To appear in the Journal of Experimental and Theoretical AI

Exploiting Domain Geometry in Analogical Route Planning

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

Jonathan Richard Shewchuk
jrs@cs.cmu.edu
(412) 268-7553

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

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

Running Head: Exploiting Geometry in Analogical Route Planning

Abstract

Automated route planning consists of using real maps to automatically find good map routes. Two shortcomings to standard methods are (i) that domain information may be lacking, and (ii) that a "good" route can be hard to define. Most on-line map representations do not include information that may be relevant for the purpose of generating good realistic routes, such as traffic patterns, construction, and one-way streets. The notion of a good route is dependent not only on geometry (shortest path), but also on a variety of other factors, such as the day and time, weather conditions, and perhaps most importantly, user-dependent preferences. These features can be learned by evaluating real-world execution experience.

These difficulties motivate our work on applying analogical reasoning to route planning. Analogical reasoning is a method of using past experience to improve problem solving performance in similar new situations. Our approach consists of the accumulation and reuse of previously traversed routes. We exploit the geometric characteristics of the map domain in the storage, retrieval, and reuse phases of the analogical reasoning process. Our route planning method retrieves and reuses multiple past routing cases that collectively form a good basis for generating a new routing plan. To find a good set of past routes, we have designed a similarity metric that takes into account the geometric and continuous-valued characteristics of a city map. The metric evaluates its own performance and uses execution experience to improve its prediction of case similarity, adaptability and executability. We present the replay mechanism that the planner uses to produce a route plan based on analogy with past routes retrieved by the similarity metric. We use illustrative examples and show some empirical results from a detailed on-line map of the city of Pittsburgh, containing over 18,000 intersections and 25,000 street segments.

Contents

1 Introduction

Automated planning in real-world domains is well-known to be a complex research task, in particular due to their inherent dynamic properties, uncertainty, incompleteness of domain specifications, and difficulty in evaluating solution quality. Route planning using a real-world map and considering real execution is a challenging example of such domains.

On-line representations of maps are usually devoted to describing the geometry of the domain, facilitating a careful algorithmic use of path distances. However, an ideal automated real-world route planner should also examine key properties of the world. In particular, map representations do not usually include information that may be relevant for route planning, such as traffic patterns, construction, one-way streets, "no left turn" signs, time of day, weather, or a particular user's driving preferences. The route planning task demands the ability to react to changing conditions as well as the ability to learn from experience while forming and executing plans.

This article presents in detail a route planning system designed for a real-world environment, and demonstrates the system using a real-world city map. The complete system was first presented by Haigh & Veloso (1995), and employs a case retrieval algorithm first presented by Haigh & Shewchuk (1994). Our system has been tested using a real city map with the ultimate goal of being used as guidance systems for private vehicles and autonomous robotic vehicles.

The basis for our approach is the use of analogical reasoning, as an instance of case-based reasoning (Schank 1982; Hammond 1990; Kolodner 1993), integrated with generative planning. The approach enables a planner to reuse *cases*—solutions to previous, similar problems—to help solve new problems. Therefore the system can accumulate user-tailored solutions that have been validated by real-world execution experience. Although many case-based reasoning systems try to capture the human analogical reasoning process, the psychological plausibility of our *algorithm* is secondary to our goal of producing an effective system. We aim at creating a *solution* that human users find useful and convenient. Because our system learns user preferences and responds to external changing conditions, it creates situation-dependent user-tailored routes, thereby increasing the likelihood of actually being used.

We use PRODIGY/ANALOGY (Veloso 1994) as the substrate for our route planning implementation. PRODIGY/ANALOGY is the analogical reasoner of the PRODIGY planning and learning system (Carbonell *et al.* 1990; Veloso *et al.* 1995). Our system, built atop this substrate, creates route plans through analogical reasoning, presents them for execution, and learns from experience by the following process:

- it accumulates route planning episodes (cases) in a case library so that it can reuse previously visited routes and avoid unguided search;
- it retrieves a set of previously traversed routes, chosen by examining the geometric features of the domain, that collectively form a good basis for a new routing plan;
- it merges these cases into a new route, adding new steps where necessary;
- it uses feedback from the user to modify its case indexing file so that future case retrieval operations respect the user's preferences.

The geometric continuous-valued features of the route planning domain inspire our innovative approaches to case retrieval and replay.

Traditional methods for evaluating case similarity are unable to deal with continuous values, where domain knowledge may describe distance, time or cost. Generally, automated systems simply discretize continuous-valued inputs, thereby reducing the knowledge transfer. As an alternative to PRODIGY/ANALOGY's default similarity metric, we have developed a geometric similarity metric that takes advantage of domain geometry to perform case selection efficiently. It considers symbolic similarity combined with the geometric information to retrieve multiple and/or partial relevant case fragments, considerably increasing the transfer rate of experience.

PRODIGY/ANALOGY's general analogical replay mechanism reasons about multiple possible orderings for case merging. We take advantage of the geometry of the route planning domain to introduce a sequential merging technique. It uses the ordering suggested by the geometric similarity metric to knit this information together into a plan. It also revalidates choices and resolves details, such as one-way streets and illegal turns, that are not considered by the geometric algorithm.

Our system also presents a learning technique to incorporate user evaluations which are taken into account at retrieval time, allowing it to improve case selection and replay with experience.

The methods are not restricted to inherently geometric domains, but are applicable in domains that can be reduced to a continuous two-dimensional representation. For instance, Broos and Branting (1993) create schedules for a telescope by determining the desirability of alternative states in a state space by geometrically composing multiple training examples. In comparison, our method also allows partial information reuse by composing sections of cases, in addition to entire ones.

Our route planner takes advantage of both geometric information and qualitative knowledge. Feedback provided by the user about the quality of the generated routes allows the system to improve its evaluation of potential new routes, and thereby learn road qualities and adapt to users' preferences with time and experience. The contributions of our system include

- a similarity metric that exploits geometric features of the domain, and can identify both multiple and partially relevant cases,
- a case replay mechanism that takes the advice provided by the similarity metric and, with additional planning, knits the cases together to form a new route, and
- a learning algorithm that uses user evaluations of execution experience to update the case library and thus improve case retrieval.

The main activities of analogical reasoning are the generation, storage, retrieval, and replay of planning episodes. We describe in this article how we exploit the geometric characteristics of the routing domain at each stage of this process. We introduce our data representation in Section 2, describing the map, the domain operators, the cases and the case library. In Section 3 we present a similarity metric that can predict case *similarity* from high-level features, case *adaptability* through the geometric features, and case *executability* through user evaluations of performance. Our case replay mechanism, which exploits the information produced by the similarity metric, is described in Section 4. We discuss the mechanism by which the case library is updated with new cases in Section 5. We demonstrate the success of our system through several illustrative examples on a real-world city map in Section 6. We describe related work in Section 7 and draw conclusions in Section 8.

2 Domain and Data Representation

The information available to the planner consists of a map database, a set of operators (rules used to model changes in state or knowledge), and a case library.

2.1 The Map

The map used in our work is a complete map of Pittsburgh (Bruegge *et al.* 1992) containing over 18,000 intersections and 5,000 streets divided into 25,000 segments (Segments typically correspond to city blocks).

The map is represented as a planar graph, where the edges indicate street segments and the nodes indicate intersections. Associated with the intersections are the x, y coordinates of the intersection, and a list of segments that 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 which can be viewed graphically as we illustrate later in the article.

Although it includes all of the city's streets, the representation lacks much important information. In particular, it does not indicate

- direction of one-way streets,
- illegal turning directions,
- overpasses and other nonexistent intersections,
- traffic conditions,
- road quality (including speed limits and number of lanes), and
- temporary conditions, such as construction.

