SANDIA REPORT

SAND95-0722 • UC-~~510~~
Unlimited Release 405
Printed April 1995

Learning to Improve Path Planning Performance

Pang C. Chen

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-94AL85000

Approved for public release; distribution is unlimited.

SF2900Q(8-81)

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED
CH

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
PO Box 62
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from
National Technical Information Service
US Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A03
Microfiche copy: A01

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

SAND95-0722
Unlimited Release
Printed April 1995

Distribution
Category UC-405

LEARNING TO IMPROVE PATH PLANNING PERFORMANCE

Pang C. Chen
Sandia National Laboratories
Albuquerque, NM 87185-0951

Abstract

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. 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 present and analyze a learning algorithm that uses past experience to increase 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 robot configurations is learned to support faster planning. More generally, the algorithm provides a speedup-learning framework in which a slow but capable planner may be improved both cost-wise and capability-wise by a faster but less capable planner coupled with experience. The basic algorithm is suitable for stationary environments, and can be extended to accommodate changing environments with on-demand experience repair and object-attached experience abstraction. To analyze the algorithm, we characterize the situations in which the adaptive planner is useful, provide quantitative bounds to predict its behavior, and confirm our theoretical results with experiments in path planning of manipulators. Our algorithm and analysis are sufficiently general that they may also be applied to other planning domains in which experience is useful.

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

GH

1 Introduction

One of the most important problems in robotics is path planning, which in known environments refers to finding a short, collision-free path from an initial robot configuration to a desired configuration. Path planning algorithms have to be fast (ideally within seconds) to support real-time task-level robot programming. Accordingly, path planning has received much attention [21, 17] and there are now a number of implemented path planners based on a variety of approaches. However, the practicality of these planners in general has been hampered by their time-consuming search, as they often require several minutes, if not hours of computation.

To improve the performance of these planners and hence increase their practical value, we present a learning algorithm that uses past experience to increase future performance. Our work is motivated by the observation that robots often perform multiple tasks in virtually the same environment. In such environments, the total planning time can be amortized and significantly reduced by reusing the computation results for one task to plan for another. One example is Sandia National Laboratories' Remote Radiation Survey and Analysis (RRSAS) project, which is to automatically inspect nuclear waste transport casks for radiation using a robot [16]. Without hitting any obstacle, the robot is to maneuver through the workspace and sample a set of randomly chosen points on the cask. For this series of tasks, learning is possible because the workspace is unchanged during inspection, therefore past experience is useful. Learning is also feasible for this problem because there are only a small number of different kinds of movements that the robot needs to learn.

Thus, we make the underlying assumption that similar tasks are to be performed repeatedly before the robot environment is changed. To perform each task in this stationary environment, our algorithm primarily uses a fast but necessarily incomplete planner that responds quickly and can solve simple tasks. For difficult tasks, however, the algorithm relies on a more complete but slower planner to provide solutions. The algorithm learns from these solutions, building an evolving sparse network of useful robot configurations that guides and supports fast planning. More generally, the algorithm provides a speedup-learning framework in which a slow but capable planner may be improved both cost-wise and capability-wise by a faster but less capable planner coupled with experience.

We demonstrate the effectiveness of the algorithm via both mathematical analysis and experimentation. Our focus is on the fundamental algorithmic behavior of learning as opposed to other equally important issues of knowledge representation, solution abstraction, and implementation. Thus, we present the algorithm in its abstract form so that we may provide an in-depth theoretical analysis for understanding its behavior. Our analysis begins with some general quantitative relationships governing the learning process, followed by a specific case analysis illustrating these results. To achieve predictive power while preserving some generality, we next study the algorithm under models with additional simplifying assumptions. Using these models, we derive global quantitative bounds on planning cost and capability in terms of training time. We show that the reliance of the improved planner on the original slow planner is at most inversely proportional to the training time. We also characterize the situations in which learning is useful and prescribe the amount of training required. Finally, in our empirical evaluation of the algorithm, we validate our theory and use it to gain insight into several experimental results. Not only can we explain the observed data using our theory, but we can also use it to predict unobservable quantities such as the maximum achievable speedup.

After studying our algorithm for the fundamental stationary case, we next extend it to handle incrementally changing environments. In this more general environment, we assume that for

each robot task, the obstacles are stationary, but may slowly change their configuration or shape over the course of the robot performing many tasks. One example application is manufacturing of evolving products in which the design changes made to a product will cause incremental changes to the robot environment. Another example is waste-site remediation in which wastes are typically removed one by one, resulting in a slowly changing environment. Our algorithmic extension consists of two experience manipulation schemes: For minor environmental change, we use an object-attached experience abstraction scheme to increase the flexibility of the learned experience; for major environmental change, we use an on-demand experience repair scheme to retain those experiences that remain valid and useful. With these modifications in place, we show that the learning algorithm is indeed able to adapt to its working environment, provided that the frequency of change is sufficiently low.

2 Related Work

As mentioned in the introduction, a large amount of research has been done on robot path planning, most of which deals with solving one-time problems in stationary environments [2, 3, 6, 15, 20, 22, 27]. Most implemented path planners have been developed for mobile robots and manipulators with a few degrees of freedom (dof). There are some that are designed for many dof manipulators based on random [2] (Brownian motion), sequential [15] (backtracking with virtual obstacles), or parallel [3] (genetic optimization) search. For mobile robots, there is also some work on solving one-time problems in time-varying environments that contain moving obstacles with known trajectories [12, 19, 29]. All of these planners, however, typically require minutes of computation for mobile robots, and tens of minutes for 6 dof manipulators. Further, little work has been done for changing environments [1] in which movable obstacles remain relatively stationary during sequences of tasks, as opposed to time-varying environments with constantly moving obstacles.

For solving several problems in stationary environments, there are a few other path planners that incorporate learning: some [14, 26] take a higher-level, reasoning approach, while others [18, 25] take a lower-level, memory-based approach similar to ours. Learning can be done incrementally, or in phases which some consider as preprocessing [18]. To decrease the effective cost of solving each problem, all of these works maintain a network (roadmap) of useful robot configurations (landmarks) and employ some sort of a local planner for moving through the network. Algorithmically, there are some differences between ours and that of other memory-based approach [18, 25]. First, we assume and use the same distribution of tasks (problems) for both training and subsequent problem-solving. In other works [18, 25], a uniform problem distribution is used for training. Second, we assume the existence of a fairly reliable, albeit slow, global planner to act as a teacher, whereas they do not. Thus, while their algorithms may be more general, they may also require more training time to compensate for the lack of solutions when local planning fails.

The work presented here is the culmination of three years of research [5, 8, 9]. The algorithm for stationary environments is initially presented [5] with some general but preliminary analysis on the learning process. Later, the algorithm is extended to cope with incrementally changing environments [8]. Most recently, a deeper analysis for the fundamental stationary case is developed [9]. Overall, the most significant difference between all of the aforementioned work and ours is that we aim to provide a theoretical foundation for algorithm analysis to: 1) better understand and predict our experimental results; and 2) suggest similar analysis techniques that others may apply to better understand their algorithms.

3 Algorithmic Framework

Given an arbitrary but fixed environment, let task (u, w) be defined as finding a collision-free path to move the robot from configuration point u to w . We assume that there are initially two path planners available: **fast** and **slow**. Both return true (1) if successful, false (0) otherwise. The **fast** 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. Any greedy hill-climbing method using a potential field [2] or sliding [13] approach should be sufficient to implement **fast**. The **slow** planner, on the other hand, is required to be much more globally effective than **fast**, and hence may be very slow. It is the performance of this planner that we wish to improve with our speedup learning algorithm. Notice that this 'planner' can even be the human operator himself.

In our learning algorithm, we retain the global effectiveness of **slow** by calling it whenever necessary, while reducing the overall time cost by calling **fast** whenever possible. We plan paths for arbitrarily shaped robots or arbitrarily jointed manipulators by planning for a point robot in the corresponding configuration space (C-space). To utilize **fast** fruitfully, we remember significant intermediate robot configurations learned from the solution paths of **slow**. These subgoals (landmarks) represent fully specified robot configurations and are stored in memory V , with connecting edges E (indicating successes of **fast**) maintained so that complete solution paths may be regenerated through applications of **fast**. The subgoals V can be thought of as 'trail-markers' in that each marker can be traced to one another through the trails E . We call the connected network of trail-markers the experience graph $G = (V, E)$. (In contrast, the graphs constructed by other algorithms [18, 25] are not necessarily connected since the availability of planner **slow** is not assumed.) Ideally, G is to be used by **fast** to achieve most tasks without the help of **slow**. If **fast** is incapable of achieving a task through G , **slow** is called. If **slow** is also incapable of finding a solution, then we simply skip to the next task. Otherwise, we learn from the solution of **slow** by abstracting (or compressing) it into a chain consisting of a short sequence of intermediate robot configurations that **fast** can use later to achieve the same or similar tasks.

Incidentally, instead of representing the trail-markers as fully specified robot configurations, we can try to be more sophisticated and remember a more general subgoal representing subspaces of configurations. With a more general subgoal representation, the experience graph can be more compact and powerful in solving new tasks. However, taking this research direction requires us to delve into deeper knowledge representation issues instead of focusing on the basic learning process behavior. Hence, we restrict our subgoals to specific configurations to gain simplicity, which allows us to develop a more rigorous understanding of the learning processes within the framework, and hopefully will provide further insight into the more sophisticated ones.

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 **fast** from u and w , respectively. We may view these pointers as tethers. The algorithm is based on two planners: **Fast** and **Slow**, which are in turn based on **fast** and **slow**, respectively. Both **Fast** and **Slow** have task (u, w) as arguments, and graph G and a heuristic vertex ordering function h as parameters. Planners **Fast** and **Slow** attempt to achieve (u, w) using G as guideline. Since in stationary environments, G forms a connected component and the tether from u to $\hat{u}$ stays valid, the planners only need to check the reachability of w from a known reachable configuration v of G . For planner **Fast**, we use **Fast**($\cdot$) to denote the predicate that **Fast** is successful, and **Fast**[$\cdot$] to denote the path found when **Fast** succeeds, and similarly for **Slow**.

The algorithm for **Fast**($\cdot$) is simple: Search the vertices of G in order according to heuristic h ,

```

Algorithm Adapt(Fast, Slow)
   $u \leftarrow$  current position;  $v \leftarrow u$ ;  $G \leftarrow (\{v\}, \emptyset)$ ;
  do forever
     $w \leftarrow$  goal;
    if (not Fast( $u, w; G, h$ )) then
      if (not Slow( $u, w; G, h$ )) then continue;
       $\rho \leftarrow$  Abstract(Slow( $u, w; G, h$ ));
       $G \leftarrow$  Learn( $G, \rho$ );
    endif
    execute(Fast( $u, w; G, h$ ));  $u \leftarrow w$ ;
  enddo
end.

```

Figure 1: A learning algorithm for improving path planning.

and find a vertex v satisfying **fast**(v, w). If v exists, then set $\hat{w} \leftarrow v$, and return success; else return failure. To generate **Fast**[$\cdot$] once **Fast**($\cdot$) succeeds, first trace a shortest sequence of vertices Γ in G from $\hat{w}$ back to $\hat{u}$, so that $\Gamma_1 = \hat{u}$, and $\Gamma_k = \hat{w}$ for some $k \geq 1$. Then, set $\Gamma_0 = u$, $\Gamma_{k+1} = w$, and output **fast**[Γ_j, Γ_{j+1}] for j going from 0 to k .

The algorithm for **Slow**($\cdot$) is even simpler: Call **slow**(v, w) with v being the best vertex in G according to h , and set $\hat{w} \leftarrow w$. To generate **Slow**[$\cdot$] once **Slow**($\cdot$) succeeds, simply output **slow**[v, w].

