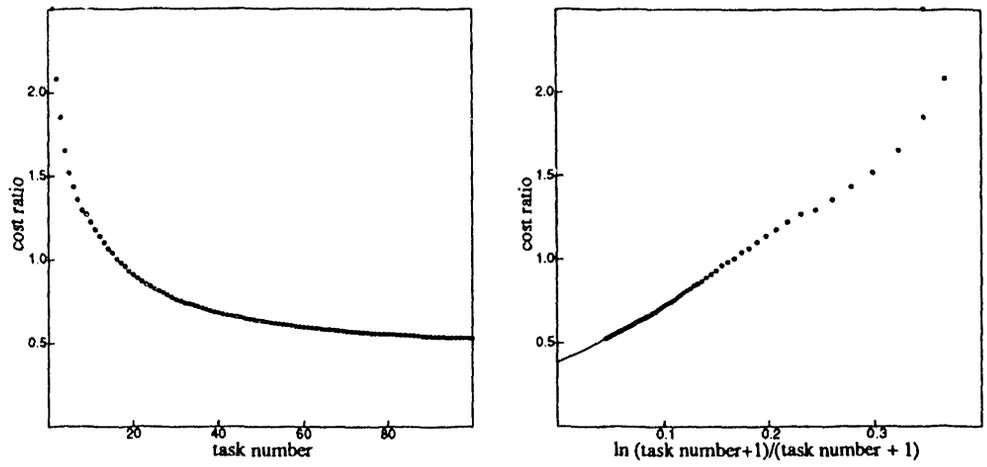


Figure 3: Time improvement of Adapt over Solve

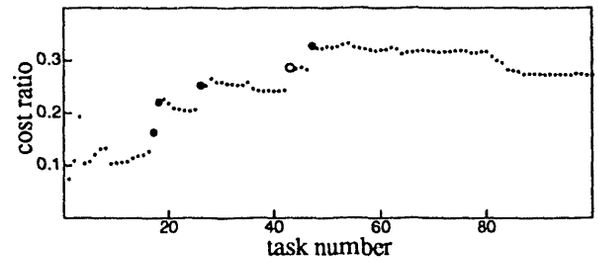


Figure 4: Time improvement of Adapt on a 3-d cask problem.

giving us $eA_0\alpha n \doteq 19$, or $n \doteq 17.2$, which is very close to the observed $n = 17$ in the plot.

We use our theory to explain another experiment performed previously to simulate a radiation survey environment [4, 6]. In this experiment, Adapt is applied on a 6-dof gantry robot working in a world with 4 obstacles: a $(16+2)$-sided polyhedral approximation of a cylindrical cask, two cask stands, and a floor. To make the problem more difficult, the joint limits of the robot are restricted so that there is not much room to maneuver. The goal positions are chosen randomly, and correspond to the robot end effector touching the cask surface in a prescribed orientation. The problem is sufficiently difficult that the original path planner, Solve, fails to accomplish 7 tasks out of a sequence of 100 random goals. In contrast, Adapt is able to accomplish all but 1 task during the exercise, thereby increasing the capability of the original planner. Moreover, Adapt calls Solve only 5 times, and the final graph learned by Adapt contains only 12 vertices. Figure 4 shows the actual time improvement. The ratio of the cumulative effort expended by Adapt to that expended by Solve only is plotted. The effort is measured by the number of robot-to-obstacle distance evaluations, which dominates the computational cost of each planner. The 5 large points indicate Adapt's calling of Solve, and the single white point indicates the only failure of Adapt. Initially, Adapt is able to plan without Solve because the tasks are relatively easy. Later, Adapt starts to learn as indicated by the jumps of the cost ratio. When the task number reaches 50, Adapt has basically "learned" the environment as shown by the gradual decline of the cost ratio.

Using the data, we estimate $\sigma \doteq 93\%$ because of the 7 failures; $\bar{\mu} \doteq 1 - 0.01^{1/11} \doteq 34\%$ because the 11 non-root vertices are able to cover 100 random tasks; $\mathbf{E}\mu(\rho) \doteq 1 - 0.01^{1/5} \doteq 80\%$ because only 5 branches are involved. Using Corollary 7, we then estimate $n \doteq 1/(0.93 \cdot 0.07 \cdot 0.8) \doteq 19.2$ to be the number of training tasks n required for Adapt to succeed in improving both the speed and the capability of Solve. This estimate means that Reach is already very powerful, and that roughly only 2 calls (#17 and #18 in the plot) to Solve are necessary for Adapt to catch up with Solve in task solving capability.