An effective system will need to learn much of this information through real-world execution.

```
(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 s0 S_Craig_St)
(address Union_Grill 413 S_Craig_St)
```

Figure 1: Excerpt from the map database.

2.2 The Operators

Our underlying planner, PRODIGY (Veloso *et al.* 1995), is a nonlinear state-space generative planning algorithm. PRODIGY uses operators that specify the domain actions' preconditions and effects.

The operators in the route planning domain 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 with specific locations; for example `(buy milk)` could be achieved at any of several grocery stores.

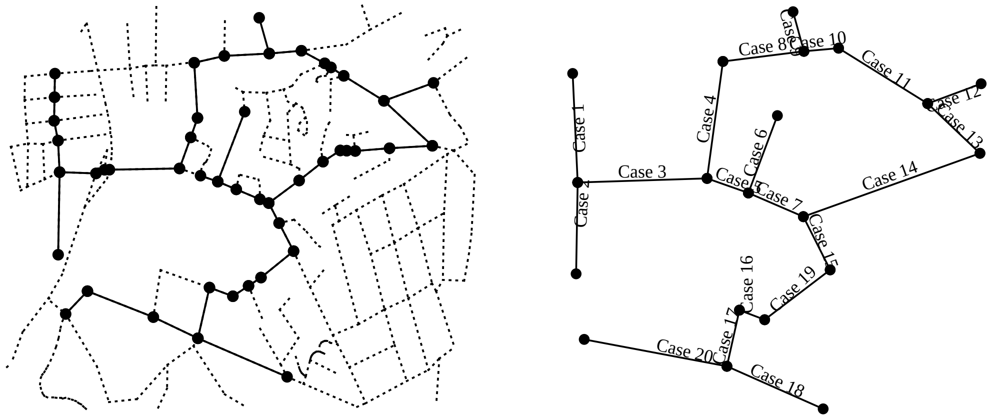
2.3 The Case Library

A case-based planner adapts prior stored planning episodes into solutions for new problems. Stored planning episodes are known as *cases*, and are stored in a *case library* along with an *indexing file* used for easy identification at retrieval time. For the case identification phase to be efficient, the planning system should have an effective method to store and subsequently retrieve past information. This section describes the case library and indexing file in our route planning domain.

Each time PRODIGY generates a plan, a detailed derivational trace of the planning solution is stored as a case. Failed search decisions are annotated so they may be avoided at reuse time. For our route planning domain, the representation of each case is extended to 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.

The case is indexed by relevant distinguishing features including time of day, day of week, and user. The existing PRODIGY/ANALOGY similarity metric would utilize these features (and only these featuers) at retrieval time. In order to exploit the geometric properties of the domain, we extend the indexing file with an abstraction of the problem's geometric properties. Each case is approximated by a set of straight line segments (*case segments*) in a two-dimensional graph, called the *case graph*. (The case graph is identical to what computational geometers call a *planar straight line graph.*) Along with the symbolic features, the case graph acts as the index into the case library so that cases can be easily retrieved.

For example, Figure 2 illustrates a set of cases and their abstraction in the case graph. Figure 2a is a map in which solid line segments represent previously visited streets, and dotted segments represent unvisited streets. Figure 2b shows the abstract manner in which these paths are stored in the indexing file. Case 20 oversimplifies the path, but the bend in the road would not change the final routing (since there are no intersections along the route), so this abstraction is acceptable. Note that several case segments may together describe one complete case route, and one case segment may index several cases.



(a) Map. (b) Case graph representation of map.

Figure 2: (a) Marked streets and intersections in the map (1% of the complete Pittsburgh map) are locations visited during previous planning. (b) Abstracted cases in the case graph.

These abstractions are heuristically generated. When a route is planned, it is broken 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. Case segments may intersect only at their endpoints; if a new case is added that crosses other cases, each intersecting case segment is split into smaller segments at the intersection points. The vertices of the case graph correspond to the endpoints of the case segments. The case graph does not represent perfectly the geometry of the roads represented by its cases, but the planner can compensate for the deviations.

The case graph effectively captures the geometric information of the cases. Along with the symbolic features, such as time and traffic, all relevant information is recorded and available for case retrieval.

3 A Geometric Similarity Metric

A *similarity metric* is a mechanism for identifying the cases (in a case library) that are most likely to be useful in creating a plan for a new problem. An ideal metric should

- take into account the relative desirability of different cases, where desirability includes an evaluation of task similarity, ease of adaptation and appropriateness of the final solution;
- suggest how multiple cases might be ordered in a single new solution; and
- identify which part(s) of a case is likely to be relevant.

Finding a similarity metric that is both effective and efficiently computable is a difficult task for the researcher.

It is sufficiently difficult that most of the existing case-based reasoning systems identify neither multiple cases nor partial cases. This is unfortunate, because in many domains, and in route planning in particular, the transfer rate of past experience is considerably reduced if only complete plans can be reused.

The domain of route planning adds an additional difficulty: it occurs in a continuous-valued state space. Traditional methods of ranking candidate analogs by evaluating their matching degree in terms of simple role substitution are not easily applied. Although PRODIGY/ANALOGY's default similarity metric is domain-independent, it relies on role substitution and counting of matching features, and assumes that specific constants can be parameterized and rematched to other instances at retrieval time. When dealing with continuous values, however, it is hard to find a representation that satisfies this assumption.

A common technique for dealing with continuous values is to simply create discrete ranges. Values are not considered similar if they fall in different ranges. As a result, knowledge transfer is considerably reduced.

The metric we designed for this domain, described below, effectively exploits the geometric information in this domain. It incorporates situation-dependent symbolic features to effectively evaluate the similarity of cases to the new problem, the ease of adapting cases to suit the new problem, and also the quality of the route at execution time. By abstracting away low-level information about the actual path of the route, our metric can detect cases that only partially match the current route planning problem, and it can also detect when several cases can be combined to form a solution to the current problem. It also identifies a possible ordering of those cases for use at replay time.

3.1 Definitions

Suppose we undertake to plan a route on our map from some initial location to some goal location. Although we want to reuse cases, we also want to avoid long meandering routes and are therefore willing to traverse unexplored territory. It is important to find a reasonable compromise between staying on old routes and finding new ones. Hence, we introduce a scheme for indexing cases.

We assign to each case an *efficiency* β, which is a rough measure of how much a known case should be preferred to unexplored areas under particular situations. β is an indicator of the "quality" of the road or roads that comprise a case segment as evaluated by the user, and is independent of the length of the case segment. β is usually a number between zero and one, though it may be larger; lower values of β correspond to more desirable streets. An efficiency of one implies that traversing a case is no better than traversing unexplored territory of the same length. Hence, we say 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. Most cases are preferable to an equivalent length of unexplored territory, and hence have an efficiency less than one; values of $\beta > 1$ indicate streets that should probably be avoided.

The efficiency of a case is not necessarily a constant; it may vary with the various factors that affect the quality of the corresponding roads, such as road conditions or traffic. For example, we might want to have a higher β value at rush hour:

$\beta =$ if (15:00 $\leq$ current_time $\leq$ 19:00)
    then return 1.5
  else return 0.6

Other possible comparisons might involve specific dates (*e.g.* construction), seasons or weather (*e.g.* impassability due to snow or mud), or direction (*e.g.* one-way streets).

The efficiency parameter is a key to our goal of using experience to learn a good mapping from the situation-dependent features to the value of β. Each time a route is executed, the user evaluates its quality. The system then incorporates this evaluation into the efficiency function of all the cases used to create the route. After a case has been executed a few times, the β value associated with it will start to reflect the quality of the case as experienced in the real world. The system will therefore be able to use β to balance the tradeoffs between using routes in unexplored territory and using known cases. Theorem B.4 demonstrates why distant cases will not be considered.

