Route Planning and Learning from Execution

Karen Zita Haigh
khaigh@cs.cmu.edu
(412) 268-7670

Jonathan Richard Shewchuk
jrs@cs.cmu.edu
(412) 268-3778

Manuela Veloso
mmv@cs.cmu.edu
(412) 268-8464

School of Computer Science
Carnegie Mellon University
Pittsburgh PA 15213-3891

Abstract

There is a variety of applications that can benefit from the ability to automatically find optimal or good routes from real maps. There have been therefore several efforts to create and use real maps in computer applications. However, for the purpose of route planning, maps cannot be seen as static and complete, as there are dynamic factors and missing information that affect the selection of good routes, such as time of the day, traffic, construction, one versus multi-lane roads, residential areas, etc. In this paper, we describe our method for route planning and dynamic update of the information available in a map. We show how we do route planning by reusing past routing cases that collectively form a good basis for generating a new routing plan. We briefly present our similarity metric for retrieving a set of similar routes. The metric effectively takes into account the geometric and continuous-valued characteristics of a city map. We then present how the planner produces the route plan by analogy with the retrieved similar past routes. Finally we show how a real traversal of the route is a learning opportunity to refine the domain information and produce better routes. We illustrate our algorithms on a detailed online map of the city of Pittsburgh containing over 18,000 intersections and 25,000 street segments.

Introduction

We are interested in having a computer generate good routes from an online representation of a map. The fact that this task is to be performed by a computer on a map of a *real* city raises several issues, including:

- The representation of the map: Independent from which representation we choose to describe the map, it cannot be static. There will be constant changes to make, which correct, add, or delete facts from the map description.
- Route planning: The problem involves finding a sequence of paths to visit multiple locations in the map. This problem is well known for its complexity. Approximate solutions are sufficient, given that at execution time we will meet unexpected situations in the real-world.

The path finding task is dynamic and complex and therefore learning is necessary. We believe that the best way to do this is by taking advantage of previous route planning and execution experience.

Our approach for incorporating learning with planning and execution within this real-world task consists of:

- Accumulating route planning episodes in a case library, so that we can reuse previously visited routes, and avoid unnecessary repetitive search.
- Using execution experience to identify characteristics of particular routes that are not represented in the map and update the map to reflect them.
- Using experience gained from changing plans during execution failures to acquire an understanding of when particular replanning techniques are applicable.

In this paper, we show the representation of the real map, present the methods we have developed to store and retrieve previously visited paths, and describe the learning mechanisms to update the map from execution experience.

We are implementing our algorithm within the context of the PRODIGY planning and learning system [2]. Our algorithm consists of a planning part and a learning part. Results to date show that the planning half of the algorithm markedly improves plan quality and reduces total planning time [7; 8]. We illustrate the use of the algorithm within the domain of robotics path planning using a complete map of Pittsburgh [1].

Representation of Domain Theory

The domain theory used by PRODIGY consists of a knowledge base (in this case, the map) and a set of operators — rules used to model changes in state.

The Map

The map used in our work is a complete map of Pittsburgh containing over 18,000 intersections and 5,000 streets comprised of 25,000 segments. (An entire street is comprised of several segments corresponding to city blocks.)

The map is represented as a planar graph with the edges indicating street segments and the nodes indicating intersections. Associated with the intersections are the (x, y) coordinates of the intersection and a list of segments which meet at that intersection. Associated with each street segment is the name of the street containing it, and a range of numbers corresponding to building numbers on that block. In addition, there are several addresses of restaurants and shops in the city. Figure 1 shows a short excerpt from our files.

The representation, although it describes the map com-

```
(intersection-coordinates  i0 631912 499709)
(intersection-coordinates  i1 632883 485117)
(segment-length  s0 921 )
(segment-street-numbers  s0 4600 4999)
(segment-intersection-match  s0 i0 )
(segment-intersection-match  s1 i0 )
(segment-street-mapping  0 S_Craig_St)
(address Great_Scot 413 S_Craig_St)
```

Figure 1: Excerpt from map database.

pletely in terms of which streets exist, lacks in several areas important to an executing system. In particular, it does not indicate:

- direction of one-way streets,
- illegal turning directions,
- overpasses and other nonexistent intersections,
- traffic conditions,
- construction, and
- road quality, determined by factors such as number of lanes, surface (cobblestone, tarmac), and neighbourhood designation (residential, business).

