# Learning Situation-Dependent Costs: Using Execution to Refine Planning Models

**Karen Zita Haigh**
khaigh@cs.cmu.edu
http://www.cs.cmu.edu/~khaigh

**Manuela M. Veloso**
mmv@cs.cmu.edu
http://www.cs.cmu.edu/~mmv

Computer Science Department,
Carnegie Mellon University,
Pittsburgh PA 15213-3891

## Abstract

Physical environments are so complex that it is hard to hand-tune all of the domain knowledge, especially to model the dynamics of the environment. The work presented in this paper explores machine learning techniques to autonomously identify situations in the environment that affect plan quality. We introduce the concept of *situation-dependent costs*, where situational features can be attached to the costs used by the path planner. These costs effectively diagnose and predict situations the robot encounters so that the planner can generate paths that are appropriate for each situation.

We present an implementation of our situation-dependent learning approach in a real robotic system, Rogue. Rogue learns situation-dependent costs for arcs in a topological map of the environment; these costs are then used by the path planner to predict and avoid failures. In this article, we present the representation of the path planner and the navigation modules, and describe the execution trace. We show how training data is extracted from the execution trace. We present experimental results from a simulated, controlled environment as well as from data collected from the actual robot. Our approach effectively refines models of dynamic systems and improves the efficiency of generated plans.

## 1 Introduction

Robots operating in the real world have many tasks they need to perform autonomously. Reliability and efficiency are key issues when designing real systems. The robot must be able to effectively deal with noisy sensors and actuators, as well as incomplete and changing knowledge about the environment. It must act efficiently, in real time, to deal with dynamic situations. In most current systems, the behaviour of an autonomous robot is completely dependent on the predictive ability of the programmer. Most real world environments are hard to model completely and correctly. Moreover, the world is dynamic and models need to account for changes. Any system that repeats its errors cannot be considered reliable. To be *truly* autonomous, the robot needs to be able to use accumulated experience and feedback about its performance to improve its behaviour. It needs to *learn*.

Learning has been applied to robotics problems in a variety of manners. Common applications include map learning and localization (e.g. [9; 10; 21]), or learning operational parameters for better actuator control (e.g. [1; 3; 15]). Instead of improving low-level *actuator* control, our work focusses at the *planning* stages of the system.

A few other researchers have explored this area as well, learning costs of actions, or their applicability criteria (e.g. [8; 11; 17; 14; 20]).

IMPROV [14] learns action descriptions, but its performance degrades dramatically with environmental noise. Clementine [11] and CSL [20] both learn sensor utilities, including which sensor to use for what information. LIVE [17] learns a model of the environment, as well as the costs of applying actions in that environment.

In some situations, it is enough to learn that a particular action has a certain average probability or cost. However, there are times when actions may have different costs under different situations. Instead of learning a global description, we would rather that the system learn the pattern by which these situations can be identified. It is important to have a more fine-grained diagnosis of problem situations. The system needs to learn the correlation between features of the environment and the situations, thereby creating *situation-dependent costs*.

We would like a *path planner* to learn, for example, that a particular corridor gets crowded and hard to navigate when classes let out. We would like a *task planner* to learn, for example, that a particular secretary doesn't arrive before 10am, and tasks involving him can not be completed before then. We would like a *multi-agent planner* to learn, for example, that one of its agents has a weaker arm and can't pick up the heaviest packages. Once these problems have been identified and correlated to features of the environment, the planners can then predict and avoid them when similar conditions occur in the future.

We have been developing a robot architecture, Rogue, which aims at equipping a real robot with the ability to learn from its own execution experiences [7]. This paper presents the continuation of the project, in which learning has been incorporated. Rogue processes uncertain navigation data to create improved domain models for its planners, successfully abstracting numeric sensor information into symbolic planner information. Rogue effectively refines the domain model to capture the patterns of the dynamic world.

Our approach relies on examining the execution traces of the robot to identify situations in which the planner's behaviour needs to change. Our approach requires that the robot executor defines the set of available situation *features*, $\mathcal{F}$, while the planner defines a set of relevant learning opportunities, or *events*, $\mathcal{E}$, and a *cost function*, $\mathcal{C}$, for evaluating those events.

1. Create plan
2. Execute; record execution trace including features $\mathcal{F}$
3. Identify events $\mathcal{E}$ in execution trace
4. Learn mapping: $\mathcal{F} \times \mathcal{E} \rightarrow \mathcal{C}$
5. Update planner

Table 1: General approach for learing situation-dependent costs.

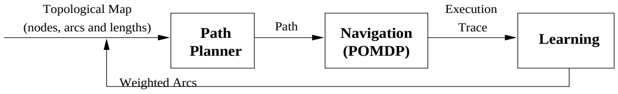



Figure 2: Learning framework. The path planner and navigation modules are part of Xavier's software architecture; the learning module is provided by ROGUE.