We 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. Recall that the cost of traversing all or part of a case is β times the distance traversed, and unexplored territory has an effective β of one.

We reduce the problem of finding a good set of cases to a geometric problem in which one must find an *optimum route* (that is, a route with lowest cost) from an initial vertex to a goal vertex in the case graph. The case segments found in the shortest route are returned to the planner, which creates a detailed plan using the cases for guidance. (To apply the geometric similarity metric to the entire detailed map, at the level of individual streets instead of case segments, is infeasible as we will discuss below.)

Our efficiency parameter captures symbolic situation-dependent features in the geometric case graph, and thus allows the retrieval procedure to select routes that are "good" with respect to both geometric and symbolic conditions.

The geometric problem we have defined can be solved by an algorithm due to Mitchell and Papadimitriou that solves the more general *weighted region problem* (Mitchell & Papadimitriou 1991); however, their algorithm is unnecessarily elaborate. Below we present two much simpler algorithms. The first finds the optimum route, albeit too slowly to be useful in practice; it is included here as a bridge to understanding the second algorithm, which is fast but may return a route that is slightly longer than the optimum route.

3.2 An Exact Algorithm to Find an Optimum Route

To solve the geometric problem, we transform it into a graph problem. We define G to be the complete graph (*i.e.*, every pair of vertices is connected by an edge) whose vertices are the vertices of the case graph (the endpoints of the cases) plus an initial vertex i and a goal vertex g. Each edge (x, y) of G is assigned a cost that represents the cost of a good simple route between x and y. Given this graph, one can solve the geometric problem by applying Dijkstra's shortest path algorithm (Dijkstra 1959; Cormen *et al.* 1990) to find the lowest-cost path from i to g in G. This procedure is very slow (because of the number of edges in the complete graph), but will yield an exact solution to the geometric problem.

The success of this procedure depends on assigning an appropriate cost to each edge of G prior to applying Dijkstra's algorithm. If two vertices are connected by a case segment, the cost of their connecting edge is (usually) simply β times the Euclidean distance between the vertices; if they are not connected, the calculation is not always so simple. To see why, consider Figure 3, and imagine that the case segment $\overline{yz}$ represents a highway. Suppose that it is faster to take the highway for part of the route than to drive a straight line from x to y. Even without any information about the roads between x and the highway, one can compute a geometric best point to merge with the highway, denoted a, that minimizes the cost of the route. There may not be an on-ramp at a in the real world; it is the planner's responsibility to find some legal merge point close to a on the complete map, after the similarity metric has finished its work. The planner can use the location of a as a rough indicator of what portion of the case (corresponding to the case segment $\overline{yz}$) should be used.

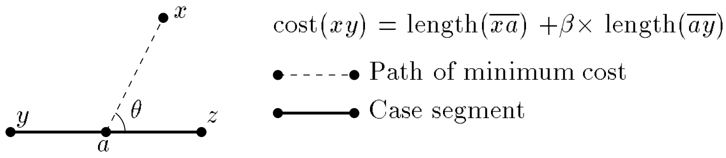


Figure 3: Finding the route of minimum cost.

In the figure, the optimum route between x and y is $\langle \overline{xa}, \overline{ay} \rangle$, where the point a depends on the value of β for the segment $\overline{yz}$ under the current conditions. The cost of this route is $\mathrm{length}(\overline{xa}) + \beta \times \mathrm{length}(\overline{ay})$.

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 4 illustrates some sample optimum paths for cases with different β values.

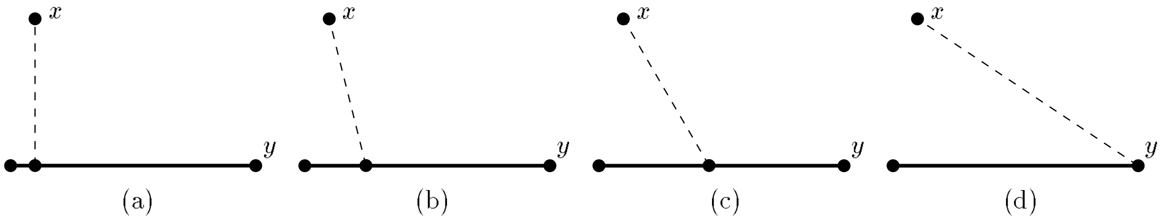
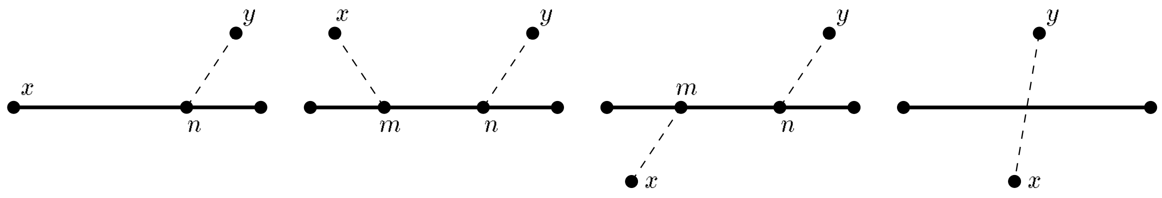


Figure 4: Examples of optimum routes between x and y for several β values. Solid lines are case segments, dotted lines are a portion of the optimum route. (a) $\beta = 0$. (b) $\beta = 0.25$. (c) $\beta = 0.5$. (d) $\beta \geq 0.83$.

Our algorithm assigns a cost to each edge (x, y) of G by considering a number of possible routes between x and y, and taking the route with minimum cost. The first route considered is a straight line between x and y. Then, for each case segment in the case graph, our algorithm considers the optimum route that uses that segment alone.

Several examples of shortest routes between x and y are demonstrated in Figure 5. Points m and n in examples (a), (b), and (c) are derived from the same equation, $\theta = \cos^{-1}\beta$. In example (d), however, the segment lying between x and y does not improve the optimum route, a straight line. In examples (b) and (c), if the vertices x and y are slid horizontally together past the position where the points m and n coincide, the shortest route becomes a straight line from x to y.



(a) (b) (c) (d)

Figure 5: Four examples of shortest routes between vertices x and y. The positions of the points m and n depend on the efficiency β of the case segment. (Solid lines are case segments. Dotted lines indicate the shortest route.)

We can now state our algorithm for finding a "good" set of case segments (including partial case segments) for the planner. For each edge (x, y) in G, we perform the edge cost calculation described above, and record which simple route from x to y had the minimum cost so that the entire route (and set of cases) can be reconstructed later. We find the shortest path from the initial vertex i to the goal vertex g in the graph G using Dijkstra's shortest path algorithm. We convert this shortest path into a shortest route by mapping each edge in the shortest path back to the corresponding simple route; the shortest route from i to g is the concatenation of these simple routes, which include information about which cases or partial cases they take advantage of. A proof that this route is indeed the geometrically shortest route from i to g under our cost function is given in Appendix B.

Unfortunately, the cost of this algorithm is prohibitive because one must calculate the cost between each pairing of vertices to be assured of the shortest route. (Though i and g may be far apart, the optimal route between them might be a straight line.) If the case graph has n case segments (and hence $\mathcal{O}(n)$ vertices), then the complete graph G has $\mathcal{O}(n^2)$ edges. It takes $\mathcal{O}(n)$ time to determine the cost of each edge (since we must consider each case), so the time complexity to determine the edge weights of G is $\mathcal{O}(n^3)$. The shortest path problem can be solved in $\mathcal{O}(n^2 \log n)$ time. Since the computation of edge weights is the asymptotically slowest part of the algorithm, the exact algorithm takes a total of $\mathcal{O}(n^3)$ time.