The two nested if-statements present **Adapt** as essentially **Fast** backed up by **Slow**. Learning occurs when **Fast** fails but **Slow** succeeds. In this case, we apply **Abstract** to condense the solution path **slow**[v, w] into a short chain ρ of trail-markers with each edge traversable by **fast**. We do not specify how **Abstract** is to be implemented, only that it return a short chain efficiently. In practice, this is a reasonable assumption, since a typical task consists of only 3 smooth motions: departure, traversal, and approach. Moreover, the abstraction can be implemented efficiently by locating the markers with binary search on a discretized solution path. After digesting work experience **slow**[v, w] into ρ , we next incorporate it into our repertoire G to achieve incremental learning. Here, there are many ways to implement **Learn**, ranging from connecting all feasible edges between ρ and G to simply connect ρ to G at v . The tradeoff is between time cost and solution quality. The more edges we attempt to introduce, the more time we will take, but the more choices of solution paths we will have. We do not specify how **Learn** is to be implemented, just that it needs to augment G with enough edges of ρ to ensure a solution path for reaching w if it were to be requested again.

We illustrate the learning algorithm with a simple example designed to capture the key aspects of the cask inspection problem. Consider a point robot in a two-dimensional workspace with an open disk in the center as obstacle. Let the goal points be uniformly distributed on the boundary of the disk, with the robot positioned initially at one of these points. Let **fast** implement a go-straight procedure, with **fast**(u, w) returning success iff w is visible from u , and **fast**[u, w] returning the line segment $\overline{uw}$. Let **slow** implement a greedy 2-step go-straight procedure, with **slow**(u, w) returning success iff there is a point v visible from both u and w , and **slow**[u, w] returning the shortest path from u to w through such v . To complete the specification, let the heuristic used in **Fast** and **Slow** be $h = h_1$, with h_1 ordering the vertices of G according to the distance to w , starting with the closest point first. We call the above specification $\mathcal{E}_1$.

Figure 2 illustrates **Adapt** with a series of snapshots. Frame (1) shows the initial setting with

Figure 2: Snapshots of Adapt.

the robot at position u , and G initialized to the single vertex v_1 . The first goal indicated by w is shown in Frame (2). Since **Fast** is unable to plan using only **fast** and G , **Adapt** then calls **Slow**. Using h , **Slow** chooses to extend from v_1 to w , since v_1 is the only vertex in G . The path produced by $\text{slow}(v_1, w)$ consists of the line segments $\overline{v_1 v_2}$ and $\overline{v_2 v_3}$ in Frame (3). This path is then abstracted into the chain connecting v_1 to v_2 and v_2 to v_3 . The result of augmenting G is that G now becomes the 3-vertex chain. Using this augmented G , **Fast** is now able to produce a path from u to w , which consists of the segments $\overline{uv_1}$, $\overline{v_1 v_2}$, $\overline{v_2 v_3}$, and $\overline{v_3 w}$, with $\overline{uv_1}$ and $\overline{v_3 w}$ being null segments. With the first task accomplished, the next task is shown in Frame (4). This time, **Fast** is able to solve the task without the help of **Slow**. Using h , it first picks v_2 to test for $\text{fast}(v_2, w)$. Accordingly, **fast** succeeds; hence, **Fast** sets $\hat{w} = v_2$ and produces the path with segments $\overline{uv_3}$, $\overline{v_3 v_2}$, and $\overline{v_2 w}$. The result is shown in Frame (5), with the next task indicated. This time, **Fast** is not able to accomplish the task, so **Slow** is called. Using h , **Slow** chooses to extend from v_1 , since it is the closest point to w . The resulting path produced by $\text{slow}(v_1, w)$ is shown in Frame (6). The chain produced connects v_1 to v_4 , and v_4 to v_5 . After G is augmented with this chain, **Fast** is able to solve the third task by producing the segments $\overline{uv_2}$, $\overline{v_2 v_1}$, $\overline{v_1 v_4}$, $\overline{v_4 v_5}$, and $\overline{v_5 w}$.

4 General Analysis

In this section, we provide some general analysis to better understand the performance of **Adapt**. A specific case analysis follows in the next section to illustrate the general results derived here. The techniques developed here should also be useful in analyzing other types of probabilistic learning.

In studying the speedup-learning framework provided by **Adapt**, two performance measures are of interest: efficiency and capability. To quantify, we assume that the problems are drawn randomly and independently from a distribution (as in PAC-learning [24]) on some configuration space (C-space) S . We do not require **slow** to be complete; we do require that it have a success probability σ in solving a random task. We assume that only **slow**, **fast**, and **Learn** have costs, each being a constant. (The cost of **Abstract** can be absorbed into the cost of **Learn**.) To normalize, let

Table 1: Random variables of interest.

A_n	The probability that Adapt will need to call slow in solving problem $n + 1$, i.e., $1 - \mathbf{E}(\text{Fast}(u, w; G_n, h))$, the probability that a random goal w will not be Fast -reachable via G_n .
I_n	0-1 variable indicating the invocation of Slow when w_{n+1} is not Fast -reachable.
L_n	The probability that a random goal w not Fast -reachable via G_n will now be Fast -reachable via G_{n+1} , assuming that $I_n = 1$.
K_n	The number of times that Slow has been called at the end of the n^{th} loop.
E_n	The cost of Fast in solving problem $(n + 1)$.
F_n	The cumulative cost of Adapt after n steps of training.

1, r , and c be the respective costs of **slow**, **fast**, and **Learn**. (Both r and c are typically $\ll 1$.) We use subscript n on a program variable to denote its value at the end of the n^{th} loop. Thus, G_n denotes the experience graph G after **Adapt** has been trained with n problems. We are interested in both the speedup that **Adapt** has over the plain iterations of **slow**, and the capability of **Adapt** as it increases with training. We are also interested in the performance of **Fast**, which is **Adapt** without the backup of **Slow** after some training. Hence, we analyze the relationships between the random variables in Table 1. These variables are important in characterizing the performance of **Adapt**. In particular, how fast A_n , the failure probability of **Fast** after n steps of training, goes to zero determines how well learning takes place.

To analyze these random variables, we use standard techniques in conditional probability theory [30]. Let superscript (n) on an operator denote the conditional operator given A_n . Table 2 summarizes the basic relationships between the random variables (Theorems 1,2,3,4) as well as the major results on estimating the failure probability A_n (Theorems 7,9,10). With respect to **Adapt**, Theorems 1 and 2 measure time cost; Theorem 3 measures space cost; and Theorem 4 measures capability. Since all these measures depend critically on A_n , we need to analyze A_n carefully. Our analysis is based on the learning rate L_n , which is a key quantity governing the learning process. The expected learning rate $\mathbf{E}(L_n | A_n) = \mathbf{E}^{(n)}L_n$ is also important in determining the success of **Adapt**. Thus, Theorems 7 and 9 explore the consequences when the expected learning rate has a lower bound, while Theorem 10 explores the same when the learning rate has an upper bound.

We now begin our analysis in detail. Readers uninterested in these details may skip to the next section.

Theorem 1 *The average number of calls that **Adapt** will make to **Slow** after n steps of training is*

$$\mathbf{E}K_n = \sum_{0 \leq j < n} \mathbf{E}A_j.$$

Proof This is an immediate result of $K_n = \sum_{0 \leq j < n} I_j$, and $\mathbf{E}I_j = \mathbf{E}\mathbf{E}^{(j)}I_j = \mathbf{E}A_j$. ■

Theorem 2 *The average planning cost of **Adapt** per problem after n steps of training is*

$$\mathbf{E}\Delta F_n \stackrel{\text{def}}{=} \mathbf{E}(F_{n+1} - F_n) = (1 + \sigma c)\mathbf{E}A_n + \mathbf{E}E_n. \quad (1)$$

*Consequently, the average cumulative cost of **Adapt** after n steps of training is*

$$\mathbf{E}F_n = (1 + \sigma c)\mathbf{E}K_n + \sum_{0 \leq j < n} \mathbf{E}E_j. \quad (2)$$

Table 2: Summary of general analysis.

Thm	Condition	Implication
1		$\mathbf{E}K_n = \sum_{0 \leq j < n} \mathbf{E}A_j$
2		$\mathbf{E}F_n = (1 + \sigma c)\mathbf{E}K_n + \sum_{0 \leq j < n} \mathbf{E}E_j$
3		$\mathbf{E}\ V_n\ = \Theta(\mathbf{E}K_n)$
4	$\mathbf{E}A_n \leq \epsilon \delta$	$\Pr(A_n \geq \epsilon) \leq \delta$
7	$\mathbf{E}^{(n)}L_n \geq \alpha$ $\bar{\alpha} \stackrel{\text{def}}{=} 1 - \alpha < 1$	$\mathbf{E}A_n \leq A_0 \bar{\alpha}^n$ $\mathbf{E}K_n \leq A_0(1 - \bar{\alpha}^n)/\alpha \leq A_0/\alpha$
9	$\mathbf{E}^{(n)}L_n \geq \alpha A_n^r$ $r > 0, \alpha > 0$	$\mathbf{E}A_n \leq (\alpha r(n+1))^{-1/r} \exp\left(\frac{0.52 - \ln n}{2rn}\right)$
10	$L_n \leq \beta A_n$ $0 < \beta < 1$	$\left(\frac{1-\beta}{\beta}\right) \ln(A_0/\mathbf{E}A_n) \leq \mathbf{E}K_n \leq \ln_{1-\beta}(\mathbf{E}A_n/A_0)$

Proof The cost for ΔF_n is obvious since in addition to E_n , a cost of $(1 + \sigma c)A_n$ is required to call **slow** with probability A_n and **Learn** with probability σA_n . The second equation follows immediately from Theorem 1. $\blacksquare$

Theorem 3 *The average amount of memory required by **Adapt** as measured by $\|V_n\|$ after n steps of training is $\Theta(\mathbf{E}K_n)$.*

Proof Obvious since by our initial assumption on the abstraction process, each call to **Slow** generates at least one and at most a constant number of vertices. $\blacksquare$

Theorem 4 *Suppose that **Adapt** is to be trained with random instances until the probability of further **Fast**-failures becomes less than ϵ with confidence at least $1 - \delta$, i.e., until $\Pr(A_n \geq \epsilon) \leq \delta$. Then the number of training steps required is at most n_t , with n_t being the smallest n satisfying $\mathbf{E}A_n \leq \epsilon \delta$.*

Proof If $\mathbf{E}A_n \leq \epsilon \delta$, then $\Pr(A_n \geq \epsilon) \leq \delta$ because $\mathbf{E}A_n \geq \Pr(A_n \geq \epsilon)\epsilon$. $\blacksquare$

To interpret this result, suppose that we would be satisfied if **Adapt** can solve $90\% = 1 - \epsilon$ of the problems using only **Fast**. Then with n_t steps of training, where n_t is the smallest n satisfying $\mathbf{E}A_n \leq 0.001$, we can guarantee with $1 - \delta = 99\%$ confidence that the trained **Adapt** will be adequate.

The four theorems above show that the expected performance of **Adapt** depends critically on the behavior of $\mathbf{E}A_n$, whether it is terms of time, space, or accuracy. We now examine the behavior of A_n more closely.

Lemma 5 $A_{n+1} = A_n - L_n I_n$.

Proof If $I_n = 0$, then no change will be made to the experience graph, implying $G_{n+1} = G_n$, and $A_{n+1} = A_n$. If $I_n = 1$, then the graph will be augmented to solve more problems whose probability is measured by L_n . ■

Lemma 6 For $j \geq 1$, the j^{th} moment of A_n under Adapt satisfies

$$\begin{aligned} \mathbf{E}A_{n+1}^j &= \mathbf{E}A_n^j - \mathbf{E}A_n(A_n^j - \mathbf{E}^{(n)}(A_n - L_n)^j) \\ &\leq \mathbf{E}A_n^j - \mathbf{E}A_n(\mathbf{E}^{(n)}L_n)^j. \end{aligned}$$

In particular, for $j = 1$, $\mathbf{E}A_{n+1} = \mathbf{E}A_n(1 - \mathbf{E}^{(n)}L_n)$.