Figure 1: Xavier the Robot

Events are things in the environment for which additional knowledge will cause the planner's behaviour to change. Features discriminate between those events, thereby creating the required additional knowledge. The cost function allows the planner to evaluate the situation and select appropriate actions. The learner creates a mapping from the execution features and the events to the costs: $\mathcal{F} \times \mathcal{E} \rightarrow \mathcal{C}$. For each event $\varepsilon \in \mathcal{E}$, in a given situation described by features $\mathcal{F}$, this learned mapping calculates a cost $c \in \mathcal{C}$. We call this mapping a *situation-dependent rule*.

Once the rule has been created, the information is given back to the planner so that it will avoid re-encountering the problem events. Essentially, a situation-dependent rule gives a planner to diagnose problems before they occur, hence effectively predicting and avoiding failures. These steps are summarized in Table 1. Learning occurs incrementally and off-line; each time a plan is executed, new data is collected and added to previous data, and then all data is used for creating a new set of situation-dependent rules.

ROGUE uses the Xavier robot [18] as its learning platform (see Figure 1). Knowledge in the path planner is represented as a topological map of the environment in which the robot navigates. The map is a graph with nodes and arcs representing office rooms, corridors, doors and lobbies. Learning appropriate arc-cost functions will allow the path planner to avoid troublesome areas of the environment when appropriate. Therefore events for this planner are arc traversals, features are robot data, and costs are travel time and position confidence.

Xavier's execution trace is generated by a probabilistic navigation module. Identifying the planners' events from this trace is challenging because the execution traces contain a massive, continuous stream of uncertain data.

In this article, we present the learning algorithm as it applies to Xavier's path planner. Our system demonstrates the ability to learn *situation-dependent costs* for the arcs of the path planner. It extracts relevant training data from the massive, continuous, probabilistic execution traces, and creates appropriate situation-dependent costs that the planner can use to create more efficient paths.

We present the representations of the path planner and the navigation module in Section 2. In Section 3 we describe the execution trace and how we use it to identify training data (planner arc traversals) from the probabilistic information. In Section 4 we present the learning mechanism we use to create the mapping from situation features ($\mathcal{F}$) and arc traversals ($\mathcal{E}$) to arc costs ($\mathcal{C}$). In Section 5 we present some experimental results. Finally, in Section 6 we summarize the main points of the paper.

# 2 Architecture & Representation

Xavier is a mobile robot being developed at CMU [18]. It is built on an RWI B24 base and includes bump sensors, a laser light striper, sonars, a color camera and a speech board. The software controlling Xavier includes both reactive and deliberative behaviours, much of which can be classified into five layers: Obstacle Avoidance, Navigation, Path Planning, Task Planning (ROGUE), and the User Interface.

In this paper, our concern is to improve the reliability of path planning. Learning opportunities are extracted from the execution traces created by by the navigation module. Figure 2 shows how our algorithm fits into the framework of the Xavier architecture. The path planner uses a decision-theoretic A* algorithm that operates on a topological map with metric information [6]. Navigation is done using Partially Observable Markov Decision Process Models (POMDPs) [19]. Our learning algorithm affects the arc costs of the topological map so that the planner will select plans with a higher efficiency. The challenge is to create variable costs depending on high-level features and to extract this information automatically from the robot's execution data.

One challenge of this work is to process vast amounts of uncertain, continuous navigation data. At no point in the robot's execution is the robot aware of where it *actually* is. It maintains a probability distribution, making it more robust to sensor and actuator errors, but making the learning problem more complex.

# 3 Execution Trace & Event Identification

The goal of learning in our system is to identify events ($\mathcal{E}$) during execution which do not meet expectations, and to then correlate situational features ($\mathcal{F}$) to those events. Events are then evaluated by a cost function ($\mathcal{C}$).

Events in any planner can be identified by asking the question: "*What will change the planner's behaviour?*" The designer can easily extract these learning opportunities from the model of the environment. For example, events in a task planner are learning opportunities related to task achievement, such as delivering an object or missing a deadline. Events for a network router are packets and messages being sent.

In ROGUE, we would like the learner to diagnose problems in the environment so that the path planner can predict and avoid areas which are difficult to navigate (and similarly, exploit areas that are easy to navigate). Problem events are therefore arc traversals that do not meet expectations. Improved cost estimates on these arc traversals will cause the planner to select more appropriate plans.

Arc cost estimates are primarily determined by traversal weight. The traversal weight of an arc can be calculated by examining the travel times of the arcs actually travelled. The difference between the *expected* travel time and the *actual* travel time yields an appropriate modifier for the traversal weight.

The kinds of features available in Xavier include speed, time of day, sonar observations (walls, openings), camera images (empty, crowded, cluttered,...), other goals, and the desired route. For example, travelling too fast past a particular intersection might lead to missing a turn. Images with lots of people might indicate difficult navigation.

An *execution trace* from the robot is generated by the navigation module. The trace includes:

- the features describing the situation,
- the sequence of actions executed by the robot, and
- the probability distribution over the Markov states at each time step.