Fortunately, our exact algorithm can be relaxed to yield a much faster approximation algorithm that is not guaranteed to find the optimum route, but generally finds a route that is reasonably close to optimal.

3.3 A Fast Algorithm for Finding a Good Route

To derive a fast approximation algorithm, we take advantage of *locality*, the principle that an edge between nearby vertices is much more likely to be part of the optimal path than an edge between distant vertices, and a case segment is more likely to affect the cost of nearby edges than distant ones. This saves computational effort because we can ignore interactions between distant vertices and segments. The complete graph G is replaced by a type of sparse graph known as a *low-dilation* graph; a graph that has the property that the shortest path between any two vertices in the graph is at most a constant factor longer than a straight line. A variety of low-dilation graphs that could be used for this purpose are surveyed by Eppstein (1996). For our implementation, we have chosen a planar graph called a *Delaunay triangulation* (Delaunay 1934; Fortune 1992). Delaunay triangulations have several desirable properties.

- They provide a structure that makes it possible to quickly determine edge weights in the graph.
- Local modifications of the triangulation can be made inexpensively.
- Several free implementations are available to the public.
- They take into account interactions between vertices that are close to each other. Take for example Figure 6. This figure shows a small set of vertices and the Delaunay triangulation of those vertices. Imagine that each of the vertices a through h are endpoints of case segments, and that i is the initial vertex of the new problem. The shortest route from i to a vertex outside the polygon centered at i (such as g or h) is likely to utilize one of the intermediate cases (*e.g.* involving b, c, or d), so our heuristic ignores the possibility that it might connect directly.

There are three steps to our similarity metric. First form a Delaunay triangulation D, of the vertices of the case graph. Second, calculate edge costs for D. Finally use Dijkstra's shortest path algorithm to select a set of cases. Each of these steps is discussed in detail below.

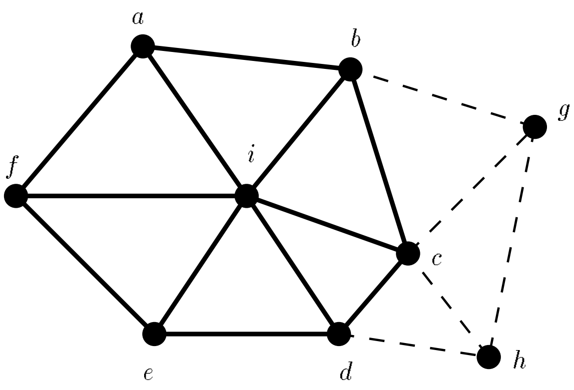


Figure 6: A set of vertices and their triangulation.

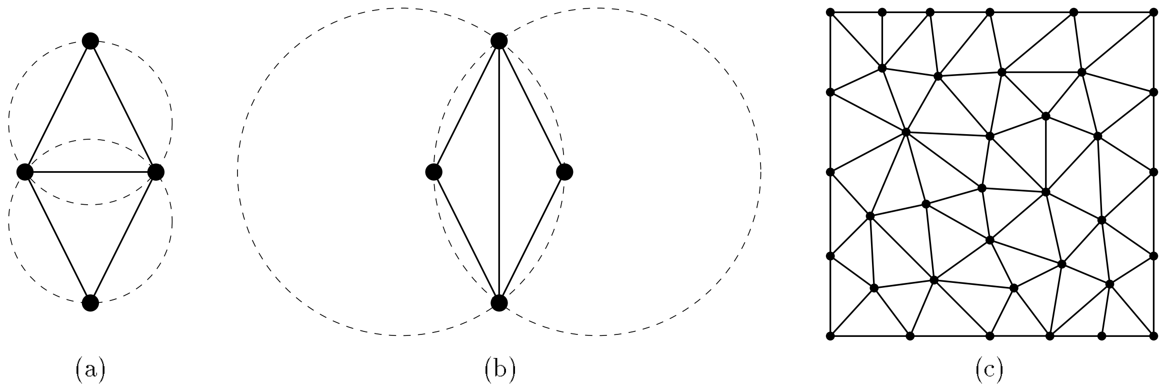


Figure 7: (a) A Delaunay triangulation of four vertices. The circumcircles of both triangles are illustrated. (b) Another triangulation (not Delaunay) of the same four vertices. (c) Example of a larger Delaunay triangulation. The Delaunay triangulation of a point set tends to connect vertices that are closer together.

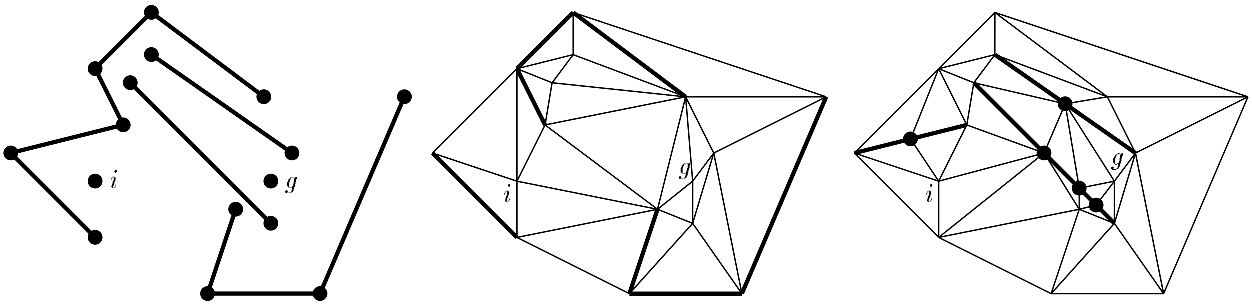
3.3.1 Step 1: Conforming Delaunay Triangulations

The Delaunay triangulation of a set of vertices is a triangulation in which there are no vertices inside any triangle's circumcircle. (The circumcircle of a triangle is the circle that passes through all its vertices.) For example, the triangulation in Figure 7a is Delaunay, but Figure 7b is not, because both triangles' circumcircles contain vertices.

Unfortunately, some of the case segments might not be represented as edges in the Delaunay triangulation. Our algorithm will require that every case segment is represented by an edge, or by a linear sequence of edges, in the triangulation. Hence, we add additional vertices to the triangulator's input to ensure that the Delaunay triangulation will represent all the case segments. This can normally be done by splitting in half any segment that would not otherwise appear in the Delaunay triangulation, adding a vertex at its midpoint. If the two resulting subsegments do not appear as edges in the triangulation, they are split recursively until bthe entire segment is represented by a sequence of edges in the triangulation. The final triangulation is known as a *conforming Delaunay triangulation* (Bern & Eppstein 1992), because it conforms to the case segments as well as the input vertices.