Proof To derive the equation, it suffices to prove that

$$\mathbf{E}^{(n)}A_{n+1}^j = A_n^j - A_n(A_n^j - \mathbf{E}^{(n)}(A_n - L_n)^j),$$

which follows from the fact $\mathbf{E}^{(n)}I_n = A_n$ so that during the $(n+1)^{\text{th}}$ loop, A_{n+1} will remain unchanged from A_n with probability $1 - A_n$, and decrease by L_n from A_n with probability A_n .

To derive the inequality, observe that

$$X^j + Y^j \leq (X + Y)^j$$

for $X = A_n - L_n \geq 0$, $Y = L_n \geq 0$, and $j \geq 1$. Hence, we have

$$\begin{aligned} \mathbf{E}A_{n+1}^j &\leq \mathbf{E}A_n^j - \mathbf{E}A_n(A_n^j - \mathbf{E}^{(n)}(A_n^j - L_n^j)) \\ &= \mathbf{E}A_n^j - \mathbf{E}A_n(\mathbf{E}^{(n)}L_n^j), \end{aligned}$$

yielding the result. ■

Theorem 7 Suppose that Adapt has at least an expected learning rate of $\mathbf{E}^{(n)}L_n \geq \alpha = 1 - \bar{\alpha}$, for some positive α . Then $\mathbf{E}A_n \leq A_0\bar{\alpha}^n$ and

$$\mathbf{E}K_n \leq A_0(1 - \bar{\alpha}^n)/\alpha \leq A_0/\alpha.$$

Proof Obvious from unfolding the first moment recurrence of Lemma 6. ■

Lemma 8 Any positive decreasing sequence $\{a_n\}_{n \geq 0}$ with $a_0 \leq 1$ that satisfies the inequality

$$a_n \leq a_{n-1}(1 - \alpha a_{n-1}^r) \tag{3}$$

for some positive α and r , has an upper bound of

$$a_n \leq (\alpha r(n+1))^{-1/r} \exp\left(\frac{0.52 - \ln n}{2rn}\right), \tag{4}$$

for $n > 0$.

Proof Let $a_n = (\alpha r(n+1))^{-1/r} e^{b_n}$. Then Inequality (3) becomes

$$e^{b_n} \leq (1 + 1/n)^{1/r} e^{b_{n-1}} \left(1 - e^{r b_{n-1}} / r n\right).$$

Taking the logarithm of both sides and Taylor-expands the RHS yields

$$b_n \leq b_{n-1} + \frac{1}{r} \left(\frac{1}{n} - \frac{1}{2n^2} + \frac{1}{3n^3} \right) - \frac{e^{r b_{n-1}}}{r n},$$

which further simplifies to

$$b_n \leq \left(1 - \frac{1}{n}\right) b_{n-1} - \frac{1}{r n^2} \left(\frac{1}{2} - \frac{1}{3n}\right).$$

Multiplying both sides by n and letting $c_n = n b_n$ yields

$$c_n \leq c_{n-1} - \frac{1}{r n} \left(\frac{1}{2} - \frac{1}{3n}\right).$$

Finally, the recurrence unfolds and simplifies to

$$c_n \leq (\pi^2/9 - \ln n - \gamma)/(2r) \leq (0.52 - \ln n)/(2r),$$

which yields the lemma. ■

Theorem 9 Suppose that **Adapt** has at least an expected learning rate of $\mathbf{E}^{(n)} L_n \geq \alpha A_n^r$, for some positive α and r . Then the average Fast-failure probability of **Adapt** after $n > 0$ steps of training has an upper bound of

$$\mathbf{E} A_n \leq (\alpha r(n+1))^{-1/r} \exp \left(\frac{(0.52 - \ln n)}{2r n} \right).$$

Proof From Lemma 6, we have for $n > 0$ that

$$\begin{aligned} \mathbf{E} A_n &= \mathbf{E} A_{n-1} (1 - \mathbf{E}^{(n-1)} L_{n-1}) \\ &\leq \mathbf{E} A_{n-1} (1 - \alpha A_{n-1}^r) \\ &\leq (\mathbf{E} A_{n-1}) (1 - \alpha \mathbf{E}^r A_{n-1}), \end{aligned}$$

where the last inequality comes from the fact that $\mathbf{E} X^r \geq \mathbf{E}^r X$ for $X \geq 0$. The desired upper bound can be obtained by identifying $\mathbf{E} A_n$ with a_n of Lemma 8. ■

Theorem 10 Suppose that **Adapt** has a maximum learning rate of $L_n \leq \beta A_n$ for some positive $\beta < 1$, and that the initial probability of failure is $A_0 = \mathbf{E} A_0 > 0$. Then $\mathbf{E} K_n$ has, in terms of $\mathbf{E} A_n$, the bound of

$$\beta^{-1} (1 - \beta) \ln(A_0 / \mathbf{E} A_n) \leq \mathbf{E} K_n \leq \ln_{1-\beta}(\mathbf{E} A_n / A_0),$$

which implies that $\mathbf{E} A_n$ has, in terms of $\mathbf{E} K_n$, the bound of

$$e^{-(\beta/(1-\beta))\mathbf{E} K_n} \leq \mathbf{E} A_n \leq (1 - \beta)^{\mathbf{E} K_n}.$$

Proof We first prove the lower bound for $\mathbf{E}K_n$. From Lemma 5, we have

$$A_{n+1} = A_n(1 - L_n I_n / A_n) \geq A_n(1 - \beta I_n) > 0.$$

Therefore, $A_{n+1}^{-1} \leq A_n^{-1} / (1 - \beta I_n) = A_n^{-1} (1 + \beta I_n / (1 - \beta))$, implying that

$$\begin{aligned} \mathbf{E}A_{n+1}^{-1} &\leq \mathbf{E}A_n^{-1} (1 + \beta I_n / (1 - \beta)) \\ &= \mathbf{E}A_n^{-1} (1 + \beta A_n / (1 - \beta)) \\ &\leq (\mathbf{E}A_n^{-1}) (1 + \beta \mathbf{E}A_n / (1 - \beta)). \end{aligned}$$

Unfolding the recurrence above yields $\mathbf{E}A_n^{-1} \leq A_0^{-1} \prod_{0 \leq j < n} (1 + \beta \mathbf{E}A_j / (1 - \beta))$. Taking the logarithm of both sides and using the fact that $\mathbf{E}A_n^{-1} \geq 1 / \mathbf{E}A_n$ gives us

$$\begin{aligned} \ln(A_0 / \mathbf{E}A_n) &\leq \sum_{0 \leq j < n} \ln(1 + \beta \mathbf{E}A_j / (1 - \beta)) \\ &\leq \sum_{0 \leq j < n} \frac{\beta}{1 - \beta} \mathbf{E}A_j = \frac{\beta}{1 - \beta} \mathbf{E}K_n, \end{aligned}$$

which yields the desired lower bound.

To obtain the upper bound of $\mathbf{E}K_n$, we simply unfold $A_{n+1} \geq A_n(1 - \beta I_n)$ into

$$A_n \geq A_0 \prod_{0 \leq j < n} (1 - \beta I_j).$$

Taking the logarithm of both sides yields

$$\begin{aligned} \ln A_n / A_0 &\geq \sum_{0 \leq j < n} \ln(1 - \beta I_j) \\ &= \sum_{0 \leq j < n} \sum_{i \geq 1} (\beta I_j)^i / i \\ &= \sum_{0 \leq j < n} \sum_{i \geq 1} \beta^i I_j / i \\ &= \ln(1 - \beta) \sum_{0 \leq j < n} I_j. \end{aligned}$$

The upper bound follows by taking the expectation of both sides and using the fact that $\mathbf{E} \ln A_n \leq \ln \mathbf{E}A_n$. ■

5 A Specific Case Analysis

To demonstrate our general theorems, consider again the simple two-dimensional environment $\mathcal{E}_1$ introduced earlier in illustrating the algorithm. In this environment, G is a chain wrapping around the obstacle with the boundary vertices on the circle. Points that are **Fast**-reachable are exactly those points of the arc covered by G and delimited by the two boundary vertices. Let C be the arc covered by G , and let the circumference be of unit length. Then probability A_n is exactly 1 minus the length of C_n .

To evaluate L notice that since we are using heuristic $h = h_1$ to guide our vertex selection for graph extension, we will always choose the boundary vertex closest to goal w as $\hat{w}$ in calling **Slow** when w lies outside C . Moreover, $\text{arc}(w, \hat{w})$ outside C contains exactly those points that would not be **Fast**-reachable after learning w . Hence, probability L_n is exactly the length of $\text{arc}(w_{n+1}, \hat{w}_{n+1})$ when $I_n = 1$, where I_n indicates the case of w_{n+1} lying outside C . Because w_{n+1} is uniformly distributed, we then have L_n uniformly distributed on the interval $[0, A_n/2]$. Consequently, **Adapt** has a maximum learning rate of $L_n \leq A_n/2$, and an expected learning rate of $\mathbf{E}^{(n)} L_n = A_n/4$.

Theorem 11 *The average number of calls that **Adapt** will make to **Slow** under $\mathcal{E}_1$ is $\mathbf{E}K_n = \Theta(\ln n)$.*

Proof Using Theorem 9 with $\alpha = 1/4$ and $r = 1$, we obtain $\mathbf{E}A_n = O(1/n)$. Applying Theorem 1 with this fact gives us immediately the upper bound of $\mathbf{E}K_n = O(\ln n)$. Applying the first inequality of Theorem 10 with $\beta = 1/2$ gives us the lower bound of $\mathbf{E}K_n = \Omega(\ln 1/\mathbf{E}A_n) = \Omega(\ln n)$. ■

Define the *inefficiency* of **Adapt** to be the ratio of $\mathbf{E}K_n$ over n . From Theorem 3 and the theorem above, we see that although the inefficiency of **Adapt** under $\mathcal{E}_1$ does approach 0, the memory requirement is actually unbounded. Fortunately, we can avoid this problem by seeking an **Fast**-planner with less than 100% accuracy. We simply stop the learning process after a training period as prescribed by the following theorem.

Theorem 12 *For **Adapt** under $\mathcal{E}_1$ to obtain, with confidence $1 - \delta$, a **Fast**-planner that can solve all but ϵ fraction of the tasks, at most $n_t \leq 5.2/(\epsilon\delta)$ steps of training are necessary.*

Proof From Theorem 9, we have $\mathbf{E}A_n \leq 4e^{0.26}/(n+1)$. Applying Theorem 4 with this fact yields the result. ■

Incidentally, if we have a convex m -sided polygon instead of a disk as the obstacle, **Adapt** would be able to perform much better. Once **Adapt** learns to reach a particular point of a side, every point on that side becomes reachable. Therefore, for **Adapt** to learn in this environment $\mathcal{E}_2$, at most m calls to **slow** and $O(m)$ number of trail-markers are required (cf. Theorem 7).

Let us now reconsider the unbounded memory problem that **Adapt** faces under $\mathcal{E}_1$. The difficulty is that its set of reachable goals can only approach, but never be the desired set. We can rectify this situation by modifying **Adapt** so that it learns more in the beginning. The modification allows the desired set to be learned completely so that slow learning does not occur at the end. Figure 3 shows the modified algorithm, **Modapt**, that satisfies our need.

In **Modapt**, we separate the initial training period that uses heuristic h' from the later working period that uses h . Ideally, h' should be more stringent than h , i.e., $\text{Fast}(u, w; G, h') \leq \text{Fast}(u, w; G, h)$, so that **Slow** $(u, w; G, h')$ can be called more often. For $\mathcal{E}_1$, we use h_2 that not only orders the vertices as in h_1 , but also rules out vertices v whose best potential path from u to w through v has length exceeding a certain value, say half the circumference. Using $h' = h_2$, we can force **Fast** $(u, w; G, h')$ to fail on cases where w lies on C , but the length of $\text{arc}(u, w)$ covered by C exceeds half the circumference. Subsequently, **Modapt** would be able to call **Slow** $(u, w; G, h')$, so that G can be extended to cover the complement of C in one step. Once the entire circle is covered by G , **Modapt** can stop learning and use **Fast** $(u, w; G, h)$ to reach every goal on the circle.