In particular, an execution trace does *not* include arc traversals. We therefore need to use the Markov state distributions to identify arc traversals. This process can be described pictorially as in Figure 3. As the robot wanders down the corridor, it sees doors at time steps 6 and 8. The Markov state distribution changes as shown. In order to modify the arc cost estimates for the path planner, ROGUE needs to determine which arcs the robot travelled, and for how long.

Our algorithm examines the execution trace, identifies the most likely path that the robot traversed, and then identifies the corresponding planner arcs. It then maps situational features to the arc traversals to create situation-dependent costs.

### 3.1 Identifying the Most Likely Markov Sequence

Probabilistic navigation systems have a common feature in that, by maintaining probability distributions for all possible positions, they can quickly recover from sensor errors. This feature has a side-effect, however, in that the robot never knows where it *actually* is. At any given moment, it may believe that it is in one of several different locations.

However, the *action sequence* stored in the trace, together with the probability distribution, can be used to calculate the most likely sequence of Markov states that the robot passed through. The algorithm to calculate this sequence is known as Viterbi's algorithm [16]. However, Viterbi's algorithm *was not* designed for use in Markov models with additional uncertainty variables, such as time or length.

We developed an improvement to Viterbi's algorithm, called Multi/Markov Viterbi [7], which heuristically improves the estimate of the robot's trajectory for Markov models with structural changes that capture additional uncertainty. Multi/Markov Viterbi generates a sequence of Markov states that is a reasonable estimate of the robot's actual trajectory.

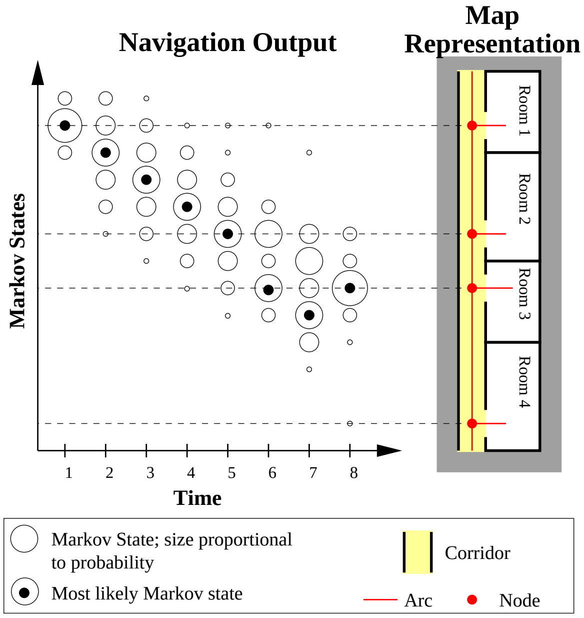



Figure 3: Extracting arc traversals from Markov state distributions.

### 3.2 Identifying the Planner's Arcs

Once a reasonable Markov sequence has been reconstructed, we need to identify which of the planner's arcs the robot traversed. Although it might appear to be a simple problem, the representation of the planner and of the POMDP are significantly different and the mapping is not direct.

The POMDP represents the world in a set of discrete square blocks. The path planner, on the other hand, represents the world in a set of arcs, where nodes correspond to topological junctions like doors and corridors. Figure 4 demonstrates the difference for a lobby area. Although these representations clearly make sense for each module, there is no direct correlation between the Markov states and the arcs.

This fact complicates the reconstruction of the arc path because a single Markov sequence may map to multiple arc sequences. Selecting the correct one is a challenging problem that we address with a greedy heuristic, based on expectation times [7].

a) Markov Representation    b) Planner Arc Representation

Figure 4: Different representations of a Foyer

### 3.3 Creating a Uniform Output

Once arc traversal events have been identified from the execution trace, updated costs need to be calculated.

The cost evaluation function, $\mathcal{C}$, yields an updated arc traversal weight. In our implementation, this weight is equal to the product of the *desired velocity* on that arc by the *actual time* spent traversing it, divided by the *modelled length* ($w = vt/l$). The cost represents the experienced difficulty of the arc traversal: if (i) if the robot travelled slower than desired, (ii) the robot were to travel along a sinuous path at the desired speed, or (iii) the modelled length of the arc were shorter than the actual length, then the cost will be higher than the default of 1.0.

The data is stored in a matrix format along with the cost evaluation and the environmental features observed when the event occurred. (Those environmental features which change during the traversal are averaged.) Table 2 shows a sampling from an *events matrix* generated by the system.

This collection of feature-value vectors is presented in a uniform format for use by any learning mechanism. Additional features from the execution trace can be trivially added; this matrix was recorded for the experiments described in Section 5, while sonar readings and other features were added for experiments involving the task planner [7].

## 4 Learning Algorithm

In this section we present the learning mechanism we use to create the mapping from situation features ($\mathcal{F}$) and events ($\mathcal{E}$) to costs ($\mathcal{C}$). The input to the algorithm is the events matrix described in Section 3.3. The desired output is situation-dependent knowledge in a form that can be used by the planner.

We selected *regression trees* [4] as our learning mechanism. Other learning mechanisms may be appropriate in different robot architectures with different data representations.