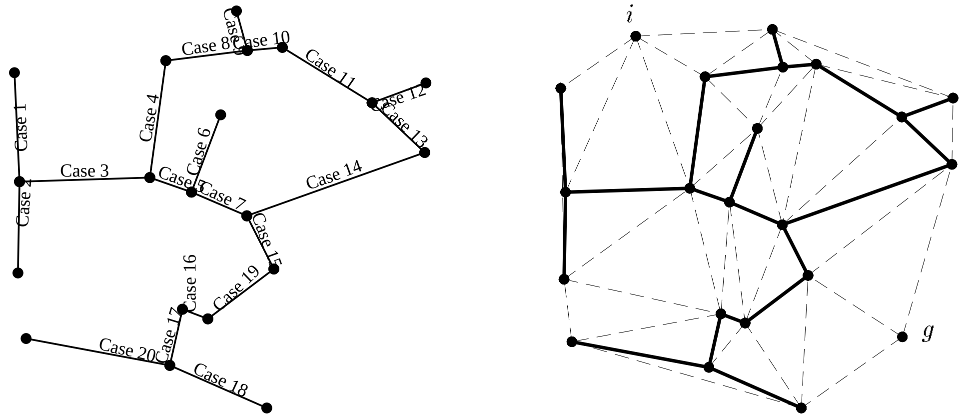
The process of creating a conforming triangulation is illustrated in Figure 8. Figure 8b is the Delaunay triangulation of the segment endpoints and the initial and goal vertices shown in Figure 8a; note that three of the segments are not represented. In Figure 8c, three additional vertices have been inserted, and the formerly missing segments now appear as the union of several edges in the final Delaunay triangulation.

In some cases where connected segments define a small angle, a more sophisticated method of placing additional vertices is needed; details are given by Ruppert (1995). This method of producing conforming Delaunay triangulations is implemented in the program Triangle (Shewchuk 1996), a triangular mesh gener-



(a) (b) (c)

Figure 8: (a) Case graph. (b) Delaunay triangulation of the vertices of the case graph. Some case segments are represented by edges of the triangulation, but some are not. (c) A conforming Delaunay triangulation of the case graph. Additional vertices have been inserted to ensure that all case segments are represented.



(a) Case graph. (b) Conforming Delaunay triangulation.

Figure 9: (a) Case graph (reproduced from Figure 2b). (b) Conforming Delaunay triangulation of case graph, for a particular initial vertex i and goal vertex g. Solid lines represent case edges, dashed lines represent other triangulation edges.

ator freely available through the Internet. We used an early version of Triangle to implement our similarity metric.

Figure 9 offers another example of a conforming Delaunay triangulation, based on an example from our Pittsburgh map database.

3.3.2 Step 2: Edge Costs

The cost calculation for an edge e is similar to that of the exact algorithm. However, the fast algorithm saves time by limiting the number of case segments that are considered when assigning an edge weight. We prove in Appendix B (proof B.4) that the only case segments that need to be examined are those that intersect the interior of the diametral circle of e (the unique circle having e as a diameter). We can use the triangulation as a search structure to find the case segments that fall in this circle, while ignoring all portions of the triangulation outside the circle (usually the vast majority of the map).

For our implementation, we have used a faster heuristic: when assigning a cost to an edge, only the two triangles containing that edge are examined. Case segments outside those two triangles are ignored. Figure 10 illustrates an example of such an edge cost computation. To determine the cost of edge (x, z), only a small number of alternative routes need be considered; in this case, the routes xz, xaz, xbz, and xcz.

$$\begin{aligned}\text{cost}(xz) = \min(&\text{length}(\overline{xz}),\\ &\text{cost}(\overline{xa}) + \text{length}(\overline{az}),\\ &\text{length}(\overline{xb}) + \text{cost}(\overline{bz}),\\ &\text{length}(\overline{xc}) + \text{cost}(\overline{cz}))\end{aligned}$$

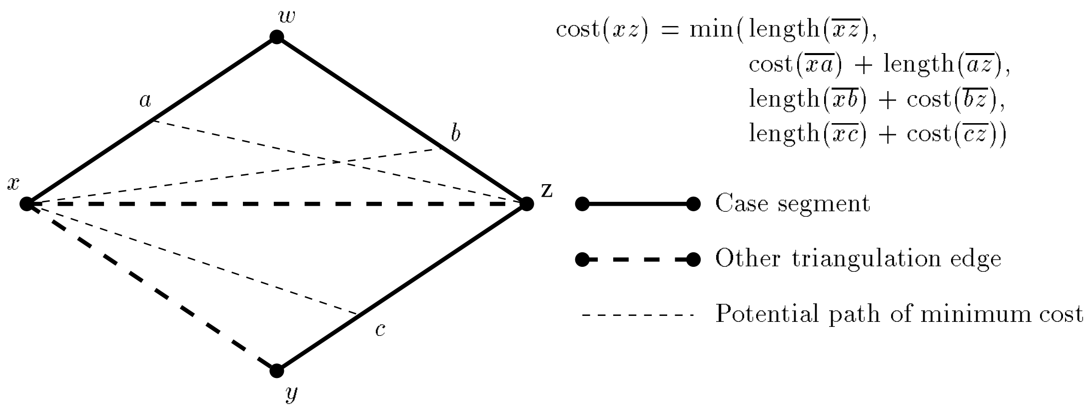


Figure 10: Example of cost calculation for an edge shared by two triangles. Note that the exact positions of a, b, and c depend on the β values of the case segments they fall on.

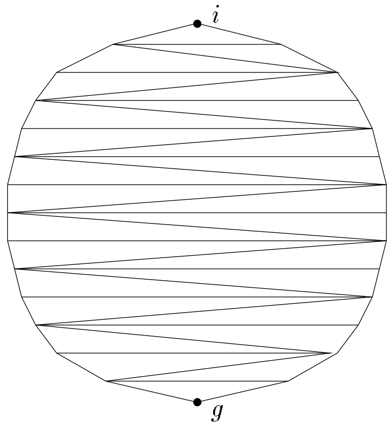


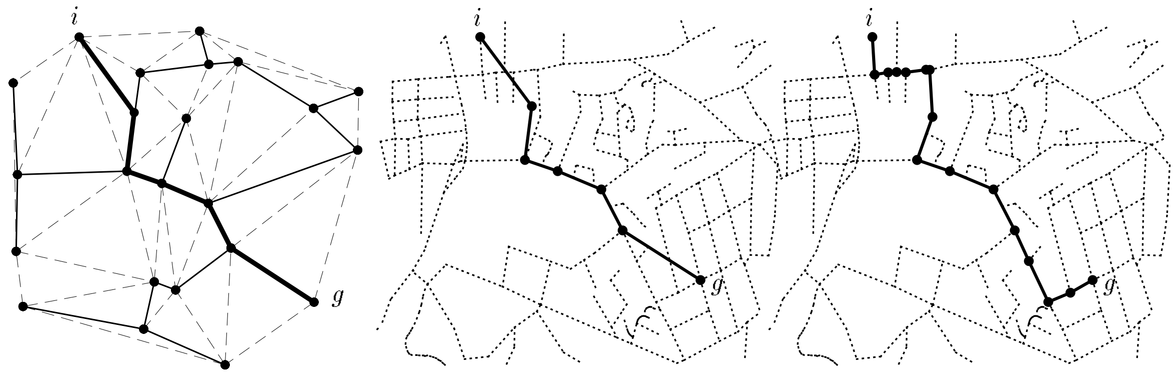
Figure 11: Example of a triangulation in which the shortest path is up to $\frac{\pi}{2}$ times as long as the true shortest route.

We compute the cost of each of these routes, and choose the route with smallest cost. Note that the direct route may be selected, even if there is no case directly connecting the x and z. Because there are never more than five routes to consider (one for each edge of the two adjacent triangles), each edge's cost is computed in constant time.

3.3.3 Step 3: Dijkstra's Shortest Path Algorithm

Once the triangulation and cost assignment steps are complete, Dijkstra's shortest path algorithm finds the optimum route through the triangulation, as in the exact algorithm. Our algorithm appears to approximate the best route well; it has never returned a route worse than 1.3 times longer than the optimum route, and often succeeds in finding the optimum route. We can also take some theoretical comfort in a result of Keil and Gutwin (1992) which states that the shortest path between two vertices in a Delaunay triangulation is never longer than 2.42 times the Euclidean distance between those vertices. Although this result is not directly applicable to our problem (because the cost of traversing a case segment does not generally equal the Euclidean distance traversed), it does give us additional confidence in the use of Delaunay triangulations in our approximation algorithm.