This lack of information will lead to many situations in which the system needs to learn.

The Operators

Domain operators can be conceptually abstracted into three main categories:

- those that deal with navigation;
- those that deal with higher-level map information, such as restaurants and shops; and
- those that deal with higher-level goals, for example `(buy milk)` or `(mail letter)`.

Navigation operators examine information in the map regarding intersections, connected streets, and goal coordinates. These operators also examine any learned domain knowledge, such as one-way streets and construction. Map information operators examine the addresses within the map, placing restaurants, shops and street addresses at specific coordinates so the navigation operators can construct a path. Goal-oriented operators associate high-level goals to appropriate locations; for example `(buy milk)` could be achieved at any of several grocery stores.

Route Planning by Analogy

We apply case-based reasoning (CBR) [9; 11] methods to path planning because it enables a planner to reuse *cases* — solutions to previous, similar problems — to solve new problems. This technique allows us to take advantage of prior experience.

A CBR planning system has to first identify cases that may be appropriate for reuse, and then modify them to solve the new problem. We follow the case reuse strategy developed by Veloso [20] within the framework of PRODIGY. The replay technique involves a closely coupled interaction between planning using the domain theory (static knowledge of the world) and modification of similar cases.

The cases are derivational traces of both successful and failed decisions in past planning episodes, as well as the justifications for each decision. The case replay mechanism of PRODIGY/ANALOGY involves a reinterpretation of the justifications for case decisions within the context of the new problem, reusing past decisions when the justifications hold true, and replanning using the domain theory when the transfer fails.

For example, if a case indicated that the next move was to cross a bridge, PRODIGY/ANALOGY would check that it believes the bridge is still functional before committing to the same move. If the bridge was unusable, it would use its domain knowledge to find an alternate route over the river.

Note that the change in status of the bridge is an example of a changed situation in the real world that the system is able to learn and integrate into its domain knowledge – we do not alter cases since that could be computationally expensive in a large case library. In addition, a case might become relevant again at a later date.

If the case library is empty or there are no cases applicable in the new situation, the system will use its current domain knowledge to construct a viable solution, and incrementally expand the case library.

The final portion of our algorithm is the learning phase, described in more detail below.

Storing and Retrieving Cases

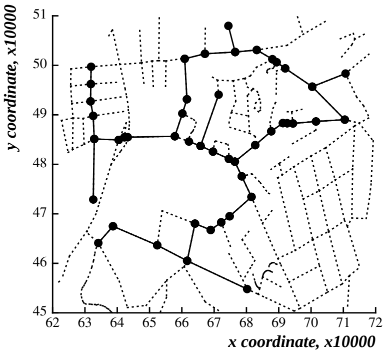
In order for the case identification phase to be efficient, the planning system must have a clear and easy method to store and subsequently retrieve past information. The following subsections describe our method for storing routes. More detailed information regarding case retrieval (including runtime and efficiency results) can be found in a previous paper [8].

Case Representation and Indexing

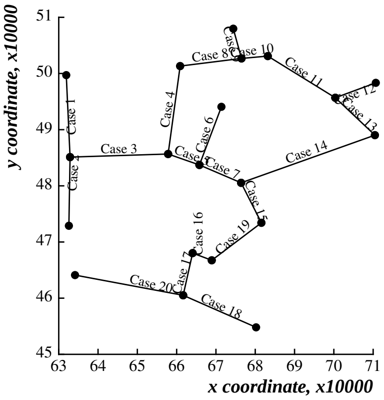
When PRODIGY generates a plan, the detailed derivational trace of the solution produced is stored as a single case, or broken into several pieces and stored as a series of cases. Once we start executing our plans, the representation of each case will also include a detailed description of the situations encountered at execution time, including explanations of any errors that occurred and all replanning that was done to correct the problems.

For our path planning domain, each case is also approximated by a line segment in a two-dimensional graph, and line segments are allowed to intersect only at their endpoints. This graph acts as an index into the case library so that cases can be easily retrieved.

When PRODIGY generates a plan that intersects existing cases, the plan and the cases it intersects are broken into smaller cases at the intersection points to maintain these constraints. The resulting graph, which we call the *case graph*, is illustrated in Figure 2b. Figure 2a is a map. Solid line segments are previously visited streets; dotted segments are unvisited streets. Figure 2b shows the abstract manner in which these paths are stored in the CBR indexing file. Note that Case 20 oversimplifies the path, but the bend in the road



(a) Map.