We selected an off-the-shelf package, namely S-PLUS [2], as the regression tree implementation. A regression tree is fitted for each arc using *binary recursive partitioning* where the data is successively split until data is too sparse or nodes are pure (below a preset deviance). Splits are selected to maximize the reduction in deviance of the node. Deviance of a node is calculated as $D = \sum(y_i - \mu)^2$, for all examples $i$ and predicted values $y_i$ within the node. Chambers & Hastie [5] discuss the method in more detail.

We prune the tree using *10-fold random cross validation*, in which a tree is built using 90% of the data, and then the remaining 10% of the data is used to test the tree, resulting in the relationship between tree size and misclassification rates. This calculation is done 10 times, each time holding out a different 10% of the data. The results are averaged, giving us the best tree size so as not to over-fit the data. The least important splits are then pruned off the tree until it reaches the desired size. Figure 5 shows a sample learned regression tree after pruning. This tree represents the situation-dependent arc costs for arc 208.

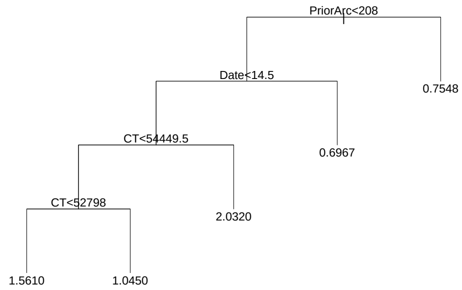



Figure 5: The pruned tree from arc 208 (an arc in corridor 2 of the exposition world presented in Section 5.1).

## 5 Experimental Results

We will present two sets of experiments. The first set involves a simulated world because it is an environment where the experiments can be tightly controlled. The second set was run on the real robot, validating the algorithm and the need for it.

### 5.1 Simulated World

Xavier has a simulator whose primary function is to test and debug code before running it on the real robot. The simulator allows software to be developed, extensively tested and then debugged off-board before testing and running it on the real robot. The simulator is functionally equivalent to the real robot: it creates noisy sonar readings, it has poor dead-reckoning abilities, and it gets stuck going through doors. Most of these "problems" model the actual behaviour of the robot, allowing code developed on the simulator to run successfully on the robot with no modification [13]. The simulator allows us to tightly control the experiments to ensure that the learning algorithm is indeed learning appropriate situation-dependent costs.

Figure 6 shows the world we used to test the algorithm: an exposition of the variety one might see at a conference. Figure 6a shows the simulated world, complete with sample obstacles (dark boxes in corridors). Figure 6b shows the topological map used by the path planning module, which does not contain obstacles.

The simulator of course has limited capabilities for dynamism: currently doors can only be opened and closed at the whim of the user; obstacles are static. For our experimental stage, we needed the robot to be operating in a dynamic world. We add dynamism by running each experiment in a *variation* of the map shown in Figure 6. The position of the obstacles in the simulated world changes according to the following schedule:

- corridor 2 always clear
- corridor 3 with obstacles
  - EITHER Mon, Wed, or Fri between (midnight and 3am)

| ArcNo | Weight | CT | Speed | PriorArc | Goal | Year | Month | Date | DayOfWeek |
|---|---|---|---|---|---|---|---|---|---|
| 233 | 0.348354 | 38108 | 34.998001 | 234 | 90 | 1997 | 06 | 30 | 1 |
| 232 | 0.264036 | 38105 | 34.998001 | 233 | 405 | 1997 | 06 | 30 | 1 |
| 196 | 3.762347 | 37816 | 34.998001 | 195 | 284 | 1997 | 06 | 30 | 1 |
| 246 | 0.552367 | 60099 | 34.998001 | 247 | 253 | 1997 | 07 | 07 | 1 |
| 201 | 1.002090 | 64282 | 34.998001 | 202 | 379 | 1997 | 07 | 07 | 1 |
| 134 | 16.549173 | 61208 | 34.998001 | 234 | 262 | 1997 | 07 | 09 | 3 |
| 238 | 0.640905 | 54 | 34.998001 | 130 | 379 | 1997 | 07 | 10 | 4 |
| 169 | 0.429588 | 39477 | 27.998402 | 168 | 31 | 1997 | 07 | 13 | 0 |
| 165 | 1.472222 | 8805 | 34.998001 | 164 | 379 | 1997 | 07 | 17 | 4 |
| 196 | 5.823351 | 3983 | 34.608501 | 126 | 253 | 1997 | 07 | 18 | 5 |
| 194 | 1.878457 | 85430 | 34.998001 | 193 | 262 | 1997 | 07 | 18 | 5 |

Table 2: Events Matrix. Each feature-value vector (row of table) corresponds to an arc traversal event ($\mathcal{E}$). The columns contain environmental features ($\mathcal{F}$) valid at time of the traversal. *Weight* is arc traversal weight determined by $\mathcal{C}$, *CT* is CurrentTime (seconds since midnight), *Speed* is velocity, in cm/sec, *PriorArc* is the previous arc traversed, *Goal* is the Markov state at the goal location, *Year, Month, Date, DayOfWeek* is the date of the traversal.