To analyze **Modapt** under $\mathcal{E}_1$, again let A_n be the probability of **Fast**-failure using G_n and h . Let $J_n = [\hat{w}_{n+1} = w_n]$, with $[\cdot]$ denoting the indicator function: 1 if the predicate is satisfied; 0 otherwise. Let $X_n \leq 1/2$ be the length of the shorter arc connecting w_{n+1} and $\hat{w}_{n+1}$ when **Fast** fails using G_n and h' , and 0 otherwise. Define $\bar{a} = 1 - a$, and $b = a - 1/2$.

```

Algorithm Modapt(Fast, Slow)
   $u \leftarrow$  current position;  $v \leftarrow u$ ;  $G \leftarrow (\{v\}, \emptyset)$ ;
  do while training
     $w \leftarrow$  goal;
    if (not Fast( $u, w; G, h'$ )) then
      if (not Slow( $u, w; G, h'$ )) then continue;
       $\rho \leftarrow$  Abstract(Slow[ $u, w; G, h'$ ]);
       $G \leftarrow$  Learn( $G, \rho$ );
    endif
    execute(Fast[ $u, w; G, h'$ ]);  $u \leftarrow w$ ;
  enddo
  do forever
     $w \leftarrow$  goal;
    if (not Fast( $u, w; G, h$ )) then continue;
    execute(Fast[ $u, w; G, h$ ]);  $u \leftarrow w$ ;
  enddo
end.

```

Figure 3: A modified algorithm to speed up learning.

Lemma 13 For $0 < x \leq 1/2$, the density functions $f_j(x, a)$ for X_n at x given $A_n = a$ and $J_n = j$ are

$$\begin{aligned}
 f_1(x, a) &= 1 + \lceil x \leq b \rceil \\
 f_0(x, a) &= (2/\bar{a}) \min(\bar{a}, 1/2 - x).
 \end{aligned}$$

Proof We use the fact that the length of C_n is $\bar{a}$, and the length of $\text{arc}(w_n, w_{n+1})$ containing $\hat{w}_{n+1}$ is at most $1/2$. For case $J_n = 1$, w_n is at either ends of C_n . So to achieve $X_n = x$, when $x + \bar{a} \leq 1/2$, two places for w_{n+1} are possible, and when $x + \bar{a} > 1/2$, only one place for w_{n+1} is possible. For case $J_n = 0$, w_n is uniformly distributed on C_n . So to achieve $X_n = x$, when $x + \bar{a} \leq 1/2$, exactly $\min(1/2, \bar{a})/\bar{a}$ fraction of C_n can be used for w_n to realize one of two sites for w_{n+1} . The fraction decreases linearly to 0 at $x = 1/2$. ■

With this lemma, we can evaluate the effective learning rate

$$\Delta A_n \stackrel{\text{def}}{=} A_n - A_{n+1}$$

by conditioning it on the variables $\lceil A_n \geq 1/2 \rceil$ and J_n . To simplify, let

$$E_{i,j}(a) \stackrel{\text{def}}{=} \mathbf{E}(\Delta A_n \mid A_n = a, \lceil A_n \geq 1/2 \rceil = i, J_n = j).$$

Lemma 14

$$\begin{aligned}
 E_{1,1}(a) &= 1/8 + b^2/2 \\
 E_{1,0}(a) &= (1/8 - b^3)/(3\bar{a}) \\
 E_{0,1}(a) &= a\bar{a}/2 \\
 E_{0,0}(a) &= (a^2(1/2 - 2a/3) + ab^2)/\bar{a}.
 \end{aligned}$$

Proof We use the fact that $\Delta A_n = \min(X_n, A_n)$ and the previous lemma on X_n to evaluate

$$E_{i,j}(a) = \int_0^{1/2} \min(x, a) f_j(x, a) dx$$

conditioned on $\lceil a \geq 1/2 \rceil = i$. In particular, we have

$$\begin{aligned} E_{1,1} &= \left(\int_0^b 2 + \int_b^{1/2} 1 \right) x dx \\ E_{1,0} &= \left(\int_0^b 2 + \int_b^{1/2} (2/\bar{a})(1/2 - x) \right) x dx \\ E_{0,1} &= \left(\int_0^a x + \int_a^{1/2} a \right) 1 dx \\ E_{0,0} &= \left(\int_0^a x + \int_a^{1/2} a \right) (2/\bar{a})(1/2 - x) dx. \end{aligned}$$

The rest is just simple computation. ■

Theorem 15 *For Modapt under $\mathcal{E}_1$ to obtain, with confidence $1 - \delta$, a Fast-planner that can solve all but ϵ fraction of the tasks, at most $n_t \leq \ln_{19/24}(\epsilon\delta)$ steps of training are necessary.*

Proof We show that $\mathbf{E}^{(n)} \Delta A_n \geq 5A_n/24$, which implies the upper bound of $\mathbf{E} A_n \leq (19/24)^n$ in the application of Theorem 4. For $J_n = 1$, w_n must be at the boundary of C_n , an event which occurs with probability $A_{n-1} \geq A_n$. Using the previous lemma, we bound $\mathbf{E}^{(n)} \Delta A_n$ by considering whether $A_n \geq 1/2$. Thus, let $a = A_n$, $x = \Pr(J_n = 1) \geq a$, and $i = \lceil a \geq 1/2 \rceil$. Then

$$\mathbf{E}^{(n)} \Delta A_n = x E_{i,1}(a) + (1 - x) E_{i,0}(a)$$

can be bounded below by setting $x = a$ because $E_{i,1}(a) - E_{i,0}(a)$, the coefficient of $x \geq a$, is nonnegative in the respective ranges of a depending on i . Specifically,

$$\begin{aligned} E_{1,1}(a) - E_{1,0}(a) &= \bar{a}^2/6, \\ E_{0,1}(a) - E_{0,0}(a) &= a(a^2/3 - b)/(2\bar{a}). \end{aligned}$$

Finally, the derived lower bound $a E_{i,1}(a) + \bar{a} E_{i,0}(a)$ simplifies to $g_i(a) a$ with

$$\begin{aligned} g_1(a) &= b^2/3 + (1 + 1/(3a))(1/8 + b^2) \\ g_0(a) &= (1/2 - a^2/3)/2, \end{aligned}$$

where $g_i(a)$ has a minimum of $5/24$ at $a = 1/2$ regardless of i . ■

Figure 4: An environment with two traps.

6 Particular Analysis

So far we have presented in the general analysis section a list of general relationships between important parameters of the learning process such as time cost, space cost, capability, and learning rate. We have also presented in previous section a specific case analysis of the learning algorithm applied to environment $\mathcal{E}_1$. Although the theorems of Table 2 are applicable to all environments, they are too general to yield immediate, useful results. On the other hand, results concerning one specific environment cannot generally be extended to another. It is thus the objective of this section to bridge the gap between general and specific case analysis.

In this section, we present an approach to analyzing speedup learning [31], which is what the algorithm is doing – seeking to improve program efficiency through learning. We first formalize the concept of improvability, and derive general conditions for such improvements. Next, we introduce two models with additional simplifying assumptions and parameters. Using these models, we then derive sharp bounds on planning cost and capability in terms of training time. Finally, we characterize the improvable situations in terms of the model parameters, and prescribe the amount of training required.

Definition 1 *Let A be a speedup learning algorithm designed to improve the efficiency of another algorithm A' . We say that*

1. *A can (cost-wise) improve A' with average failure probability p iff A can perform the same task as A' with average probability at least $1 - p$, while costing less on average.*
2. *A can effectively improve A' iff A can improve A' with failure probability no greater than that of A' .*
3. *A can effectively replace A' iff A can effectively improve A' without relying on A' .*

Using the variables of Table 1, we can immediately characterize the conditions under which improvements can be achieved. (Proof is clear.)

Lemma 16 *After n steps of training,*

1. *Fast can improve slow with failure probability $\mathbf{E}A_n$ iff $\mathbf{E}E_n < 1$.*
2. *Adapt can effectively improve slow iff $\mathbf{E}\Delta F_n < 1$.*
3. *Fast can effectively replace slow iff $\mathbf{E}E_n < 1$ and $\mathbf{E}A_n \leq 1 - \sigma$.*

To express these conditions in more useful terms of training time, we need to have further information such as the specification of the vertex ordering function h and the incremental learning strategy **Learn**. Thus, we introduce two models, one pessimistic (on task complexity), the other randomized (on experience utility), each with different applicability and additional simplifying assumptions. Using these models and the theorems of Table 2, we derive sharper bounds on the variables of Table 1, and explore the ramifications of Lemma 16.

In the pessimistic model, we study the worst-case consequence of learning in environments in which the strategy of **Learn** is specified, and the connectivity of S under **fast** is characterized. To motivate, consider a point robot in a planar polygonal environment shown in Figure 4, and let **fast** be 'go-straight'. Since we are dealing with a point robot, the C-space and the work space are the same. Clearly, the C-space is well-connected locally in the sense that each feasible (configuration) point is connectable (visible) to at least half of the entire C-space under **fast**. However, this environment may be difficult for learning algorithms with no teachers [18, 25] to handle in that the points randomly sampled will tend to form two disconnected components (traps) in A and B , and will not help in solving problems that require reaching B from A . In contrast, with the help of **slow**, our algorithm will adapt to this environment efficiently as long as the number of components induced by **fast** is not too large. Thus, using the following definition of the pessimistic model, we explore the consequences when the complexity of the C-space S relative to **fast** is measured by m in that S is m -coverable. Incidentally, the environment in Figure 4 is 2-coverable under **fast**, with the two components being A and B .

Definition 2 Under pessimistic model $\mathcal{M}_p$,

1. **Learn** adopts the minimal memory strategy of adding all necessary edges to take in only the minimal subchain sufficient to **Fast**-solve the current problem.
2. C-space S is m -coverable for some m in that S can be covered by m components (not necessarily disjoint), with the initial configuration in S_1 and each component S_i ($1 \leq i \leq m$) being connected under **fast**, i.e., every pair of points u, v in S_i satisfies **fast**(u, v).

In the randomized model, we study the average-case consequence of learning in environments in which the number of new trail-markers acquired by **Learn** and the power of **fast** are randomized. Thus, we are interested in the average behavior for a class of environments instead of a fixed environment. Using the following definition of the randomized model, we explore the consequences when the utilities of the learned trail-markers are measured by $\bar{\mu}$ and $\tilde{\mu}$. While $\mathcal{M}_r$ may not be physically realizable as opposed to $\mathcal{M}_p$, it does simplify the corresponding results for $\mathcal{M}_p$, and provide reasonable estimation tools as demonstrated in the applications section.

Definition 3 Under randomized model $\mathcal{M}_r$,

1. The number of new trail-markers acquired by **Learn**, λ , is an independent random variable.
2. **fast**(u, w) is 1 for any established edge (u, w) in V , and is 1 otherwise with independent probability $\bar{\mu}$.

Table 3 summarizes and compares the consequences of the two models. It shows that the reliance of **Adapt** on **slow** is at most inversely proportional to the training time n ; it also shows that the reliance is at most proportional to m , the complexity of S under $\mathcal{M}_p$, and is correspondingly, at most proportional to $1/\tilde{\mu}$, the power of **fast** under $\mathcal{M}_r$. In fact, the correspondence between these model parameters: m with $\tilde{\mu}$ and $\bar{\mu}$ is indicated throughout the table. The situation in which learning is useful (effective improvement or replacement of **slow**) is discerned by weighing $1/r$, the speed of **fast**, against m and correspondingly $1/\bar{\mu}$. To achieve this usefulness, the necessary training time is also prescribed. Finally, the speedup performance of **Adapt** as measured by the ratio of the planning cost of **Adapt** to that of **slow** is presented. The rest of this section is devoted to the details of these results. Readers uninterested in these details may skip to the next section on applications.