(b) Case graph representation of map.

Figure 2: Marked streets and intersections in the map (a part of the total Pittsburgh map) are locations visited during previous planning. Straight line approximations are used to create the representation by cases. Several case segments may together describe one route, and several streets may be contained in one case segment.

would not change the final routing (since there are no intersections along the route), so this abstraction is acceptable.

These abstractions are heuristically generated. When a route is planned, we break it up into pieces which approximate straight lines. Each of these pieces becomes a segment in the case graph. When a sinuous route has few intersections it is abstracted into a single straight line. This heuristic is not always guaranteed to be correct, but the planner can compensate.

Similarity Metric and Retrieval

Identifying cases relevant to the new problem is done by the use of a *similarity metric*, which estimates the similarity of cases to the problem at hand. An ideal metric might:

- take into account the relative desirability of different cases;
- suggest how multiple cases may be ordered in a single new solution; and
- identify which part(s) of a case are likely to be relevant.

Finding a similarity metric that is both effective and fast is a difficult task for the researcher. It is sufficiently difficult that many existing CBR systems identify neither multiple cases nor partial cases at all. The metric developed by Haigh and Shewchuk [8] effectively takes into account the geometric and continuous-valued characteristics of a city map, and can generate multiple and partial cases.

Suppose we undertake to plan a route on our map from some initial location i to some goal location g. Although we want to reuse cases, we are willing to traverse unexplored territory to avoid long, meandering routes. It is important to find a reasonable compromise between staying on old routes and finding new ones. Hence, we assign each case an *efficiency* value β, which is a rough measure of how much a known case should be preferred to unexplored areas. β is an indicator of the "quality" of a road, and is independent of the road's length.

In the map domain, the efficiency of a particular case might depend on such factors as road conditions or traffic. The efficiency satisfies $\beta \geq 0$, and may vary from case to case; low values of β correspond to more desirable streets. Values of $\beta > 1$ correspond to undesirable streets.

We assume that the *cost* of traversing an unknown region of the plane is equal to the distance travelled, while the cost of traversing a known case is equal to β times the distance travelled. Define a *route* to be a continuous simple path in the plane. A route may include several case segments (or parts thereof), and may also traverse unexplored regions. Assign each route a cost which is the sum of the costs of its parts.

The problem of finding a good set of cases is reduced to a geometric algorithm in which one finds an *optimum route* (that is, a route with the lowest cost) from the initial vertex i to the goal vertex g in the case graph. Our algorithm is an approximation algorithm that finds a close to optimal route.

The case segments found in the shortest route are returned to the planner, which creates a detailed plan using the cases for guidance.

Our similarity metric consists of several steps.

Delaunay Triangulations First we form a *Delaunay triangulation* [3] which will allow us to take advantage of *locality*, the principle that one is most likely to travel from vertices to other nearby vertices. This saves computational effort because we can ignore interactions between distant vertices and segments. A correct algorithm might need to consider all interactions.

Delaunay triangulations have several desirable properties:

- They provide a structure that makes it possible to quickly determine the edge costs of the case graph;
- Local modifications of the triangulation can easily be made; and
- They form a good approximation of which vertices are closest to each other. Take for example Figure 3.

This figure shows a small set of points and the Delaunay

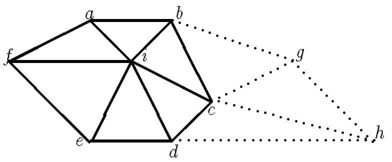
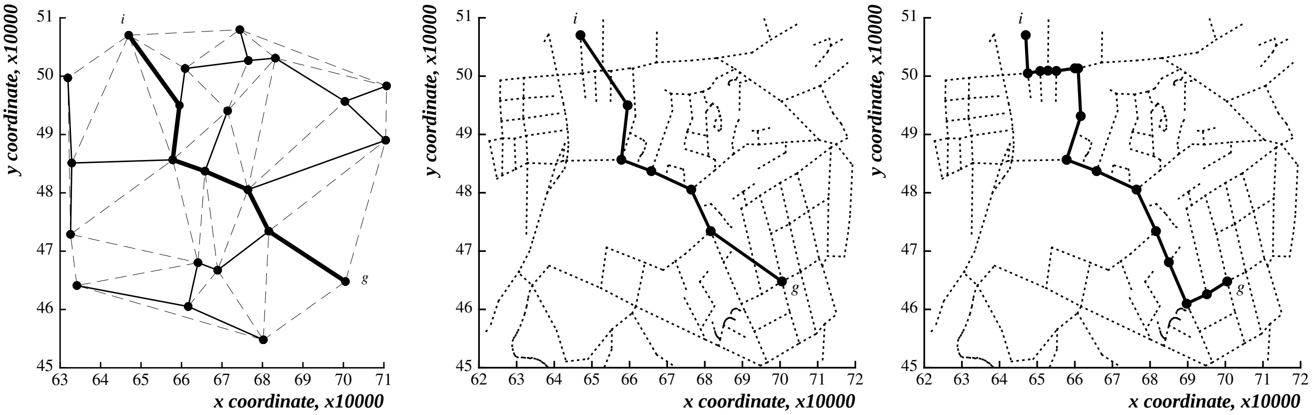