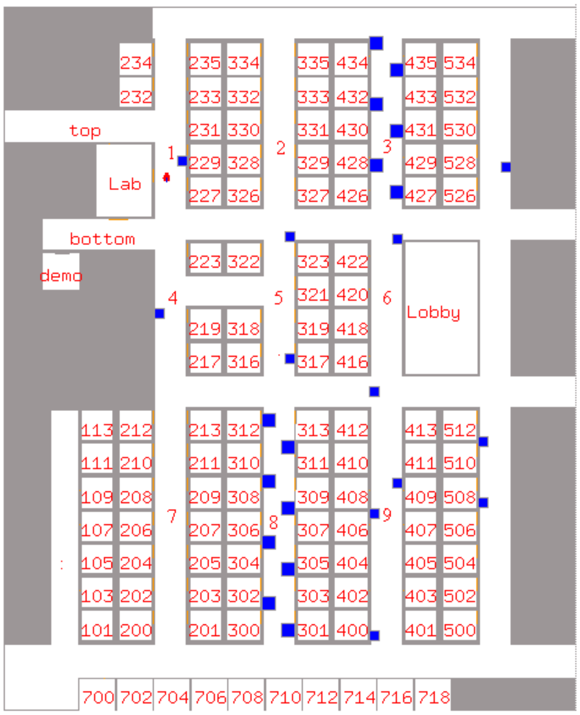
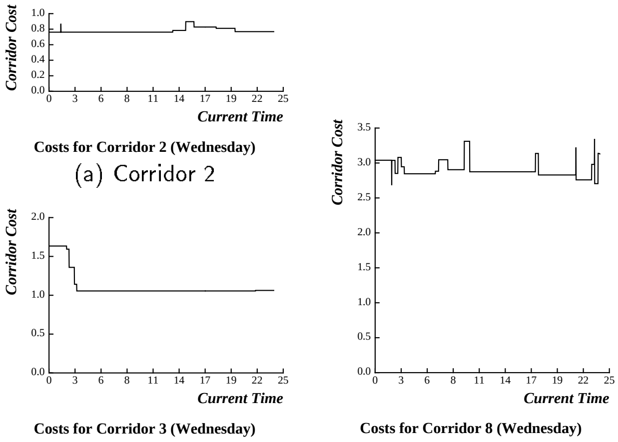



Figure 6: Exposition World, as seen by the simulator. The path planner does not know about the obstacles (small dark boxes).




(a) Corridor 2

(b) Corridor 3

(c) Corridor 8

Figure 7: Corridor cost (average over all arcs in that corridor) for Wednesdays.

  – OR one of the other days between (1 and 2am)

- corridor 8 always with obstacles
- remaining corridors with random obstacles (approximately 10 per map)

In each map, we ran a fixed path through the environment: from corridor 1 to booth 303 to 411 to 327 to 435 to 210, collecting the execution trace. (We used actual user requests for the experiments in Section 5.2.)

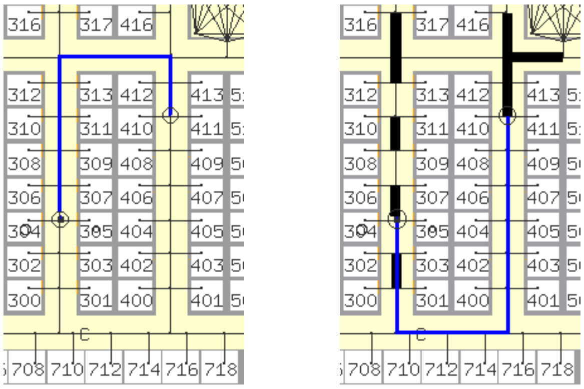
This environment allowed us to test whether the system would successfully identify:

- permanent phenomena (corridors 2 and 8),
- temporary phenomena (random obstacles), and
- patterns in the environment (corridor 3).

The events table was generated as described in Section 3, and then processed as described in Section 4.

Over a period of 2 weeks, 651 execution traces were collected. Almost 306,500 arc traversals were identified, creating an events table of 15.3 MB. The 17 arcs with fewer than 25 traversal events were discarded as insignificant, leaving 100 arcs for which the system learned trees. (There are a total of 331 arcs in this environment, of which 116 are doors, and 32 are in the lobby.) Trees were generated with as few as 25 events, and as many as 15,340 events, averaging 3060. Generated trees had an average size of 6.9 nodes (3.45 leaf nodes). The tree shown previously in Figure 5 was generated in this environment.

Figure 7 shows the cost, averaged over all the arcs in each corridor, as it changes throughout the day (a Wednesday). The system has correctly identified that corridor 3 is difficult to traverse between midnight and 3am, and during the rest of the day is close to default cost of 1.0. Corridor 8, meanwhile, is *always* well above default, while corridor 2 is slightly below default. These data show that Rogue is capable of learning both permanent phenomena and patterns in the environment. Corridor 4, meanwhile, had an average cost of 1.13, hence demonstrating that Rogue can



(a) Default Path (b) New Path

Figure 8: (a) Default path (when all corridor arcs have default value of 1.0). (b) New path (when corridor arcs have been learned) on Wednesday 01:05am; note that the learned expensive arcs have been avoided (arcs with cost > 2.50 are denoted by very thick lines).

differentiate different types of phenomena.