With Theorem 8, we can also estimate the limiting cost ratio. We estimate $A_0 \doteq 1/17 \doteq 6\%$ because Adapt first failed at task #17. We also estimate $c \doteq 0.1$ from empirical observation. Again, if we believe that the upper bound provided in the theorem is also a lower bound, then the maximum cost ratio is

$$r/\bar{\mu} + (1 + 0.1 - r/\bar{\mu})0.06 = 0.32 \tag{14}$$

from the plot, which implies that $r/\bar{\mu} \doteq 0.27$, which is incidentally very close to the cost ratio at the end of task #100. Consequently, we do not anticipate Adapt to do much better with more training.

5 Conclusion

We have presented a learning algorithm that can improve path planning. The algorithm adapts to its working environment by maintaining an experience graph with vertices corresponding to useful robot configurations. The algorithm is suitable for both stationary and incrementally-changing environments. It can both reduce time cost and increase task solving capability of existing planners. To gain insight into this algorithm for the stationary case, we have presented some theoretical analysis based on a simple, yet general stochastic model that quantifies experience utility. Using this model, we characterize the situations in which the adaptive planner is useful, and provide quantitative bounds to predict its behavior. The results are used to explain some experimental results in manipulator planning. Our algorithm and analysis are sufficiently general that they may also be applied to other planning domains where experience is useful.

References

[1] Barraquand, J. and Latombe, J., "A Monte-Carlo algorithm for path planning with many degrees of freedom," *Proc. of IEEE Int. Conf. on Robotics and Automation,* 1990, pp. 1712-1717.

[2] Chen, P.C., and Hwang, Y.K., "SANDROS: A Motion Planner with Performance Proportional to Task Difficulty," *Proc. of IEEE Int. Conf. on Robotics and Automation,* 1992.

[3] Chen, P.C., "Effective Path Planning through Task Restriction," Sandia Report SAND91-1964, 1992.

[4] Chen, P.C., "Improving Path Planning with Learning," *Machine Learning: Proc. of the Ninth Int. Conf.,* 1992.

[5] Chen, P.C., "Adaptive Path Planning for Incrementally-Changing Environments," submitted to *Tenth Int. Conf. on Machine Learning,* 1993.

[6] Harrigan, R.W., Sanders, T.L., "A Robotic System to Conduct Radiation and Contamination Surveys on Nuclear Waste Transport Casks," Sandia Report SAND89-0017, 1990.

[7] Kondo, K., "Motion Planning with Six Degrees of Freedom by Multistrategic Bidirectional Heuristic Free-Space Enumeration", *IEEE Tran. on Robotics and Automation,* vol. 7, no. 3, pp. 267-277, June 1991.

[8] Latombe, J., *Robot Motion Planning,* Kluwer Academic Publishers, 1991.

[9] Lozano-Pérez, T., "A Simple Motion-Planning Algorithm for General Robot Manipulators," *IEEE J. of Robotics and Automation,* vol. RA-3, no. 3, pp. 224-238, June 1987.

[10] Natarajan, B.K., *Machine Learning: A Theoretical Approach,* Morgan Kaufmann, 1991.

[11] Paden, B., Mees, A. and Fisher, M., "Path Planning Using a Jacobian-Based Freespace Generation Algorithm," *Proc. of IEEE Int. Conf. on Robotics and Automation,* pp. 1732-1737, 1989.

[12] Pearl, J., *Heuristics: Intelligent Search Strategies for Computer Problem Solving,* Addison-Wesley, 1984.

A Proofs

Proof (Lemma 1) The first part of the lemma is obvious since the average probability that all of the learned experience will not be applicable in solving the next task is

$$\mathbf{E}\prod_{i=1}^{N}(1-\mu(v_i)) = Q_N, \tag{15}$$

where v_i denotes the i^{th} node of G. The second part of the lemma follows immediately from the fact that the required cost is $r\sum_{i=0}^{N-1} Q_i$ with Q_i being the average probability that the applicability of node v_{i+1} will be queried through Reach. ∎

Proof (Theorem 2) Fix N, the number of vertices in G. By Lemma 1, the average cost of Adapt in solving task $n+1$ without calling Solve is $r(1-Q_N)/\bar{\mu}$. Averaging over all possibilities for N yields the desired result since

$$\mathbf{E}A_n = \mathbf{E}\,\mathbf{E}(A_n \mid N) = \mathbf{E}Q_N. \tag{16}$$

∎

Proof (Theorem 3) The two equations are equivalent since $\mathbf{E}K_n = \sum_{j<n}\mathbf{E}A_j$ [4, Theorem 1]. The second equation follows from the fact that in addition to E_n, a cost of $(1+\sigma c)\mathbf{E}A_n$ will also be required to to call Solve with probability $\mathbf{E}A_n$ and Abstract with probability $\sigma\mathbf{E}A_n$. ∎