Another way to improve our confidence is to try to invent maps that befuddle our algorithm. The worst example we know is depicted in Figure 11. In this example, a large number of vertices fall on the surface of an ellipse whose height is infinitesimally larger than its width. Because the Delaunay triangulation of these points admits no shorter route than one about the perimeter of the ellipse, the shortest path between i and g may cost as much as $\frac{\pi}{2}$ times the distance between i and g. To minimize the likelihood of occurrences



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

Figure 12: (a) The path found by Dijkstra's algorithm in the Delaunay triangulation of Figure 9b; 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/ANALOGY to conform to real world constraints.

like this, we suggest a fast postprocessing step that can be applied after Dijkstra's algorithm. Search the shortest path for occurrences of two or more consecutive edges that do not coincide with case segments. As the cost of these edges is equal to their length, they should be contracted into a single straight edge.

Figure 12 shows a path chosen by Dijkstra's algorithm between the initial and goal vertices (i and g). Figure 12a shows the path through the triangulation; each segment in the path indexes one or more cases that the similarity metric determines as potentially helpful in solving the new problem. The planner, as presented in Section 4, uses the path to determine which cases to inspect, how to link them together, and where there are no appropriate cases and planning must be done from scratch.

Figure 12b shows the same path—as it is handed to PRODIGY/ANALOGY—superimposed on the real map. The path given to PRODIGY/ANALOGY is not yet executable in the real world because it traverses several regions where there are no streets. The planner's reuse algorithm processes this information to generate a viable plan. One might imagine applying our geometric algorithm to the entire fine-grained street map, rather than the case graph, and avoid using the planner altogether; however, there are several reasons why this is not feasible.

- Anomalies such as one-way streets, illegal turns, and overpasses are not easy to represent geometrically;
- The number of streets is much larger than the number of cases, and would create substantially more work;
- A graph representing individual streets rather than cases cannot easily represent prior experience with long routes and aid the selection of paths known to be reliable;
- Real road maps have minor temporary variations that are difficult to capture in a static map; and
- We wish to interleave planning with execution, so plans might have to be modified during driving. Therefore, we need a fine-grained method that is more flexible than Dijkstra's algorithm.

However, the geometric similarity metric, when applied at the coarse-grained level of case segments, greatly reduces the search space for the replay algorithm, which can therefore focus on resolving the issues listed above. Figure 12c shows the path after it has been modified by the planner to conform to constraints that the similarity metric is unaware of. This process is further described in Section 4.

3.3.4 Running Time

The running time of the similarity metric depends on the number of case segments in the case graph rather than the size of the map. We expect this number to be much smaller than the size of the map, which in our experiments contains 25,000 street segments. Running times for each part of our similarity metric, expressed in terms on the number of case segments n, are given herein.

Conforming Delaunay Triangulation. The Delaunay triangulation of the vertices of the case graph can be formed in $\mathcal{O}(n \log n)$ time (Lee & Schachter 1980; Guibas & Stolfi 1985). The number of additional vertices added to split segments to form a conforming Delaunay triangulation is small in practice[1].

Edge Costs. Because the triangulation is planar, the number of edges E in the triangulation is $\mathcal{O}(n)$. The average number of edges in the diametral circle of an edge is normally constant, and we can ensure that each edge cost is determined in constant time by resorting to our heuristic in which only the two triangles adjacent to each edge are considered, at the risk of finding a slightly worse route. For all practical purposes, the total time to compute edge costs is $\mathcal{O}(n)$.

Dijkstra. Dijkstra's algorithm, implemented with a Fibonacci heap (Fredman & Tarjan 1985; Cormen *et al.* 1990) as its priority queue, runs in $\mathcal{O}(n \log n + E)$ time on a graph with n vertices and E edges. Hence, finding the shortest path in the triangulation takes $\mathcal{O}(n \log n)$ time.

The total run time of the similarity metricis $\mathcal{O}(n \log n)$. The speed of the algorithm is further improved by that fact that the triangulation can be updated quickly whenever a new case is learned, without having to retriangulate the entire graph (see Section 5); this feature is one of the benefits of using Delaunay triangulations.

For a problem with multiple goals, there is only one minor change to the above algorithm. We build the triangulation and assign edge costs only once. We run Dijkstra's algorithm several times, using one of the goal vertices as the source each time. The complete set of cases is then given to PRODIGY/ANALOGY, which solves the problem. This change does not affect the asymptotic running time of the similarity metric since the number of goals is constant and small compared to the search space. PRODIGY/ANALOGY will find the best final ordering of the goals.

4 Case Adaptation and Replay

We follow the analogical replay strategy developed by Veloso (1994) in PRODIGY/ANALOGY. The replay technique involves a closely coupled interaction between planning using the domain theory (operators and other static knowledge of the world) and modification of a set of similar planning cases. Since PRODIGY/ANALOGY includes a generative planner, it is able to solve problems without requiring cases, but guidance from the cases can considerably reduce the search required and can bias the search to preferred routes previously visited in similar situations.

The stored planning cases are derivational traces of both successful and failed decisions made during past planning episodes, as well as the justifications for each decision. PRODIGY/ANALOGY uses the cases only as *guidance* in its decision-making process, and is therefore able to compensate for changes in the specific new situation or in the world model which may make parts of a case inapplicable. The case replay mechanism involves a reinterpretation of the justifications for case decisions within the context of the new problem, reusing and adapting past decisions when the justifications hold true, avoiding failed decisions, skipping unnecessary plan steps, and replanning using the domain theory when the transfer fails. The general replay algorithm is domain-independent, can replay multiple cases, and is capable of merging cases in several different manners, including interleaved, guided and random (Veloso 1992). For route planning, we introduce a *sequential* merging mode that exploits the information returned by the geometric similarity metric. The replay procedure provides guidance to the general choice points of the planner.

In the route planning domain, the retrieval procedure returns a list of cases ordered in the sequence in which the geometric similarity metric believes they should be reused for each goal. The geometric similarity metric also identifies which parts of each case should be used in the generation of the new solution, and therefore only the predicted relevant parts of each case are handed to the replay procedure.

The set of cases returned by the retrieval procedure are merged by the replay mechanism to form a solution route for a single one-goal problem[2]. In this sense, the use of multiple cases in this domain differs from the use of multiple cases in other planning domains where different cases cover different top-level goals (Veloso 1994).

To account for the geometric characteristics of the route planning domain, we developed a sequential merging strategy to combine the cases in PRODIGY/ANALOGY. Similar cases to be merged are returned by

- Input: The map description; the initial location; the goal location; and an ordered list of (chopped) cases $C_1, C_2, \ldots, C_n$ (each C_c consists of a sequence of steps geometrically relevant in the new problem).
- Output: P, a route from the initial to the goal locations.

procedure *sequential-analogical-replay*

1. $c \leftarrow 1$; $s \leftarrow 1$ (*c is the guiding case; s is the guiding step in the case*)
2. All possible case steps are marked usable.
3. Terminate if the goal location is *reached*, return the plan P.
4. Get the s^{th} plan step C_c^s from the case C_c.
5. If the case choice C_c^s is not viable, *i.e.*, *invalid* or *not reached yet*,
6.   then: If the case choice C_c^s is *invalid*,
7.   then: Mark unusable the case steps strictly dependent on C_c^s.
8. $s \leftarrow$ next usable step in C_c
9.   Go to step 4.
10.   else: *step* = Plan by generative search from domain model.
11.   If *step* matches some usable case step $C_{c'}^{s'}$,
12.   then: If $c' = c$
13.   then: Mark unusable the case steps C_c^i, $s \leq i < s'$.
14.   else: Mark unusable all steps in cases C_j, $c \leq j < c'$.
15.   Mark unusable the case steps $C_{c'}^k$, $1 \leq k < s'$.
16. $c \leftarrow c'$; $s \leftarrow s'$.
17.   Go to step 4.
18.   else: Add *step* to P.
19.   Go to step 3.
20.   else: Add the new step C_c^s to plan P.
21.   Advance case C_c to its next step: $s \leftarrow s + 1$.
22.   If the end of case C_c is *reached*,
23.   then: $c \leftarrow c + 1; s \leftarrow 1$.
24.   Go to step 3.