Figure 8 illustrates the effect of learning on the path planner. The goal is to have the system learn to avoid expensive arcs (those with many obstacles). Figure 8a shows the path normally generated. Figure 8b shows the path generated by the planner after learning; note that the expensive arcs have been avoided.

Table 3 shows the total *weight* × *length* values for several routes, using the learned costs to evaluate both the default path and the new path. The new path is consistently better than the default path.

The extensive data we have collected and illustrated here demonstrates that our system successfully learns situation-dependent arc costs. It correctly processes the execution traces to identify situation features and arc traversal events. It then creates an appropriate mapping between the features and events to arc traversal weights. The planner then correctly predicts the expensive arcs and creates plans that avoid difficult areas of the environment.

## 5.2 Real Robot

The second set of data was collected from real Xavier runs. Goal locations and tasks were selected by the general public through `http://www.cs.cmu.edu/~Xavier`, Xavier's web page. The robot operates in the fifth floor of our building, part of which is shown in Figure 9. For Xavier, the most challenging region of its environment is in the lobby of our building. The lobby contains two food carts, several tables, and is often full of people. The tables and chairs are extremely difficult for the robot's sonars to detect, and the people are (often malicious) moving obstacles. As a result, navigating through the lobby is challenging and expensive for the robot. During peak hours (coffee and lunch breaks), it is virtually impossible for the robot to efficiently navigate through the lobby.

This data has allowed us to validate the need for the algorithm in a real environment, as well as to test the predictive ability given substantial amounts of noise.

We show the incremental nature of Rogue through an analysis of the data at two snapshots in time.

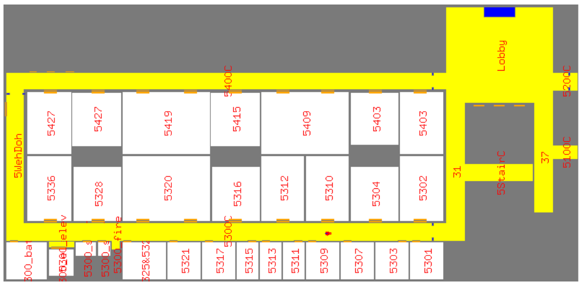


Figure 9: Robot's Map (half of the 5th floor of our building)

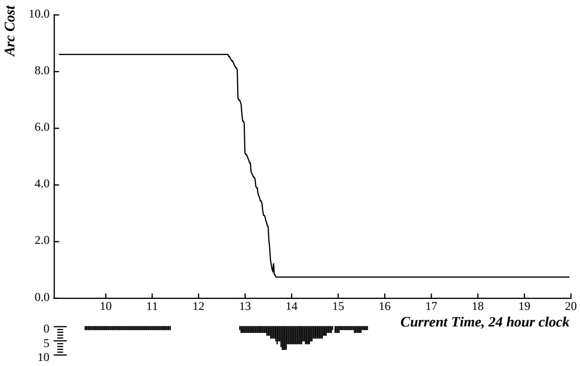



Figure 10: Costs for Wean Hall Lobby on Wednesdays. Graph generated 31 July 1997. The histogram below the graph indicates volume of training data; most data was collected between 1:30pm and 2:45pm.

### 5.2.1 31 July 1997

Over a period of 3 months, 17 robot execution traces were collected. These traces were run between 9:35 am and 3:40pm and varied from 10 minutes (0.35 MB) to 82 minutes (14 MB).

More than 15,000 arc traversal events were recorded, for a total of 766 KB of training examples. Trees were learned for 89 arcs from an average of 169 traversals per arc, yielding an average tree size of 10.2 nodes (5.1 leaf nodes).

Figure 10 shows the average learned costs for all the arcs in the lobby for Wednesdays (51 of the 89 arcs are in the lobby). Values differentiated by other features were averaged[1]. Below the average cost graph, a histogram shows how much training data was collected for each time step during the day.

The system correctly identified lunch-time as a more expensive time to go through the lobby. The minimal morning data was not significant enough to affect costs, and so the system generalized, assuming that morning costs were reflected in the earliest lunch-time costs.