Proof (Lemma 4) It is known [4, Theorem 8] that for $n > 0$,

$$\mathbf{E}A_n \le (\alpha(n+1))^{-1}\exp\left(\frac{(0.52-\ln n)}{2n}\right), \tag{17}$$

which implies the desired upper bound for $n \ge 2$. For $n = 0$, $\mathbf{E}A_0 \le 1 \le 1/\alpha$. For $n = 1$, $\mathbf{E}A_1 = \mathbf{E}A_0(1-\alpha A_0)$ has maximum value $1/(4\alpha)$, which is less than the desired upper bound of $1/(2\alpha)$. ∎

Proof (Theorem 5) In solving task $n+1$, Adapt will need to call Solve with probability A_n. In this case, a branch ρ will be acquired with probability σ. Hence, with probability $A_n\sigma$, the failure probability A_{n+1} will be $A_n\prod_{i=1}^{\|\rho\|}(1-\mu(v_i))$, where v_i is the i^{th} vertex of ρ. Thus, the expected learning rate is

$$\begin{aligned}
\mathbf{E}(L_n \mid A_n) &= A_n\sigma(1-\mathbf{E}\prod_{i=1}^{\|\rho\|}(1-\mu(v_i))) && (18)\\
&= A_n\sigma(1-\mathbf{E}\mathbf{E}(\prod_{i=1}^{\|\rho\|}(1-\mu(v_i)) \mid \|\rho\|)) && (19)\\
&= A_n\sigma(1-\mathbf{E}(1-\bar{\mu})^{\|\rho\|}) && (20)\\
&= A_n\sigma\mathbf{E}\mu(\rho). && (21)
\end{aligned}$$

The theorem now follows from the previous lemma with $\alpha = \sigma\mathbf{E}\mu(\rho)$. ∎

Proof (Theorem 6) For Adapt to effectively improve Solve, we must have $\Delta F_n < 1$, which implies that

$$\mathbf{E}A_n < (1-r/\bar{\mu})/(1+\sigma c - r/\bar{\mu}) \tag{22}$$

according to Theorem 3. To attain this bound, it suffices to have

$$(n+1)\sigma\mathbf{E}\mu(\rho) > (1+\sigma c - r/\bar{\mu})/(1-r/\bar{\mu}), \tag{23}$$

which yields the upper bound on $n_{\min}$.

If $r \geq \bar{\mu}$, then Adapt can still improve Solve by not calling Solve after a certain amount of training. For there to be improvement, we must have $E_n < 1$, which implies that

$$\mathbf{E}A_n > 1 - \bar{\mu}/r. \tag{24}$$

To violate this bound through overtraining, it suffices to have

$$(n+1)\sigma \mathbf{E}\mu(\rho) \geq 1/(1 - \bar{\mu}/r), \tag{25}$$

which yields the upper bound on $n_{\max}$. ∎

Proof (Corollary 7) For Adapt to become more capable than Solve, we must have $1 - \mathbf{E}A_n > \sigma$. On the other hand, Adapt can only improve Solve with success probability $1 - \mathbf{E}A_n \leq \bar{\mu}/r$. Hence, we must have $\bar{\mu}/r \geq 1 - \mathbf{E}A_n > \sigma$ for some n. Conversely, if such n exists, then to achieve $A_n < 1 - \sigma$, it suffices to have

$$\frac{1}{\sigma \mathbf{E}\mu(\rho)(n+1)} < 1 - \sigma, \tag{26}$$

which implies the desired result. ∎

Proof (Theorem 8) From Theorem 3, it suffices to prove that

$$\mathbf{E}K_n \leq \begin{cases} \ln(eA_0\alpha n)/\alpha & \text{if } A_0\alpha n > 1; \\ A_0 n & \text{otherwise.} \end{cases} \tag{27}$$

Since $\mathbf{E}K_n = \sum_{j<n} \mathbf{E}A_j$, and $\mathbf{E}A_n \leq \min(A_0, (\alpha(n+1))^{-1})$, we must have

$$\mathbf{E}K_n \leq A_0 x + (H_n - H_x)/\alpha, \tag{28}$$

for all positive integers $x \leq n$. Since $H_n - H_x \leq \ln(n/x)$, we may extend the domain of x to the reals and obtain

$$\mathbf{E}K_n \leq A_0 x + \ln(n/x)/\alpha, \tag{29}$$

which yields the theorem when minimized at $x = \min(n, 1/(\alpha A_0))$. ∎

DISCLAIMER

DATE
FILMED
9/23/93

END