Figure 3: A set of points and their triangulation



(a) Dijkstra path on triangulation (b) Dijkstra path on map (c) Final path

Figure 4: (a) The path found by Dijkstra's algorithm in the Delaunay triangulation; solid lines represent case edges, dashed lines represent triangulation edges, thick lines represent the path. (b) Dijkstra's path superimposed on the real map. (c) Dijkstra's path modified by PRODIGY to conform to real world constraints.

triangulation of those points. Imagine that each of the points a through h are endpoints of case segments, and that i is the initial point. In each direction around i, the triangulation indicates what case is closest to i. Any cases outside the hexagon centered at i (such as those involving g and h) are further away, and are less likely to be directly connected to i in the final solution path. It is these more distant cases that we ignore in our heuristic.

The conforming Delaunay triangulation of the set of cases from Figure 2b, plus a new initial point and goal point, is shown in Figure 4a.

Edge Costs Once the triangulation has been formed, we calculate the edge costs for the edges in the triangulation.

Where two vertices are connected by a case segment, the edge cost is simply the value β for that case times the Euclidean distance between the vertices. Under other conditions, the calculation is not always so simple. To see why, consider Figure 5.

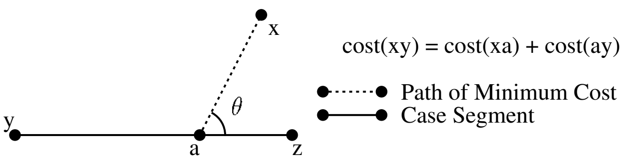


Figure 5: Finding the route of minimum cost.

Imagine that $\overline{yz}$ is a highway, and you wish to navigate your way from x to y. It is faster to take the highway for part of the route than to go directly from x to y. The best place to merge with the highway is at point a. (Note that there may not be an entrance at a in the real world; it is PRODIGY's responsibility to find some legal merge point close to a.)

In this figure, $\overline{yz}$ is a case segment. The optimum route between x and y is $\langle \overline{xa}, \overline{ay} \rangle$, for some point a that depends on the value of β. The cost of the optimum route is equal to length($\overline{xa}$)$+\beta\times$length($\overline{ay}$). Let θ represent the angle $\angle xaz$; the position of a is computed from the fact that $\theta = \cos^{-1}\beta$. In the limiting case where $\beta = 0$, $\overline{xa}$ is perpendicular to $\overline{yz}$.

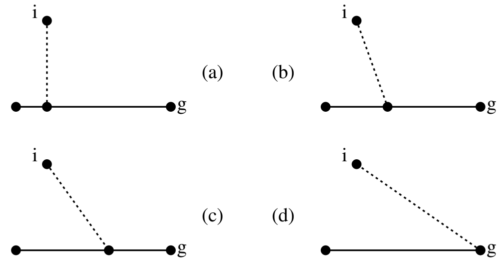


Figure 6: Sample paths for routes between i and g with varying β values of the cases. (a) $\beta = 0.00$. (b) $\beta = 0.25$. (c) $\beta = 0.50$. (d) $\beta \geq 1.00$.

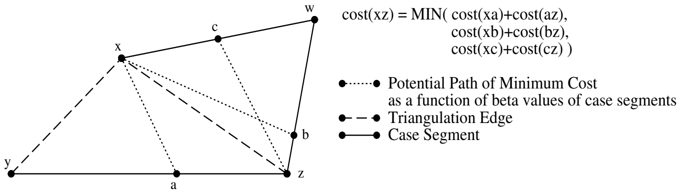


Figure 7: Cost calculation for edge (x, z) contained in two triangles

Figure 6 shows some sample paths generated for different β values on a case.