### 5.2.2 31 October 1997

To test Rogue's incremental learning ability, an additional 42 traces were collected, yielding a total of 59 execution traces. 72,516 arc traversal events were recorded, for a total of 3.6 MB of data in the events matrix. Trees were

[1]Note that since the robot operates in a less controlled environment, many features may affect the cost of an arc. In the exposition world, other features do not appear in the trees.

| Start Room | Goal Room | Situation | Default Path Default Costs | Default Path Learned Costs | New Path Learned Costs | Percent Improvement |
|---|---|---|---|---|---|---|
| 231 | 303 | Mon, 15:40 | 4503.50 | 6481.96 | 5969.99 | 8% |
| 303 | 411 | Mon, 15:40 | 2908.00 | 6753.12 | 3768.66 | 44% |
| 411 | 327 | Mon, 15:40 | 3343.00 | 5438.67 | 5438.67 | 0% |
| 327 | 435 | Mon, 15:40 | 2683.00 | 2759.07 | 1274.97 | 55% |
| 435 | 210 | Mon, 15:40 | 4969.50 | 6502.58 | 5595.47 | 14% |
| | | | *Total:* | 27423.43 | 22047.76 | 20% |
| 231 | 303 | Wed, 01:00 | 4503.50 | 6433.49 | 5586.48 | 13% |
| 303 | 411 | Wed, 01:00 | 2908.00 | 6250.80 | 3768.66 | 40% |
| 411 | 327 | Wed, 01:00 | 3343.00 | 5002.09 | 5002.09 | 0% |
| 327 | 435 | Wed, 01:00 | 2683.00 | 8902.85 | 1280.35 | 86% |
| 435 | 210 | Wed, 01:00 | 4969.50 | 12351.17 | 5305.65 | 57% |
| | | | *Total:* | 38940.40 | 20943.23 | 46% |
| 231 | 303 | Thu, 01:00 | 4503.50 | 6432.49 | 5586.18 | 13% |
| 303 | 411 | Thu, 01:00 | 2908.00 | 6090.72 | 3768.67 | 38% |
| 411 | 327 | Thu, 01:00 | 3343.00 | 4842.02 | 4842.02 | 0% |
| 327 | 435 | Thu, 01:00 | 2683.00 | 3447.87 | 1280.34 | 63% |
| 435 | 210 | Thu, 01:00 | 4969.50 | 6896.18 | 5305.66 | 23% |
| | | | *Total:* | 27709.28 | 20782.87 | 25% |

Table 3: Path length calculation for a variety of paths under three different situations. We show the default estimate of path length, evaluate the default path with the learned costs, and show the length of the path that A* finds with the learned costs. Finally, we show the percent improvement in path length between the default path and the new path.

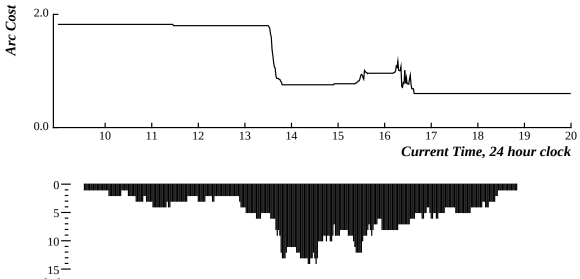



Figure 11: Costs for Wean Hall Lobby on Wednesdays. Graph generated 31 October 1997. The histogram below the graph indicates volume of training data; most data was collected between 1pm and 6pm.

learned for 115 arcs from an average of 631 traversal events per arc (min 38, max 1229). Data from nine arcs were discarded because they had fewer than 25 traversal events. The average tree size was 16.3 nodes (8.1 leaf nodes).

Figure 11 shows the average learned costs for all the arcs in the lobby on Wednesdays. Values differentiated by other features were averaged. Below the average cost graph, a histogram shows how much training data was collected for each time step during the day.

This graph shows that the system is still confident that the lobby is expensive to traverse during the lunch hour. The greater volume of data reduced the cost estimate (hence eliminating most of the noisy data), but the morning data was still not sufficient to reduce the morning cost. To our surprise, the graph shows a slightly higher cost during the late afternoon. Investigation reveals that it reflects a period when afternoon classes have let out, and students come to the area to study and have a snack. The incremental nature of the approach means that as Rogue collects additional training data, it will more accurately reflect the patterns of the environment.

This data shows that our learning approach is useful and effective, even in an environment where many of the default costs were tediously hand tuned by the programmers. The added flexibility of situation-dependent arc costs increases the reliability and efficiency of the overall robot system.

# 6 Conclusion

We have presented a learning robotic system with the ability to learn from its own execution experience. Our learning approach uses predictive features of the environment to create *situation-dependent costs* for the arcs in the topological map used by the path planner to create routes for the robot. These costs, represented as learned regression trees, reflect the patterns detected in the environment, and effectively diagnose the situations the robot encounters. The planner then knows which areas of the world to avoid (or exploit), and can therefore find the most efficient path for each particular situation.

Rogue processes the execution trace generated by the navigation module to extract events relevant for learning. The execution trace contains a massive, continual stream of probabilistic, low-level data. Our system abstracts this information to reconstruct the route, and the arcs that the robot traversed through the environment. Each of these arc traversals is then evaluated, and the cost recorded along with the situational features existing at the time of the traversal event.

This data is then correlated by a regression tree algorithm to create situation-dependent arc costs for each of the traversed arcs. Finally, the path planner uses the updated costs to create efficient, situation-dependent routes for the robot. The algorithm works incrementally, improving the situation-dependent costs after each run of the robot.