Table 1: Sketch of Serial Analogical Replay of a Sequence of Cases.

the retrieval procedure in the order in which they should be traversed. Because the cases only partially cover the new situation, the replay algorithm will generally need to do some extra planning. In particular, retrieved cases are often not continuously adjacent, and the initial and goal vertices may be disconnected. Extra planning to form a connected path is done by A* search. Informally, at the end of each case, the replay algorithm proceeds by searching carefully for the next case.

The same search process is performed if a case step becomes *invalidated*, for example, in a situation where the past case uses a segment (*e.g.* a particular road) that is not available in the current map, (*e.g.* the road is closed for construction). Essentially, PRODIGY/ANALOGY needs to plan an alternate route around the invalid segment. This alternate path is added to the case library, so that in the future the failure can be predicted and avoided at planning time.

When the failure occurs initially, PRODIGY updates the domain knowledge at execution time. It replans an alternate path and stores the updated domain knowledge for future use. Currently, it *does not* change any of the cases referring to that segment because (a) it could be computationally expensive in a large case library, and (b) the segment might become relevant again in the future.

Table 1 sketches the sequential replay algorithm, which merges an ordered sequence of cases. The replay procedure attempts to validate each step proposed by the cases. Usually case steps are viable in the current situation, but there are two situations when a choice may not be viable, as shown in line 5 of Table 1:

(a) Relevant cases for problem b-m0 (b) b-m0 solved with Analogy

Figure 13: Complex sequential merge, where extra planning is needed around the cases shown in (a) to find the solution shown in (b). (Map shown is 10% of the complete Pittsburgh map.)

1. when a case step is not *valid*, *i.e.*, when a choice made in the case is not applicable in the current situation, *e.g.* closed for construction, and
2. when a step is *not reached yet*, *i.e.*, the next step is not yet in the set of adjacent reachable locations from the current step, *e.g.* when there is a gap between cases.

In both of these situations the algorithm will diverge from the proposed case steps. If the choice is not valid, then the steps directly dependent on the invalid step are skipped in the case. The next usable step is identified and recursively considered by the replay algorithm (lines 7–9). Note that skipping case steps creates a gap (as in situation 1 above) that will be compensated for by generative planning.

When the algorithm diverges from the cases because the choice is not reached yet (*i.e.*, a gap is found), then generative planning needs to be done (line 10). Generative planning is done one step at a time. The algorithm is biased to return to the cases as soon as possible, by trying to match the new step with some step of the guiding cases (line 11). The newly generated step may or may not match some future case step. When the step does match a future case step (line 11), the case guidance is updated skipping the unusable parts and resetting the proposed case step to the new step (lines 12–16). The replay proceeds again recursively (line 17). If the new step does not match any case step, then the new plan step is added to the plan and planning continues one step at a time (lines 18–19).

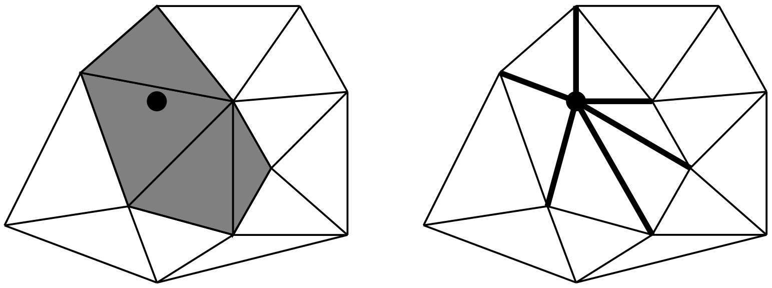
The performance of the retrieval algorithm using the geometric information is very good and also improves incrementally with execution experience. Therefore, the most frequent situation at replay time is that the case steps are valid and reusable. In this situation, the replay algorithm keeps reusing and following the cases until the goal is reached (lines 20–24).

In summary, the bias to try to link any newly generated steps to the guiding cases in the sequential merge combined with the β-biased similarity metric greatly facilitates the effective reuse of previously visited good quality routes. Figure 13 illustrates the effectiveness of this merging strategy. The cases in Figure 13a need extra planning before, after and between cases to find the solution shown in Figure 13b.

5 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, namely the conforming Delaunay triangulation, to reflect any newly learned cases. The user evaluates the route's quality, and the efficiency β is changed to reflect the evaluation. Since β captures situational information, and may vary with the identity of the user, the metric will become biased towards routes that the user prefers in specific conditions.

Vertices can be added incrementally to a Delaunay triangulation (Lawson 1977; Guibas & Stolfi 1985), so it is possible for the case library to grow without any need to retriangulate the entire graph. Inserting a new vertex is a two-step process: (i) delete any triangles whose circumcircles contain the new vertex (Figure 14a); (ii) add edges to connect the new vertex to its surrounding vertices (Figure 14b). This insertion procedure ensures that the updated triangulation will still be Delaunay. Each new case segment may be inserted as described in Section 3.3.1: first the endpoints of the segments are inserted, and if the segment does not appear as an edge in the Delaunay triangulation, the segment is recursively bisected (by inserting additional vertices) until the segment is represented as a linear sequence of edges.



(a) (b)

Figure 14: Inserting a new vertex into a Delaunay triangulation. (a) The shaded triangles are deleted because the new vertex is inside their circumcircles. (b) The new vertex is connected to its neighbors.

Virtually all Delaunay triangulations that arise in practice have the property that a vertex can be inserted in amortized constant time, if the identity of the triangle that contains the new vertex is known; see Guibas, Knuth, and Sharir (1992) for a theoretical discussion. Unfortunately, this fact does not yield a linear-time Delaunay triangulation algorithm, because finding the triangle in which an arbitrary vertex lies typically takes $\mathcal{O}(\log n)$ time. However, the vertices we insert when we navigate a new route are not arbitrary; when we are inserting a new route into the case graph, we can trace the route from start to finish through the triangulation, inserting new vertices along the way as required. Furthermore, we only need to update the costs of edges that are near the route; hence, the expected time to add a new route to the case graph is linear in the number of new vertices inserted plus the number of triangles traversed (in the original triangulation). This incremental behaviour is a major benefit of using a Delaunay triangulation.

After a new route is inserted, initial β value functions are assigned to each new case segment based on the user's evaluation of the quality of the case after execution.

If a new route involves turns or is otherwise sinuous (ignoring small bumps and features), the case is indexed by a set of straight-line segments that approximate the curve. A new route must also be split into smaller segments wherever it intersects or follows existing case segments. If each of the segments that collectively form the route is new, then each segment will have the same β value. Often, however, the new route will traverse streets in common with previous cases, and the corresponding case segments may have their β values adjusted to take into account the results of the current expedition. In this fashion, the system learns from its execution experience, and over time will select cases that are most suitable for given conditions.