We assign a cost to each edge (x, y) of the triangulation by considering a number of possible routes between x and y, and taking the route with minimum cost. The first route we consider is a straight line between x and y. Then we need to consider all routes between x and y that use case segments. Taking advantage of the locality principle, we consider only the case edges that occur in the triangles in the vicinity of (x, y). Usually we need only examine the two triangles adjacent to (x, y). Figure 7 illustrates some of the possibilities. To determine the cost of edge (x, z), only a small number of alternative routes need be considered, taking only constant time.

For each edge (x, z) in the triangulation, we record which simple route from x to z had the minimum cost so that the entire route (and set of cases) can be reconstructed later.

Shortest Paths The final step of the similarity metric is to treat the triangulation as a graph and use Dijkstra's shortest path algorithm [4] to find the optimum route through the triangulation.

Figure 4 shows a path chosen by Dijkstra's algorithm between the labelled initial (i) and goal (g) points and some β value assigned to each case in the triangulation. Figure 4a shows the path through the triangulation and Figure 4b shows the same path – as it is handed to PRODIGY/ANALOGY – superimposed on the real map. Note that the path given to PRODIGY/ANALOGY would not be executable in the real world because it traverses several regions where there are no streets. It is the planner's job to knit this information together into a plan, taking into account details such as one-way streets and illegal turns that cannot be resolved by the geometric algorithm (this process is described in the section *Route Planning by Analogy*). Figure 4c shows the path after it has been modified by the planner to conform to constraints not known by the similarity metric.

Traversal of the Route

After the set of cases is retrieved and modified into a workable plan, the plan needs to be translated into a language that the executor understands. Once the plan has been executed – as well as possible – learning can take place: successes and failures need to be identified, domain knowledge modified, β values of cases need to be adjusted, and the execution trace needs to be added to the case library.

Language Translation

Before being executed, the fine-grained plan must be translated into a language that can be understood by the executor. Plan-level commands must be converted into execution-level commands.

We have been exploring this translation process within the framework of Xavier [13], an autonomous robot built for indoor tasks that has competed in recent AAAI robot competitions. The Task Control Architecture [14], an operating system for robots, is used as a basis for the communication between the planner and the robot controller. It provides the mechanisms needed to control goals and actions as well as to monitor the environment.

Under this architecture, a plan-level command such as `turn-right` would be turned into:

```
tcaExecuteCommand( CTR_TURN, 90.00).
```

All movements must be expressed in terms of precise distances or (x, y) goal coordinates (which can be relative to the current position or absolute within the world). Xavier also has the ability to recognize landmarks such as doors or bar codes.

The planner must also decide at what speed to send commands to the executor. The planner cannot simply give the entire plan to Xavier and expect it to be executed — since the real-world is constantly changing, there are enough potential failures along the route that blindly executing the plan is unlikely to lead to success. Therefore PRODIGY must break the plan into small pieces, each of which has an easy-to-identify success condition. We are currently exploring methods of breaking down the plan in this fashion.

PRODIGY/ANALOGY can use the time spent executing each step to do contingency planning for more probably failures.

Learning from Execution

Failure Identification When the system does not detect a success within a given time-limit, the system starts trying to identify a failure and categorize it according to severity. Some failures will involve simple replanning to avoid temporary obstacles, others will require modification of the β values in the case-base, and still others will require modifications to the map knowledge.

For example, if a street does not exist when the map believes one does, then the permanent domain knowledge needs to change. If rush-hour traffic, weather or major construction caused the failure, then β values of the cases involved in creating the failed plan need to be changed (see below). If a road is blocked for a parade, then the system needs to only replan to achieve its goals.

In addition, failures that occur during execution are themselves learning opportunities. When replanning is needed, learning mechanisms (such as cases or control rules) can accumulate knowledge on efficient repair strategies for the plan. This knowledge will help improve contingency planning and other replanning situations in the future.

Since the learning part of the system has not yet been implemented, we are currently manually telling the system whether the failure was general or not. In the future, we will explore methods for automatic identification and classification of failures.

Once the system has identified and internalized the learned knowledge, it will try to generate a new plan under the new constraints.

Adjustment of β Values As experience with particular routes increases, so should the system's knowledge and confidence in using the route. When a case is first placed in the case library, it is assigned an initial β value dependent on established 'norms' of driving (currently arbitrary: we suspect that they will also be learnable).