We presented empirical data from both a controlled, simulated environment as well as the real robot. This data demonstrates the effectiveness and utility of our developed approach. Rogue effectively refines the domain model to capture the patterns of the dynamic world.

We view the approach as being relevant in many real world domains. Situation-dependent rules are useful in any domain where actions have specific *costs*, *probabilities*, or *achievability criteria* that depend on a complex definition of the state. Planners can benefit from understanding the patterns of the environment that affect task achievability. Possible applications include learning action probabilities for conditional planners, and learning sensor and actuator reliabilities or costs for planners or execution monitors (for example, Xavier's POMDP navigation module could benefit from situation-dependent probabilities for the sensor readings).

This situation-dependent knowledge can be incorporated into the planning or monitoring efforts so that tasks can be achieved with greater reliability and efficiency. Haigh [7] describes an implementation of the approach for the robot's task planner. Situation-dependent features are an effective way to capture the changing nature of a real-world environment.

Beyond using situation-dependent costs for planning, possible future research directions include customizing the environment and autonomously extracting features from the execution trace.

Humans already customize environments greatly for children and the handicapped. It seems only appropriate to also consider customizing the environment for our future co-workers: robots. Areas of the environment that the learning system identifies as being difficult or expensive to achieve tasks could be modified to improve system performance.

The area of possible research with perhaps with the greatest potential for improving the performance of learning systems, is to automatically decide what features to add to the data set. Klingspor *et al.* [Klingspor *et al.*, 1996] have already designed techniques for learning high-level feature concepts from low-level data. It remains an open research problem to automatically incorporate those features into learning.

# References


[1] C. Baroglio, A. Giordana, M. Kaiser, M. Nuttin, and R. Piola. Learning controllers for industrial robots. *Machine Learning*, 23:221–249, 1996.

[2] R. A. Becker, J. M. Chambers, and A. R. Wilks. *The New S Language.* (Pacific Grove, CA: Wadsworth & Brooks/Cole), 1988. Code available from `http://www.-mathsoft.com/splus/`.

[3] S. W. Bennett and G. F. DeJong. Real-world robotics: Learning to plan for robust execution. *Machine Learning*, 23:121–161, 1996.

[4] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* (Pacific Grove, CA: Wadsworth & Brooks/Cole), 1984.

[5] J. M. Chambers and T. Hastie. *Statistical models in S.* (Pacific Grove, CA: Wadsworth & Brooks/Cole), 1992.

[6] R. Goodwin. *Meta-Level Control for Decision-Theoretic Planners.* PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1996. Available as Technical Report CMU-CS-96-186.

[7] K. Z. Haigh. *Learning Situation-Dependent Planning Knowledge from Uncertain Robot Execution Data.* PhD thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, February 1998. Available as technical report CMU-CS-98-108.

[8] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[Klingspor *et al.*, 1996] Volker Klingspor, Katharina J. Morik, and Anke D. Rieger (1996). Learning concepts from sensor data of a mobile robot. *Machine Learning*, 23:305–332.

[9] S. Koenig and R. G. Simmons. Passive distance learning for robot navigation. In *Machine Learning: Proceedings of the Thirteenth International Conference (ICML96)*, pages 266–274, 1996.

[10] D. Kortenkamp and T. Weymouth. Topological mapping for mobile robots using a combination of sonar and vision sensing. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 979–984, 1994.

[11] J. Lindner, R. R. Murphy, and E. Nitz. Learning the expected utility of sensors and algorithms. In *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pages 583–590, 1994.

[12] T. M. Mitchell. *Machine Learning.* (New York, NY: McGraw Hill), 1997.

[13] J. O'Sullivan, K. Z. Haigh, and G. D. Armstrong. *Xavier.* Carnegie Mellon University, Pittsburgh, PA, April 1997. Manual, Version 0.3, unpublished internal report. Available via `http://www.cs.cmu.edu/~Xavier/`.

[14] D. J. Pearson. *Learning Procedural Planning Knowledge in Complex Environments.* PhD thesis, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, 1996. Available as Technical Report CSE-TR-309-96.

[15] D. A. Pomerleau. *Neural network perception for mobile robot guidance.* (Dordrecht, Netherlands: Kluwer Academic), 1993.

[16] L. R. Rabiner and B. H. Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, pages 4–16, January 1986.

[17] W.-M. Shen. *Autonomous Learning from the Environment.* (New York, NY: Computer Science Press), 1994.

[18] R. Simmons, R. Goodwin, K. Z. Haigh, S. Koenig, and J. O'Sullivan. A layered architecture for office delivery robots. In *Proceedings of the First International Conference on Autonomous Agents*, pages 245–252, 1997.

[19] R. Simmons and S. Koenig. Probabilistic robot navigation in partially observable environments. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1080–1087, 1995.

[20] M. Tan. *Cost-sensitive robot learning.* PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1991. Available as Technical Report CMU-CS-91-134.

[21] S. Thrun. A Bayesian approach to landmark discovery and active perception for mobile robot navigation. Technical Report CMU-CS-96-122, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1996.