Table 3: Summary of particular analysis.

prop.	pessimistic model	randomized model
param.	m	$\bar{\mu}, \lambda, \tilde{\mu} \stackrel{\text{def}}{=} 1 - \mathbf{E}(1 - \bar{\mu})^\lambda$
capab.	$\mathbf{E}A_n \leq \frac{m-1}{\sigma n}$	$\mathbf{E}A_n \leq \frac{1}{\sigma \tilde{\mu}(n+1)}$
cost	$\mathbf{E}E_n \leq r(3m-2)$	$\mathbf{E}E_n = \frac{r}{\bar{\mu}}(1 - \mathbf{E}A_n)$
improve time	$r < 1/(3m-2)$ $n \geq \frac{(m-1)(1+\sigma c)}{\sigma(1-r(3m-2))}$	$r < \bar{\mu}$ $n \geq \frac{1}{\sigma \tilde{\mu}} \left(1 + \frac{\sigma c}{1-r/\bar{\mu}}\right)$
replace time	$r < 1/(3m-2), \sigma < 1$ $n \geq \frac{m-1}{\sigma(1-\sigma)}$	$r < \bar{\mu}, \sigma < 1$ $n \geq \frac{1}{\sigma(1-\sigma)\tilde{\mu}}$
cost ratio	$\frac{\mathbf{E}F_n}{n} \leq r(3m-2) + \frac{(m-1)(1+\sigma c)}{\sigma n}$	$\frac{\mathbf{E}F_n}{n} \leq \frac{r}{\bar{\mu}} + \left(1 + \sigma c - \frac{r}{\bar{\mu}}\right) \frac{\ln(eA_0\sigma\tilde{\mu}n)}{\sigma\tilde{\mu}n}$

6.1 Pessimistic Model

Under the pessimistic model, the complexity of the C-space S relative to **fast** is measured by m in that S is m -coverable. Using this complexity parameter, the following theorem says that the failure probability of **Adapt** is at most proportional to m and inversely proportional to the amount of training n , and that this bound is tight up to some constant factor.

Theorem 17 *Under $\mathcal{M}_p$, the expected Fast-failure probability of Adapt after n steps of training has a upper bound of*

$$\mathbf{E}A_n \leq \frac{m-1}{\sigma n}, \quad (5)$$

and a lower bound of

$$\mathbf{E}A_n \geq \frac{m-1}{e\sigma(n+1)} \quad (6)$$

for $n \geq (m-1)/\sigma$ and some environment dependent on n .

Proof (5) Let w_j be the j^{th} random problem. Let $X_{i,j}$ be the 0-1 random variable indicating that $w_{j+1} \in S_i$ and w_{j+1} is not **Fast**-solvable using V_j . Since the S_i 's cover S , we have $\mathbf{E}A_n \leq \mathbf{E}\sum_{i>1} X_{i,n}$. Also, since **Adapt** never forgets, we have $V_j \subseteq V_{j+1}$ for all j , which implies that $\mathbf{E}X_{i,j} \geq \mathbf{E}X_{i,j+1}$ because $\text{Fast}(w; V_j, h) \implies \text{Fast}(w; V_{j+1}, h)$ for all w . Finally, if $X_{i,j} = 1$ then **Slow** will be called. If it is successful, w_{j+1} will be remembered in V_{j+1} , causing $X_{i,j'} = 0$ for $j' > j$. Consequently, for any i , $\mathbf{E}\sum_j X_{i,j} \leq 1/\sigma$, which is the expected number of times that **Slow** will be called to reach some points in S_i before one is remembered. Combining all three inequalities, we

have

$$\mathbf{E}A_n \leq \sum_{i>1} \mathbf{E}X_{i,n} \leq \sum_{i>1} \sum_{0<j\leq n} \frac{\mathbf{E}X_{i,j}}{n} \leq \sum_{i>1} \frac{1}{\sigma n} = \frac{m-1}{\sigma n}.$$

(6) Let S be composed of exactly m non-overlapping components, with every component disconnected from each other except S_1 under **fast**. Let the distribution be uniform within each component and have total probability $p = 1/(\sigma(n+1))$ on S_i for $i > 1$, and probability $1 - (m-1)p$ on S_1 . Then for $i > 1$, the probability that **Adapt** will be given j training problems from S_i after n steps and failed to learn from them is $\binom{n}{j} p^j (1-p)^{n-j} (1-\sigma)^j$. Thus, with this probability summed over j , **Adapt** will **Fast-fail** on S_i . Summing up each $i > 1$, we have

$$\mathbf{E}A_n \geq (m-1)p \sum_j \binom{n}{j} (p(1-\sigma))^j (1-p)^{n-j} = (m-1)p(1-p\sigma)^n,$$

yielding the desired lower bound. ■

Using m again as a complexity measure of the C-space, the following theorem says that the **Fast-planning** cost of **Adapt** is at most linear in r and m , and that this bound is tight up to a constant factor. Further, the number of times **slow** will be needed is at most proportional to m and inversely proportional to its capability σ , and that this bound is tight with sufficient amount of training.

Theorem 18 *Under $\mathcal{M}_p$, the expected **Fast-planning** cost of **Adapt** after n steps of training has an upper bound of*

$$\mathbf{E}E_n \leq r \left(3m - 2 - (m-1) \left(1 - \frac{\sigma}{m-1} \right)^n \right). \quad (7)$$

*The expected number of calls to **Slow** has an upper bound of*

$$\mathbf{E}K_n \leq \left(\frac{m-1}{\sigma} \right) \left(1 - \left(1 - \frac{\sigma}{m-1} \right)^n \right). \quad (8)$$

Conversely, there exists an environment in which $\mathbf{E}E_n = r(2m-1)$, and an environment in which the equality of (8) is reached.

Proof Let J_n be the number of times **slow** is successful. Classify the trail-markers of ρ into two types: type 1 sharing the same S_i for some i with the current V , and type 2 does not. According to the ‘memory minimizing’ strategy of **Learn**, at most one marker of the subchain can be of type 1, and at most two markers per component not ‘occupied’ by the current V can be of type 2. (Otherwise, an edge can be introduced to shorten the subchain.) Thus, counting all subchains, the total number of type 1 and 2 markers are at most J_n and $2(m-1)$, respectively. Hence, $\|V_n\| \leq 1 + 2(m-1) + J_n$.

For (7), it suffices to show that $\mathbf{E}J_n \leq (m-1) \left(1 - \left(1 - \frac{\sigma}{m-1} \right)^n \right)$. Partition S into m disjoint components with the i^{th} component being $S'_i = S_i \setminus \bigcup_{j<i} S_j$. Let X_i be the 0-1 random variable indicating that one of the n training problems is both in S'_i and solvable by **Slow**. Then $J_n \leq \sum_{i>1} X_i$ because there can be at most one successful call of **Slow** for each $i > 1$. Let p_i be the probability that a random problem is both in S'_i and solvable by **Slow**. Then $\mathbf{E}X_i = 1 - (1-p_i)^n$ and $\sum_{i>1} p_i = \sigma - p_1$. Using the fact that $(p_2, \dots, p_m)$ majorizes [23] $(\frac{\sigma-p_1}{m-1}, \dots, \frac{\sigma-p_1}{m-1})$, we have $\sum_{i>1} (1-p_i)^n \geq (m-1)(1 - \frac{\sigma-p_1}{m-1})^n \geq (m-1)(1 - \frac{\sigma}{m-1})^n$, as desired. For (8), simply notice that $\sigma \mathbf{E}K_n = \mathbf{E}J_n$.

For the lower bound on $\mathbf{E}E_n$, let S be composed of exactly m non-overlapping components, with the first $m-1$ components consisting of exactly 2 points, $s_{i,1}$ and $s_{i,2}$. Let the only inter-component connections under **fast** be between $s_{i,2}$ and $s_{i+1,1}$. Let $s_{1,1}$ be the initial configuration, and let the distribution be 0 on the first $m-1$ components. Let h select the markers in the increasing order of the component index. Then upon solving the first problem, a path of $2m-1$ markers connecting $s_{1,1}$ to a point in S_m will be incorporated into V . Consequently, a **Fast**-planning cost of $r(2m-1)$ is required for latter problems.

For the lower bound on $\mathbf{E}K_n$, let S be composed of exactly m non-overlapping components, with every component disconnected from each other except S_1 under **fast**. For each $i > 1$, let the distribution have equal total probability $1/(m-1)$ on S_i . Then the learning process becomes effectively a coupon collector's problem [11] with $m-1$ types of coupons. Thus, $\mathbf{E}J_n = \sum_{i>0} \mathbf{E}X_i$, where X_i is the 0-1 random variable indicating that one of the n training problems is both in S_i and solved by **Slow**. The bound now follows since $\mathbf{E}X_i = 1 - (1 - \frac{\sigma}{m-1})^n$. $\blacksquare$

Using the estimates provided above, the following theorem discerns the situations in which **Adapt** is useful by weighing $1/r$, the speed of **fast**, against m , the complexity of S . For those situations in which **Adapt** can be useful, it also prescribes the amount of training required.

Theorem 19 Under $\mathcal{M}_p$, if $1/r > (3m-2)$, then **Adapt** can effectively improve **slow** with

$$n \geq \frac{(m-1)(1+\sigma c)}{\sigma(1-r(3m-2))} \quad (9)$$

steps of training. If **slow** is also not complete, then **Fast** can effectively replace **slow** with

$$n \geq \frac{m-1}{\sigma(1-\sigma)} \quad (10)$$

steps of training. If $2m-1 < 1/r \leq 3m-2$, then after

$$n < \ln\left(\frac{m-1}{\frac{m-1}{m-2}}\right) \left(\frac{m-1}{3m-2-1/r}\right) \quad (11)$$

steps of training, **Fast** can still improve **slow** with average failure probability no greater than $(m-1)/(\sigma n)$. If $1/r \leq 2m-1$, then there exist environments in which neither **Adapt** nor **Fast** can improve **slow**.

Proof From Theorem 18, we have $\mathbf{E}E_n < r(3m-2)$ for any finite n . If (9) holds, then from Theorem 17 and Theorem 2, we have $\mathbf{E}\Delta F_n \leq 1 - r(3m-2) + \mathbf{E}E_n < 1$, as prescribed by 2 of Lemma 16. Further, if $\sigma < 1$ and (10) holds, then $\mathbf{E}A_n \leq 1 - \sigma$ and $\mathbf{E}E_n < 1$, as prescribed by 3 of Lemma 16. If $2m-1 < 1/r \leq 3m-2$ and (11) holds, then from Theorem 18, $\mathbf{E}E_n < r(2m-1 + (m-1)(1 - (3m-2-1/r)/(m-1))) = 1$, as prescribed by 1 of Lemma 16. If $2m-1 \geq 1/r$, then by Theorem 2 and Theorem 18, there exists an environment in which $\mathbf{E}F_n \geq \mathbf{E}E_n = r(2m-1) \geq 1$, contrary to what is required for improvement in Lemma 16. $\blacksquare$

Finally, the speedup performance of **Adapt** is provided by the following.

Theorem 20 Under $\mathcal{M}_p$, the ratio of the average cost of **Adapt** to that of **slow** is bounded above by

$$\frac{\mathbf{E}F_n}{n} \leq r(3m-2) + \frac{(m-1)(1+\sigma c)}{\sigma n} \quad (12)$$

Hence, it is bounded asymptotically by $r(3m-2)$ as the number of training problems approaches infinity.

Proof The upper bound follows immediately from Theorem 2 and Theorem 18 with the simplifications that $\mathbf{E}K_n \leq r(3m - 2)$ and $\mathbf{E}K_n \leq (m - 1)/\sigma$. $\blacksquare$

6.2 Randomized Model

Under the randomized model, the utilities of the trail-markers learned are measured by $\bar{\mu}$ and $\tilde{\mu}$, with their reciprocals measuring the capability of **fast**. In parallel with the previous subsection, the following theorems estimate the key variables of the learning process with the capability of **fast** essentially replacing the role of the C-space complexity m .