Each time the route is used in future episodes, the system can adjust this belief up or down, depending on the outcome of execution: if the new execution of the segment revealed that it was better than the original β value, then β will be lowered slightly, and if the execution revealed that the segment was worse, β will be raised slightly.

Since certain failures suggest that β values are not the same at all times, there is also a method for storing multiple β values for one case, and then for choosing which β value is applicable when planning next occurs. For example, we might want to have a higher β value at rush hour:

if (15:00 $\leq$ current_time $\leq$ 18:00)
then β = 1.5
else β = 0.6

1. Given a new problem, find a set of similar cases, using stored β values. .... *Fully implemented.*
2. Modify case(s) into new plan. .... *Fully implemented.*
3. Execute plan. .... *Not implemented yet.*
4. If execution of plan is successful:
   - Add new case to library. .... *Partially implemented.*
   - Assign appropriate β values. .... *Fully implemented.*

   Otherwise:
   - Identify reason for failure. .... *Not implemented yet.*
   - Modify world knowledge as applicable.
     - modify domain knowledge (permanent or long-term changes) .... *Not implemented yet.*
     - modify β values (specific situations) .... *Partially implemented.*
     - leave database unchanged (for temporary failures)
   - Add any successful parts of plan to case library .... *Not implemented yet.*

Figure 8: Our integrated planning and learning route-planning algorithm. Current status of the various stages are marked in italics.

Other possible comparisons might involve specific dates (eg. construction), season or weather (eg. impassability due to potholes, snow or mud), or direction (eg. one-way streets).

Adding Cases to the Library At the end of any problem solving episode, successful solutions are added to the case library and to the indexing file. We can efficiently update the indexing file (the triangulation) to reflect any newly learned cases. Haigh and Shewchuk [8] describe this process in more detail. This incremental behaviour is one of the benefits of using Delaunay triangulations.

Initial β values are assigned to each new case, reflecting the system's belief in the usefulness of the case.

If a particular route involves several turns or is in some other way very sinuous, the case will be indexed by a set of straight-line segments that approximate the curve. If each of these segments is derived from new planning (and not from cases), then each segment will have the same β value.

Those portions of the plan that were derived from cases (and therefore were already in the indexing graph) will have this new path added to their association lists, and potentially have a slight modification to their β values as described above.

We are currently investigating the issue of how much redundant information to add to the case library: it could be more work to retrieve and merge several non-overlapping cases than to store several cases with overlapping information thereby reducing merge cost. The tradeoff between retrievel and modification costs have been discussed at length elsewhere [10; 19].

We are also investigating the effect of adding failure cases to the library. These cases might aid in creating contingency plans (*i.e.* by identifying situations where contingency plans were necessary) and in determining efficient replanning techniques (*i.e.* by identifying which techniques related to which failures).

Similarly, there might be occasions to "forget" cases, such as cases that are rarely used, or cases in which an important piece of domain knowledge changed, making a case irrelevant. Factors that may influence the decision to forget a case might include a high β value, the length of time the case remains unused, and the likelihood that the case will become relevant again (eg. temporary construction).

Related Work

Most robotics path planners (eg. Dyna [15], COLUMBUS [18], Xavier [6], NavLab [16; 17]) don't remember paths or their quality, and typically use shortest path, dynamic programming or decision theoretic algorithms to determine routes.

Long Ji Lin [12] used *experience replay* within the framework of reinforcement learning to speed up the acquisition of knowledge about the world. The agents learned to survive in a dynamic world, and had to do basic path planning in order to achieve goals. The experiences were integrated into the domain knowledge of the agent, but not stored for future reference.

ROUTER (developed by Goel *et. al* [5]) is the only other case-based route-planning system the authors are aware of. It however has an extremely simple retrieval and modification algorithm, considerably reducing the transfer rate of prior experience. In addition, it does not remember quality of paths in an attempt to improve case retrieval.

Conclusion

In this paper, we have described our approach to route planning using a complete online map of the city of Pittsburgh. We motivated the need for an integrated planning and learning system. We presented a similarity metric that takes advantage of the geometric characteristics of the map and returns a set of similar previously traversed routes. We briefly discussed how the system creates a new route plan by integrating information from retrieved cases and planning from the map description and the routing operators. Finally, we introduced a learning algorithm that uses execution experience to update the map. We capture map changes into case parameters that are used by the similarity metric. Therefore, the system will incrementally retrieve better cases and generate more accurate route plans with experience.