Theorem 21 Under $\mathcal{M}_r$,

$$\mathbf{E}A_n \leq \frac{1}{\sigma\tilde{\mu}(n+1)}, \quad (13)$$

where $\tilde{\mu} \stackrel{\text{def}}{=} 1 - \mathbf{E}(1 - \bar{\mu})^\lambda$ denotes the average utility of a learned chain ρ .

Proof Let $L_n = A_n - A_{n+1}$ be the additional probability of problems learned through the incorporation of ρ_n into V_n . We call $\mathbf{E}(L_n | A_n)$ the expected learning rate, which evaluates to $A_n\sigma\tilde{\mu}$.

It now suffices to show that for $n > 0$, $\mathbf{E}A_n \leq 1/(\alpha(n+1))$ for an expected learning of $\mathbf{E}(L_n | A_n) = \alpha A_n$, $\alpha < 1$. For $n = 0$, $\mathbf{E}A_0 \leq 1 \leq 1/\alpha$. 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 $1/(2\alpha)$. For $n \geq 2$, we have from Theorem 9 that $\mathbf{E}A_n \leq (\alpha(n+1))^{-1} \exp((0.52 - \ln n)/(2n))$, which implies the desired upper bound. $\blacksquare$

Theorem 22 Under $\mathcal{M}_r$, the expected cost of **Fast** after n steps of training is

$$\mathbf{E}E_n = \frac{r}{\bar{\mu}}(1 - \mathbf{E}A_n). \quad (14)$$

Consequently, the expected cost of **Adapt** per problem after n steps of training is

$$\mathbf{E}\Delta F_n = r/\bar{\mu} + (1 + \sigma c - r/\bar{\mu})\mathbf{E}A_n. \quad (15)$$

Proof Let N be the number of markers in V_n , and Q_i be the probability that problem $n+1$ cannot be reduced by **fast** to any of the first i markers. Then

$$\begin{aligned} \mathbf{E}E_n &= r\mathbf{E} \sum_{0 \leq i < N} \mathbf{E}(Q_i | N) = r \sum_{0 \leq i < N} (1 - \bar{\mu})^i \\ &= \frac{r}{\bar{\mu}}(1 - \mathbf{E}(Q_N | N)) \\ &= \frac{r}{\bar{\mu}}(1 - \mathbf{E}A_n). \end{aligned}$$

The formula for $\mathbf{E}\Delta F_n$ follows from Theorem 2. $\blacksquare$

Theorem 23 Under $\mathcal{M}_r$, **Adapt** can effectively improve **slow** with sufficient training iff $r < \bar{\mu}$. Sufficient training can be achieved with

$$n \geq \frac{1}{\sigma\tilde{\mu}} \left(1 + \frac{\sigma c}{1 - r/\bar{\mu}} \right). \quad (16)$$

number of examples. If **slow** is also not complete, then **Fast** can effectively replace **slow** with

$$n \geq \frac{1}{\sigma(1 - \sigma)\tilde{\mu}} \quad (17)$$

steps of training. If $r \geq \bar{\mu}$, then **Fast** may still improve **slow**, but only with minimum failure probability $\mathbf{E}A_n \geq 1 - \bar{\mu}/r$.

Proof From Theorem 2 and Theorem 22, we have $\mathbf{E}\Delta F_n = r/\bar{\mu} + (1 + \sigma c - r/\bar{\mu})\mathbf{E}A_n$. Combining Lemma 16, Theorem 21, and this formula yields the desired theorem. $\blacksquare$

Theorem 24 Under $\mathcal{M}_r$, the ratio of the average cost of **Adapt** to that of **slow** is bounded asymptotically by $r/\bar{\mu}$ as the number of training problems approaches infinity. More globally, the behavior is

$$\frac{\mathbf{E}F_n}{n} \leq \frac{r}{\bar{\mu}} + \left(1 + \sigma c - \frac{r}{\bar{\mu}} \right) \begin{cases} \frac{\ln(eA_0\sigma\tilde{\mu}n)}{(\sigma\tilde{\mu}n)} & \text{if } A_0\sigma\tilde{\mu}n > 1; \\ A_0 & \text{otherwise.} \end{cases} \quad (18)$$

Accordingly, the maximum value that the ratio can attain at any n is at most

$$\mathbf{E}F_n/n \leq r/\bar{\mu} + (1 + \sigma c - r/\bar{\mu}) A_0. \quad (19)$$

Proof Let $\alpha = \sigma\tilde{\mu}$. From Theorem 2 and Theorem 22, it suffices to prove that $\mathbf{E}K_n \leq \ln(eA_0\alpha n)/\alpha$ if $A_0\alpha n > 1$; and $\mathbf{E}K_n \leq A_0n$ otherwise. 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 $\mathbf{E}K_n \leq A_0x + (H_n - H_x)/\alpha$, 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 $\mathbf{E}K_n \leq A_0x + \ln(n/x)/\alpha$, which yields the theorem when minimized at $x = \min(n, 1/(\alpha A_0))$. $\blacksquare$

7 Application and Verification

We now demonstrate the applicability of **Adapt** and the fidelity of our theory on a variety of robot environments. First, we investigate the simplest environment possible using the pessimistic model.

7.1 Pessimistic Model

Going back to the example in Figure 4 of a point robot in a 2-coverable workspace, we see from Theorem 17 that the expected failure probability of **Adapt** can be no greater than $1/(\sigma n)$ with n being the number of training problems. Notice that this result does not depend on what the problem distribution is, as long as it is fixed for both training and subsequent problem-solving. More generally, we have the following theorem for a point robot in simple planar environments.

Figure 5: A 10-dof robot environment.

Theorem 25 Consider a point robot in a planar simple polygonal workspace filled with b simple polygonal obstacles and having a total of c corners. Let **fast** be “go-straight”. Then the workspace is $(2b + c - 2)$ -coverable. Consequently, after n steps of training, the expected probability that **Adapt** will succeed in reaching the next random goal using only “go-straight” via the learned markers V_n is at least $1 - (2b + c - 3)/(\sigma n)$. Further, the expected cost of **Fast** is at most $3r(2b + c - 2)$.

Proof It suffices to show that the feasible workspace (a simple polygon with holes) can be triangulated into $m = 2b + c - 2$ triangles, since each triangle is convex, hence connected under “go-straight”. It is known that every simple polygon with holes has a valid constrained triangulation [28] whose edges are a superset of the input edges, and whose vertices are the input vertices (no added vertices are necessary). Let m and d be the number of triangles and edges in such a triangulation. Then by Euler’s formula [28], we have $(m + b + 1) - d + c = 2$. Also, counting each edge of every triangle yields $3m = 2d - c$. Hence, $m + b + c - (3m + c)/2 = 1$ implies the desired result of $m = 2b + c - 2$. ■

Beyond immediate applications to point robots, the theories thus developed can also help us make plausible performance predictions for more complicated robots. Figure 5a shows a 10-dof robot in a planar environment, which has been studied by others [18]. Let **fast** implement the following procedure:

1. move one end of the robot straight to the desired location with the rest of the robot complying;
2. with the first end point fixed, move the other end of the robot straight to the its desired location with the rest of the robot complying;
3. with both end points fixed, move the rest of the robot to their desired configuration using standard potential field approach.

Since the robot is snake-like with high dof, it is likely that **fast** will succeed if both end points are visible from their desired locations and if there is indeed a solution. Given that **fast** succeeds under this condition, we can bound the number of **fast**-connected components necessary to cover the 10-dimensional C-space. From Figure 5b, we see that the workspace is 11-coverable for each end point under visibility. Also, from visual inspection, we see that there are at most 12 topologically distinct inverse-kinematic solutions for a given pair of end points. Hence, the C-space is at most $11 \cdot 11 \cdot 12 = 1452$ -coverable. Consequently, if the teacher **slow** is a complete planner, it will take at most 145100 training problems for **Adapt** to attain a 99% expected capability.

Figure 6: Time improvement on a planar 2-link robot environment.

7.2 Randomized Model

Although the results from the pessimistic model can give us worst-case bounds on important quantities such as training time and capability, they are often too loose to predict the actual behaviors well. To complement and address this deficiency, we now demonstrate the randomized model by explaining data and making performance predictions on two separate experiments. Figure 6a shows a planar 2-link robot environment in which **Adapt** is applied. The environment exemplifies the planar component of a typical robot workcell in a SCARA configuration [10] with the z -component decoupled. In this experiment, **slow** implements an incomplete but fairly effective planner[4], and **fast** implements a simple potential-field based hill-climb. 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. In Figure 6b, the ratio of the cumulative planning cost of **Adapt** to that of **slow** only is plotted against problem 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. Figure 6c plots the ratio against $(\ln(n+1))/(n+1)$ to show their asymptotic linear relationship, hinted at by Theorem 24.

The experiment shows that **Adapt** is able to increase its performance relative to **slow** from 150% slower (ratio $\doteq 2.5$) to 50% faster at the end of 100 training examples. It also shows that **Adapt** needs about 16 training tasks before becoming competitive with **slow**, a fact attributable to both the task simplicity for **slow** and the significant learning costs incurred by **Adapt** during solution abstraction. We can use Theorem 24 to predict the maximum speedup achievable. If we believe that (18) is also an asymptotic lower bound, then the plot implies that $r/\bar{\mu} \doteq 0.38$ is the minimum achievable cost ratio, equivalent to a maximum speedup of 62%. From other empirical observations, 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 $\lambda \doteq 2$, we also estimate $\tilde{\mu} \doteq 0.45$. To see how consistent these numbers are, we estimate the number of training problems required by **Adapt** to have its cumulative cost first become less than that of **slow**. Using (18), we have $\sigma\tilde{\mu} \doteq 0.45$ and $\frac{\ln(eA_0\sigma\tilde{\mu}n)}{eA_0\sigma\tilde{\mu}n} \doteq \frac{1+\sigma c-r/\bar{\mu}}{eA_0(1-r/\bar{\mu})} \doteq 0.156$. giving us $eA_0\sigma\tilde{\mu}n \doteq 19$, or $n \doteq 17.2$, which is very close to the observed $n = 17$ in the plot.

We use our theory to explain and predict another experiment in which **Adapt** is applied on a 3-dimensional 6-dof gantry robot environment. The same **slow** and **fast** used for the planar case are also used here. In this environment (left side of Figure 7), there are 4 obstacles: a $(16+2)$ -sided polyhedral approximation of a cylindrical cask, two cask stands, and a floor. Motivated by problems

Figure 7: Time improvement on a 3-d, 6-dof robot problem.

in radiation survey [16], the goal positions are chosen randomly, and correspond to the robot end effector touching the cask surface in a prescribed orientation. The tasks are sufficiently difficult that the original planner, **slow**, fails to reach 7 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 **slow** only 5 times, and stores only 11 trail-markers in addition to the initial robot configuration. Figure 7 plots the task number against the the ratio of the cumulative effort expended by **Adapt** to that expended by **slow** only. Efforts are again measured by the number of robot-to-obstacle distance evaluations. The 5 large points indicate **Adapt**'s calling of **slow**, and the single white point indicates the only failure of **Adapt**. Initially, **Adapt** is able to plan without **slow** 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; $\bar{\mu} \doteq 1 - 0.01^{1/5} \doteq 80\%$ because only 5 chains are involved. Using (17) of Theorem 23, 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 **Fast** to improve both the speed and the capability of **slow**. This estimate means that **fast** is already very powerful, and that roughly only 2 calls (#17 and #18 in the plot) to **slow** are necessary for **Fast** to catch up with **slow** in task solving capability.

With Theorem 24, we can predict the maximum speedup achievable. 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 (18) is also a lower bound, then the maximum cost ratio is $r/\bar{\mu} + (1 + 0.1 - r/\bar{\mu})0.06 \doteq 0.32$ 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** will do much better with more training. Overall, **Adapt** is able to reduce the planning time of **slow** by a factor of $\bar{\mu}/r \doteq 4$, and increase the task solving capability of **slow** by $1/\sigma - 1 \doteq 7.5\%$.

8 Extension to Changing Environments

After presenting and analyzing **Adapt** for the fundamental stationary case, we now extend it to handle incrementally changing environments. In this more general environment, we assume that for each robot task, the obstacles are stationary, but may slowly change their configuration or shape over the course of the robot performing many tasks. In other words, we assume that the environmental change is both 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,

```

Algorithm Gen-Adapt(Fast, Slow; Trace)
   $u \leftarrow$  current position;  $v \leftarrow u$ ;  $G \leftarrow (\{v\}, \emptyset)$ ;
  do forever
    Repair( $G$ );
     $w \leftarrow$  goal;
    if (not Fast( $u, w; G, h$ )) then
      if (not Slow( $u, w; G, h$ )) then continue;
       $\rho \leftarrow$  Obj-Abstract(Slow[ $u, w; G, h$ ]);
       $G \leftarrow$  Learn( $G, \rho$ );
    endif
    if Trace( $u, w; G, h$ ) then
      execute(Fast[ $u, w; G, h$ ]);  $u \leftarrow w$ ;
    endif
  enddo
end.

```

Figure 8: An adaptive path planning algorithm in changing environment.

we mean that the workcell change involves only a few objects in a relative small area of the workspace. Both conditions are prevalent in many applications and have their intuitive implications: Occasional implies that old experience may be useful for a significant amount of time, and localized implies that old experience may have salvage value.

Our algorithmic extension consists of two experience manipulation schemes: For minor environmental adjustments, we use an object-attached experience abstraction scheme to increase the flexibility of the learned experience; for major environmental modifications, we use an on-demand experience repair scheme to retain those experiences that remain valid and useful. In addition to presenting this extension, we also compare it with three other variant strategies for using old experiences in new environments. Formally, the generalized learning algorithm **Gen-Adapt** is shown in Figure 8. It is the same as the algorithm for stationary environments except for the three boxed fragments. The second boxed fragment replaces **Abstract** with **Obj-Abstract**, the object-attached experience abstraction scheme. The third boxed fragment introduces **Trace**, the on-demand experience repair scheme. The first boxed fragment, which introduces **Repair**, is not part of the algorithm, but is included for later discussion of other variants of the algorithm that use it.

8.1 Object-attached Experience Abstraction

Recall in the specification of **Adapt** for stationary environments that the procedure **Abstract** is used to condense a work experience **slow**[v, w] into a short chain ρ of trail-markers for **fast** to traverse. In fixed environments, it does not matter how we choose to represent markers because they always correspond to fixed robot configurations. However, to increase the flexibility of their use in changing environments, we now require that the markers returned by **Obj-Abstract** (second boxed fragment in Figure 8) be relative robot configurations associated with nearby objects, rather than the absolute positions in the stationary case. That is, instead of remembering the robot positions as points in absolute space, we now remember each of them as an offset from some nearby object serving as a

Figure 9: Object-attached experience using critical tag-points.

landmark.

One way to implement this strategy is to create a tag-point (a 6 degrees-of-freedom coordinate frame for the robot tool point) for each critical robot position, and affix the tag-point to the local coordinate of a nearby object. Then, as this nearby object changes its location or orientation, the tag-point can be adjusted accordingly so that the robot tool point can maintain its distance to the object under change. Figure 9 shows an example. In the left frame, the robot position is recorded via the tag-point of the robot tool, and is attached to the rectangular object. As the object moves toward the right, the tag-point moves along with it, enabling the robot to comply with the change. If the tag-point had not been attached to the object, the corresponding robot position would have become invalid in the new environment.

One potential drawback of this tag-point method is that solving the inverse-kinematics for the tag-point will be necessary to recompute the robot configuration for the trail-marker. Thus, multiple solutions may arise not all of which may be feasible. Solutions may also disappear for tag-points whose attached objects have moved too much. Nevertheless, under this object-attached experience abstraction scheme, we can adjust to any minor environmental change without expensive experience repair.

8.2 On-Demand Experience Repair

Of course, if the environment changes significantly, the validity of G will deteriorate. The edges that used to be valid in previous environments may not stay valid in the same environment in the sense that **fast** may no longer be powerful enough to traverse them. How much deterioration G will 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. Therefore, we introduce an on-demand repair scheme (third boxed fragment in Figure 8) to retain those experiences that remain valid and useful.

In this scheme, we plan as if G is connected, until **Fast**($\cdot$) succeeds and we actually need to produce a path. Then, to generate **Fast**[$\cdot$], we require the success of **Trace**($\cdot$) to provide a connected sequence from $\hat{u}$ to $\hat{w}$. As **Trace**($\cdot$) 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 **slow** 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 $\text{Trace}(\cdot)$ is as follows:

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 $\text{fast}(\Gamma_i, \Gamma_{i+1})$ for all $1 \leq i < k$ then return success;
 - (b) Else remove edge (Γ_i, Γ_{i+1}) with smallest i such that $\neg \text{fast}(\Gamma_i, \Gamma_{i+1})$.
2. If $\text{slow}(\hat{u}, \hat{w})$ succeeds then augment G with $\text{Abstract}(\text{slow}[\hat{u}, \hat{w}])$; return success;
3. Else remove the (connected) component of $\hat{w}$ from G , and return failure.

8.3 Other Repair Strategies

It is also possible to cope with major environmental change using other variants of the on-demand repairing strategy. One trivial strategy as stated above is simply to forget the old experience and start over (with G reinitialized) whenever there is a change in the environment. The corresponding algorithm, A_0 , can be obtained from Figure 1 by skipping the boxed condition, and defining $\text{Repair}(G)$ to be the reinitialization procedure.

Another less trivial strategy is to verify each edge of G first whenever there is a change. Then with the time investment, we can initialize G to the home component that contains the current robot position. The corresponding algorithm, A_1 , can again be obtained from Figure 1 by skipping the boxed condition, and defining $\text{Repair}(G)$ to be the above home-component extraction procedure.

Notice that both strategies above only update G according to environmental change, and do not really repair old experience. In contrast, a third strategy that repairs actively is to first apply Trace to attempt to reach every vertex of G from home before taking on any new task. The corresponding algorithm, A_2 , can be obtained from Figure 1 by skipping the boxed condition, and defining $\text{Repair}(G)$ to be the above repair-all procedure.

All of the suggested algorithms (including the repair-on-demand algorithm A_3) have their advantages and disadvantages. Intuitively, if the environment undergoes a major and extensive change, then starting over with A_0 may be the best choice. On the other hand, if slow costs much more than fast , then using A_1 to save some old experience may be better. Alternatively, if the change is only local, then repairing old experience with A_2 or A_3 may be more beneficial. Which algorithm to use thus depends on the particular application.

8.4 Solution Quality and Redundancy

So far we have focused on task solvability but not solution quality. If solution quality is not important, then in $\text{Fast}[\cdot]$ we can simply produce the solution of going through Γ with fast . 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 seeking to "cut corners" whenever possible. The result of this compression is that G may be augmented with additional edges to enable shorter sequences in the future. Also, the redundancy introduced may be useful in combating experience deterioration. In fact, for environments with a shrinking set of obstacles such as those encountered in waste-remediation, compressing known solutions would be a simple and cost-effective approach to improving solution quality.

8.5 Example

We illustrate the generalized learning algorithm with a simple example involving a point robot in a 2D workspace. Let fast implement a go-straight procedure, with $\text{fast}(u, w)$ returning success iff w

Figure 10: Snapshots of Gen-Adapt under environmental change

is visible from u , and **fast** $[u, w]$ returning the line segment $\overline{uw}$. Let **slow** implement a greedy 2-step go-straight procedure, with **slow** (u, w) returning success iff the two points are connectable by at most 2 line segments, and **slow** $[u, w]$ returning the shortest such connecting path. To complete the algorithmic specification, let the heuristic used in **Fast** and **Slow** be $h = h_1$, with h_1 ordering the vertices of G according to the distance to w , starting with the closest point first.

Figure 10 illustrates **Gen-Adapt** with a series of snapshots. Frame (1) shows the initial setting with the robot at home $u = w_0$ amongst two objects A and B . The robot's initial tasks are to inspect both A from w_1 and w_2 , and B from w_3 and w_4 . To begin, the experience graph G is initialized to the single vertex $v_0 = w_0$.

The first goal indicated by w_1 is shown in Frame (2). Since **Fast** is unable to plan using only **fast** and G , **Gen-Adapt** then calls **Slow**. Using h , **Slow** chooses to extend from v_0 to w_1 , since v_0 is the only vertex in G . The path produced by **slow** (v_0, w_1) consists of the line segments $\overline{v_0v_1}$ and $\overline{v_1v_2}$. This path is then abstracted into the chain connecting v_0 to v_1 and v_1 to v_2 . The result of augmenting G is that G now becomes the 3-vertex chain. Using this augmented G , **Fast** is now able to produce a path from $u = w_0$ to w_1 , which consists of the segments $\overline{uv_0}$, $\overline{v_0v_1}$, $\overline{v_1v_2}$, and $\overline{v_2w_1}$, with $\overline{uv_0}$ and $\overline{v_2w_1}$ being null segments.

With the first task accomplished, the next task is to go to w_2 shown in Frame (3). Since **Fast** is again unable to plan using only **fast** and G , **Gen-Adapt** then calls **Slow**. Using h , **Slow** chooses to extend from v_2 to w_2 , and produces the line segments $\overline{v_2v_3}$ and $\overline{v_3v_4}$. The result of augmenting G is that G now becomes the 5-vertex chain with new vertices v_3 and v_4 .

With this G , **Fast** is still unable to succeed in reaching w_3 in Frame (4). Consequently, **Slow** chooses to extend from v_0 and produces 2 more segments $\overline{v_0v_5}$ and $\overline{v_5v_6}$. Thus, before calling **Fast** $[\cdot]$, G is a 7-vertex chain with new vertices v_5 and v_6 . After calling **Fast** $[\cdot]$, however, G becomes cyclic due to the addition of edges (v_3, v_1) and (v_1, v_5) as a result of locally optimizing the solution path $(v_4, v_3, v_2, v_1, v_0, v_5, v_6)$.

Frame (5) shows that **Fast** is now capable of reaching w_4 , with $\hat{w}_4 = v_1$. Consequently, for the first time, **Slow** is not called and G is not modified.

So far, the workcell has been stationary. In Frame (6), we return the robot to its home and introduce a new object C . With A_0 using the start-over strategy, we would lose the entire G and not retain anything from the 3 previous calls to **Slow**. With A_1 , we would verify all 8 edges in G with **fast**, remove the only broken edge (v_0, v_1) , and retain the rest of G since it remains connected. If **fast** costs much less than **slow**, then the return on the initial time investment is certainly justifiable compared

Figure 11: Workcell change and time improvement of **Gen-Adapt** over **slow**.

to that of A_0 . This case demonstrates that improving solution quality can also increase experience redundancy, which in turn decreases experience deterioration under change. (The introduction of edge (v_1, v_5) in locally optimizing $(v_4, v_3, v_2, v_1, v_0, v_5, v_6)$ alleviates the damage of edge (v_0, v_1) caused by object C .) With A_2 using the active-repair scheme, we would also just remove edge (v_0, v_1) from G at the end of $\text{Repair}(G)$. With A_3 using the repair-on-demand strategy, we simply do nothing.

Frame (7) shows what happens if we introduce some minor change by moving object B and its object-attached goals w_3 and w_4 . Because of the object-attached abstraction scheme, v_5 and v_6 also move along with B . Consequently, if the robot were to go back to w_3 , it would again succeed by simply reaching toward v_5 and v_6 .