The entire integrated planning and learning algorithm is summarized in Figure 8.

Our future work includes implementing each of the unfinished portions of the algorithm, creating a system that can deal with situations when all the goals are not initially known, and comparing the behaviour of the system (in each stage of development) to both classical planners as well as to those created by by humans.

Any system that interacts with the real-world will have to deal with a changing environment. We hope that our system will form a good basis for future exploration in this area.

References

[1] Bernd Bruegge, Jim Blythe, Jeff Jackson, and Jeff Shufelt. Object-oriented system modeling with OMT. In *Proceedings of the OOPSLA '92 Conference*, pages 359–376. ACM Press, October 1992.

[2] Jaime G. Carbonell, Craig A. Knoblock, and Steven Minton. PRODIGY: An integrated architecture for planning and learning. In K. VanLehn, editor, *Architectures for Intelligence*. Erlbaum, Hillsdale, NJ, 1990. Available as Technical Report CMU-CS-89-189.

[3] L. Paul Chew. Constrained Delaunay triangulations. *Algorithmica*, 4:97–108, 1989.

[4] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[5] Ashok Goel, Michael Donnelan, Nancy Vazquez, and Todd Callantine. An integrated experience-based approach to navigational path planning for autonomous mobile robots. In *Working notes of the AAAI Fall Symposium on Applications of Artificial Intelligence to Real-World Autonomous Mobile Robots*, pages 50–61, Cambridge, MA, October 1992.

[6] Richard Goodwin and Reid Simmons. Rational handling of multiple goals for mobile robots. In J. Hendler, editor, *Artificial Intelligence Planning Systems: Proceedings of the First International Conference (AIPS92)*, June 1992.

[7] Karen Haigh and Manuela Veloso. Combining search and analogical reasoning in path planning from road maps. In *Case-Based Reasoning: Papers from the 1993 Workshop*, pages 79–85, Washington, D.C., July 1993. AAAI Press. Available as Technical Report WS-93-01.

[8] Karen Zita Haigh and Jonathan Richard Shewchuk. Geometric similarity metrics for case-based reasoning. In *Case-Based Reasoning: Working Notes from the AAAI-94 Workshop*, pages 182–187, Seattle, WA, August 1994. AAAI Press.

[9] Kristian J. Hammond. Case-based planning: A framework for planning from experience. *Cognitive Science*, (14):385–443, 1990.

[10] M. T. Harandi and S. Bhansali. Program derivation using analogy. In *Proceedings of IJCAI-89*, pages 389–394, 1989.

[11] Janet L. Kolodner. *Case-Based Reasoning*. Morgan-Kaufmann Publishers, Inc., San Mateo, CA, 1993.

[12] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3-4):293–321, 1992.

[13] Joseph O'Sullivan and Karen Zita Haigh. *Xavier*. Carnegie Mellon University, Pittsburgh, PA, July 1994. Manual, Version 0.1, unpublished.

[14] Reid Simmons, Richard Goodwin, Chris Fedor, and Jeff Basista. *Task Control Architecture: Programmer's Guide*. Carnegie Mellon University, School of Computer Science / Robotics Institute, Pittsburgh, PA, 7.7 edition, May 1994.

[15] Richard S. Sutton. Planning by incremental dynamic programming. In *Machine Learning: Proceedings of the 8^{th} International Workshop*, pages 353–357. Morgan Kaufmann, 1991.

[16] Charles Thorpe and Jay Gowdy. Annotated maps for autonomous land vehicles. In *1990 IEEE International Conference on Systems, Man and Cybernetics Conference Proceedings*, Los Angeles, CA, November 1990.

[17] Charles E. Thorpe, editor. *The CMU Navlab*. Kluwer Academic Publishers, Boston, MA, 1990.

[18] Sebastian B. Thrun. Exploration and model building in mobile robot domains. In *Proceedings of the IEEE International Conference on Neural Networks*, San Francisco, CA, March 1993.

[19] Manuela M. Veloso. Variable-precision case retrieval in analogical problem solving. In *Proceedings of the 1991 DARPA Workshop on Case-Based Reasoning*. Morgan Kaufmann, May 1991.

[20] Manuela M. Veloso. *Learning by Analogical Reasoning in General Problem Solving*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1992. Available as technical report CMU-CS-92-174.