Frame (8) shows what happens if we move object C to a corner and decide not to inspect object B anymore. In this case, A_1 would be identical to A_0 in reducing G back to the single vertex v_0 , except that A_1 would also have to spend time verifying all 7 edges of G before removing them. With A_2 , G would be actively repaired, which means that it would call **slow** twice to reestablish the connectivity of the 2 components to v_0 . With A_3 , we again do nothing until the need arises. If we choose not to inspect B anymore, then only one component needs to be reconnected to v_0 , which means only one additional call to **slow** would be required in the future. This case demonstrates a situation where using A_3 is better than using A_2 .

8.6 Computational Experience

We have applied **Gen-Adapt** on the simple 2-link planar robot environment of Figure 6a again, this time with an environment change. Recall the initial setup: the workcell has 5 polygonal obstacles and the goal set consists of 9 preselected robot configurations. Starting at home 0, the robot is to go through a sequence of goals randomly selected from the goal set. During the exercise, we introduce an incremental environmental change by adding a new obstacle to the workcell and a new goal position to the goal set, as shown in Figure 11.

The result of this experiment, with **Gen-Adapt** using all 4 different repairing strategies, is shown in Figure 11. Here, the ratio of the cumulative planning cost required by **Gen-Adapt** to that required by **slow** only is plotted against the task number. The planning costs are averaged over 100 runs and are again measured by the number of robot-to-obstacle distance evaluations, which is the dominating factor in the computing cost of each planner. The environment change is introduced after task 40. To emphasize the important features of the result, the initial portion of the curve

corresponding to ratios greater than 1 is not plotted. The uncharted portion actually decreases monotonically from 2.5 at task number 1 to 1.0 at task number 16. The experiment shows that before the environmental change, **Gen-Adapt** is able to learn and speed up its performance relative to **slow** from 150% slower to 33% faster. (As shown in Figure 6b, this speedup can be increased to 50% faster if the environment stays fixed.)

After the environmental change, the performance curve for **Gen-Adapt** splits up into 4 curves, each corresponding to a different experience repairing strategy. The curves for A_0 , A_1 , and A_3 exhibit similar behaviors in that they all gradually increase and then decrease at roughly the same rate, with A_3 being clearly better than A_1 , which is in turn clearly better than A_0 . The curve for A_2 is different in that it first jumps to a high point and then comes down rapidly to approach the curve for A_3 . The jump is due to the high initial cost of active repair, and the rapid decrease is due to the benefit of the repair. Overall, the relative performance of the repairing strategy is as expected, since the environmental change is incremental, involving only local and occasional change. In fact, one can devise an experiential cost/benefit model to formalize the concept of local and occasional change, and prove the optimality of the on-demand repair strategy A_3 relative to the other variants A_0 , A_1 , and A_2 under such change [7].

9 Conclusion

We have presented a learning algorithm that can improve path planning performance. The algorithm adapts to its working environment by maintaining an experience graph with vertices corresponding to useful robot configurations. It can both reduce time cost and increase task solving capability of existing planners.

We have demonstrated the effectiveness of the algorithm via both mathematical analysis and experimentation. Our analysis involves uncovering general quantitative relationships between important variables such as capability, planning cost, memory requirement, and training time. It also includes studying the implications of these relationships under two stochastic models: pessimistic $\mathcal{M}_p$ and randomized $\mathcal{M}_r$. The models have different assumptions and applications: $\mathcal{M}_p$ quantifies C-space complexity while $\mathcal{M}_r$ quantifies experience utility. Using these models, we characterize the situations in which learning will yield a speedup, and provide global quantitative bounds on planning cost and capability in terms of training cost. Empirically, we have also demonstrated the applicability of the algorithm and the fidelity of its theory on several robot path planning environments. In particular, we have illustrated a technique for predicting the maximum achievable speedup. Finally, we have extended our algorithm to handle changing environments with object-attached experience abstraction and on-demand experience repair. The performance of the generalized algorithm is characterized and compared with three other plausible experience repair strategies.

Although our algorithm is presented in the context of robot path planning, there is no restriction that it be applied only to robotics. In fact, at a higher level of abstraction, the algorithm may even be applicable for an intelligent agent navigating in a space of information subject to potential incremental change. In this sense, our elementary theoretical results and techniques should be useful for studying other types of probabilistic learning as well.

10 Acknowledgements

This work is supported by the Laboratory-Directed Research and Development Program at Sandia National Laboratories and by the U.S. Department of Energy under Contract DE-AC04-94AL85000. The author thanks Yong Hwang, Sharon Stansfield, David Strip, and Patrick Xavier for their help in reviewing and improving this work throughout the years.

References

- [1] Barbehenn, M., Chen, P.C., Hutchinson, S., "An Efficient Hybrid Planner in Changing Environments," *Proc. of IEEE Int. Conf. on Robotics and Automation*, pp. 2755-2760, 1994.
- [2] Barraquand, J., Latombe, J., "A Monte-Carlo algorithm for path planning with many degrees of freedom," *Proc. of IEEE Int. Conf. on Robotics and Automation*, pp. 1712-1717, 1990.
- [3] Bessiere, P., Ahuactzin, J.M., *et al*, "The 'Ariadne's Clew' algorithm: Global planning with local methods," *Proc. of IEEE/RSJ Conf. on Intelligent Robots and Systems*, 1993.
- [4] Chen, P.C., "Effective Path Planning through Task Restrictions," Sandia Report SAND91-1964, 1992.
- [5] Chen, P.C., "Improving Path Planning with Learning," *Machine Learning: Proc. of the Ninth Int. Conf.*, pp. 55-61, 1992.
- [6] Chen, P.C., Hwang, Y.K., "SANDROS: A Motion Planner with Performance Proportional to Task Difficulty," *Proc. of IEEE Int. Conf. on Robotics and Automation*, pp. 2346-2353, 1992.
- [7] Chen, P.C., "Adaptive Path Planning in Changing Environments," Sandia Report SAND92-2744, 1993.
- [8] Chen, P.C., "Adaptive Path Planning for Flexible Manufacturing," *Proc. of Fourth Int. Conf. on Computer Integrated Manufacturing and Automation Technology*, Oct., 1994.
- [9] Chen, P.C., "Adaptive Path Planning: Algorithm and Analysis," *Proc. of IEEE Int. Conf. on Robotics and Automation*, 1995.
- [10] Craig, J., *Introduction to Robotics: Mechanics and Control*, pp. 268-269, 1989.
- [11] Feller, W., *An Introduction to Probability Theory and Its Application*, 3rd edition, v. 1, John Wiley & Sons, 1968.
- [12] Fujimura, K., Samet, H., "Time Minimal Paths among Moving Obstacles," *Proc. of IEEE Int. Conf. on Robotics and Automation*, pp. 1110-1115, 1989.
- [13] Glavina, B., "Solving Findpath by Combination of Goal-Directed and Randomized Search," *Proc. of IEEE Int. Conf. on Robotics and Automation*, pp. 1718-1723, 1990.
- [14] Goel, A., Callantine, T., Donnelian, M., Vazquez, N., "An intergrated experience-based approach to navigational path planning for autonomous mobile robot," *Proc. of IEEE Int. Conf. on Robotics and Automation*, pp. 818-825, 1993.

- [15] Gupta, K.K., Zhu, X., "Practical Global Motion Planning for Many Degrees of Freedom: A Novel Approach within Sequential Framework," *Proc. of IEEE Int. Conf. on Robotics and Automation*, pp. 2038-2043, 1994.
- [16] 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.
- [17] Hwang, Y.K., Ahuja, N., "Gross Motion Planning - A Survey," *ACM Computing Surveys*, **24**, no. 3, pp. 219-292, Sept. 1992.
- [18] Kavraki, L., Latombe, J.-C., "Randomized preprocessing of configuration space for fast path planning," *Proc. of IEEE Int. Conf. on Robotics and Automation*, pp. 2138-2145, 1994.
- [19] Kant, K., Zucker, S.W., "Planning Collision-free Trajectories in Time-Varying Environments: a Two-level Hierarchy," *Proc. of IEEE Int. Conf. on Robotics and Automation*, pp. 1644-1649, 1988.
- [20] Kondo, K., "Motion Planning with Six Degrees of Freedom by Multistrategic Bidirectional Heuristic Free-Space Enumeration", *IEEE Tran. on Robotics and Automation*, **7**, no. 3, pp. 267-277, June 1991.
- [21] Latombe, J., *Robot Motion Planning*, Kluwer Academic Publishers, 1991.
- [22] Lozano-Pérez, T., "A Simple Motion-Planning Algorithm for General Robot Manipulators," *IEEE J. of Robotics and Automation*, **RA-3**, no. 3, pp. 224-238, June 1987.
- [23] Marshall, A.W., Olkin, I., *Inequalities: Theory of Majorization and Its Applications*, Academic Press, 1979.
- [24] Natarajan, B.K., *Machine Learning: A Theoretical Approach*, Morgan Kaufmann, 1991.
- [25] Overmars, M.H., Švestka, P., "Probabilistic Learning Approach to Motion Planning," *Workshop on the Algorithmic Foundations of Robotics*, Feb. 1994.
- [26] Pandya, S., Hutchinson, S., "A Case-based Approach to Robot Motion Planning," *Proc. of IEEE Int. Conf. on Systems Man and Cybernetics*, pp. 492-497, 1992.
- [27] Paden, B., Mees, A., Fisher, M., "Path Planning Using a Jacobian-Based Freespace Generation Algorithm," *Proc. of IEEE Int. Conf. on Robotics and Automation*, pp. 1732-1737, 1989.
- [28] Preparata, F.P., Shamos, M.I., *Computational Geometry: An Introduction*, Springer-Verlag, 1988.
- [29] Reif, J.H., Sharir, M., "Motion Planning in the Presence of Moving Obstacles," *Proc. of 26th FOCS*, pp. 144-154, 1985.
- [30] Shiriyayev, A.N., *Probability*, Springer-Verlag, 1984.
- [31] Tadepalli, P., "A Theory of Unsupervised Speedup Learning," *Proc. of AAAI*, pp. 229-234, 1992.

Appendix A: LDRD Information

- List of publications and presentations resulting from the project.
 1. "Improving Path Planning with Learning," *Machine Learning: Proc. of the Ninth Int. Conf.*, 1992.
 2. "Adaptive Path Planning in Changing Environments," Sandia Report SAND92-2744, 1993.
 3. "An Efficient Hybrid Planner in Changing Environments," *Proc. of IEEE Int. Conf. on Robotics and Automation*, 1994.
 4. "Adaptive Path Planning for Flexible Manufacturing," *Proc. of the Fourth Int. Conf. on Computer Integrated Manufacturing and Automation Technology*, 1994.
 5. "Generating Natural-Language like Robot Motion Command Set through Machine Learning," *Proc. of IEEE Int. Conf. on Systems, Man and Cybernetics*, 1994.
 6. "Adaptive Robot Path Planning in Changing Environments," *Proc. of IEEE Int. Conf. on Tools with Artificial Intelligence*, 1994.
 7. "Adaptive Path Planning: Algorithm and Analysis," *Proc. of IEEE Int. Conf. on Robotics and Automation*, 1995.
- List of invention disclosures resulting from the project. None.
- List of patents resulting from the project. None.
- List of copyrights resulting from the project. None.
- Information regarding employee recruitment as a result of the project.

The project resulted in one internal transfer and one employment offer to a computer science PhD candidate.
- Information regarding involvement of students in the project.

One computer science graduate student participated in the project during the summer of 1993.
- Information regarding follow-on work (new activities, projects, sponsors).

New projects include work on intelligent agents for information processing and learning interfaces for interactive task planning.

Distribution for SAND95-0722:

MS9018	Central Technical Files, 8523-2	(1)
MS 0899	Technical Library, 13414	(5)
MS 0619	Technical Publications, 12615	(1)
MS 0100	Document Processing for DOE/OSTI, 7613-2	(2)
MS 0951	Pang Chen, 2121	(20)