We are currently investigating the issue of how much redundant information to add to the case library: the cost of retrieving and merging several non-overlapping cases might be more than that of storing multiple cases with overlapping information. The tradeoff between retrieval and modification costs is discussed briefly elsewhere (Harandi & Bhansali 1989; Veloso 1991), but is still considered a difficult problem.

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). Note that this process is different from annotating failed planning decisions within a successful case, a feature of the existing replay algorithm.

Similarly, there might be occasions to "forget" cases, such as when a case is rarely used, or when an important piece of domain knowledge changes, 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 an invalid case will become relevant again (*e.g.* whether a bridge closed for construction will be reopened).

6 Experiments

The experiments described in this section illustrate the use of our similarity metric combined with the sequential analogical replay of multiple cases. These experiments demonstrate the usefulness of the similarity metric and replay mechanism. We do not address the issues of real execution and learning, instead we use a simulated execution model.

It has been claimed that complex adaptation strategies should be avoided because it is difficult to guarantee a good final solution (Liu *et al.* 1994). Since our adaptation strategy is considerably more complex than in other existing case-based route planning systems, our goal in these experiments was to show that our method produces good routes for new problems. The experiments we describe have been performed on a map of Pittsburgh with approximately 18,000 intersections.

We randomly generated a set of 30 problems, and solved them using three different methods:

- A* to find the physically shortest path, using the Euclidean distance between two vertices as a lower bound on the travel distance.
  - For all intersections x for which a plan exists from the initial state i,
    - Examine all intersections y such that $\overline{xy}$ is a possible continuation of the route.
    - Select y such that the total length of the path is minimized (path length from i to x, plus x to y, plus the straight-line distance y to g).
- A best-first hill-climbing heuristic based on a local desire to head in the direction of the goal, corresponding to what someone might do while driving in unfamiliar territory. It was designed for use in an interleaved planning and execution system. The heuristic works as follows.
  - Assume that the plan has been expanded to intersection x.
  - Examine all intersections y such that $\overline{xy}$ is a possible continuation of the route.
  - Select y such that y is the closest intersection (straight-line distance) to the goal state g.
- PRODIGY/ANALOGY using as a case library the combined set of solutions from A* and the best-direction-first heuristic from the other 29 problems.

In practice, we prefer to accumulate cases from everyday planning and execution experience, biasing the system towards routes that the user prefers: something hard to capture in a heuristic function. In these experiments, the routes were evaluated by an execution model described in Appendix A.

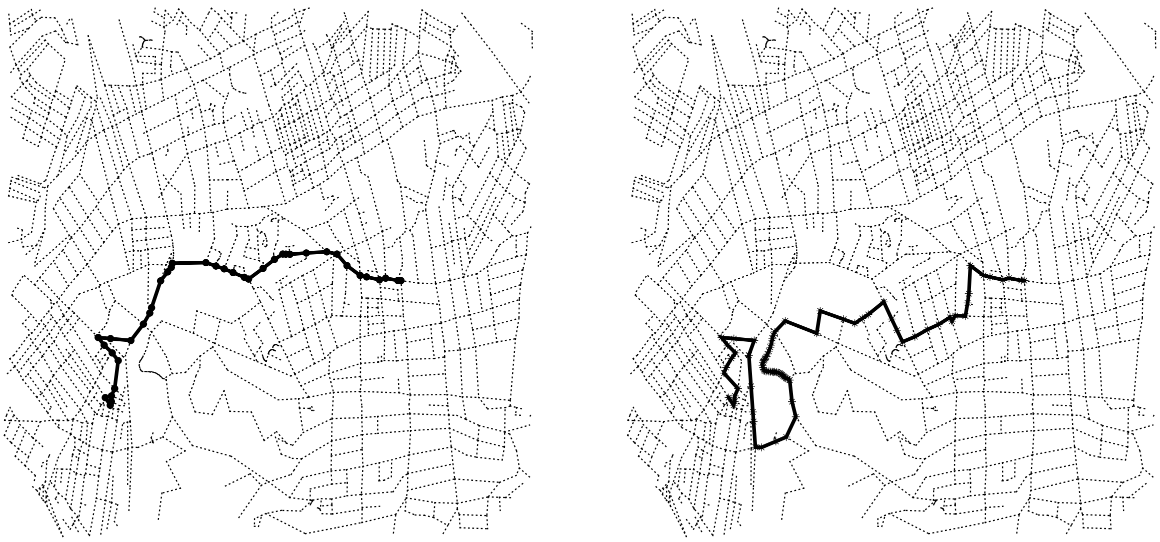
Table 2 shows the results of the average planning time, nodes expanded, and total Euclidean route length. It is clear that finding an optimal route is extremely expensive when compared to either the local direction-based heuristic or to analogy, even if a perfect, adaptable length function could be defined. Note that analogy expands more nodes than the heuristic since it is doing A* search to connect cases; while the planning time is faster because unguided PRODIGY spends more time per node deciding which node to expand next. The difference observed in route length is small enough that we can infer that the complex merge strategy generates good routes for these problems. Also, Euclidean distance is not a good evaluation function. As a result, it seems hard to justify the order of magnitude difference in planning time.

Heuristically-guided path planning also does very little search as it does no backtracking. Although reasonable in some situations, it can get sidetracked in non-grid-like cities. Several of the examples were distinctly unreasonable, more than doubling the length of the optimal route. One such example is shown in Figure 15.

Figure 16 shows an example of one problem as it was solved by A*, PRODIGY/ANALOGY and the heuristic. The A* route is the shortest route between the initial and goal vertices; however, it goes through a tightly

	Time (sec)				Nodes				Euclidean Route Length			
	μ	σ	max	min	μ	σ	max	min	μ	σ	max	min
A*	4,881	1270	34,530	86	1,068	143	3,080	84	125,175	10,416	239,486	23,549
Heuristic	686	123	3,207	94	167	30	886	24	152,666	15,914	390,679	23,549
Analogy	247	20	559	30	645	44	1271	132	154,334	7,425	283,072	29,878

Table 2: Average time, number of nodes expanded and (Euclidean) length of solution for the three search algorithms.



(a) A* (b) Heuristic

Figure 15: This figure shows how the best-direction-first heuristic can be misled. At left, a solution produced by A*. At right, a solution produced by the best-direction-first heuristic, which always selects the next intersection locally.

congested residential area, rather than taking the larger street to the south. (For those readers familiar with Pittsburgh, the residential area is Shadyside, and Fifth Avenue is the larger street.) In terms of convenience, the optimal route is in fact *worse* than either the heuristic- or analogy-generated routes. This situation occurs very often in real world cities because drivers select routes based on their convenience, familiarity and reliability rather than minimal length.

The similarity metric selected three relevant cases in the case library (Figure 16c). The metric examined all available routes (those generated by both A* and the heuristic) and selected those which described the new problem most effectively. Note that the case in the lower left corner eliminates the meandering of the heuristic path, while the case in the upper right (near the initial vertex) avoids the residential areas. The metric then returned the relevant parts of each case to the reuse algorithm, which faithfully followed the cases, adding the additional steps at the beginning and end of the cases and successfully switching between the cases along the route.

Figure 17 illustrates an example wherein the different β values assigned to case segments under different circumstances can produce different routes for the same problem. During rush hour, when larger streets are more congested, the similarity metric selects the lower route through a park and residential areas (left column). At other times, it selects the route along larger roads (right column).

Based on these observations, we infer that an algorithm more reactive than A*, such as the one we developed, is helpful for route planning. Since it is hard to develop a static evaluation function that captures the users' preferences, a system which can automatically learn and adapt its evaluation function will be more powerful. We believe that storing routes executed and evaluated by the driver is a good way to bias

(a) A* (b) Heuristic

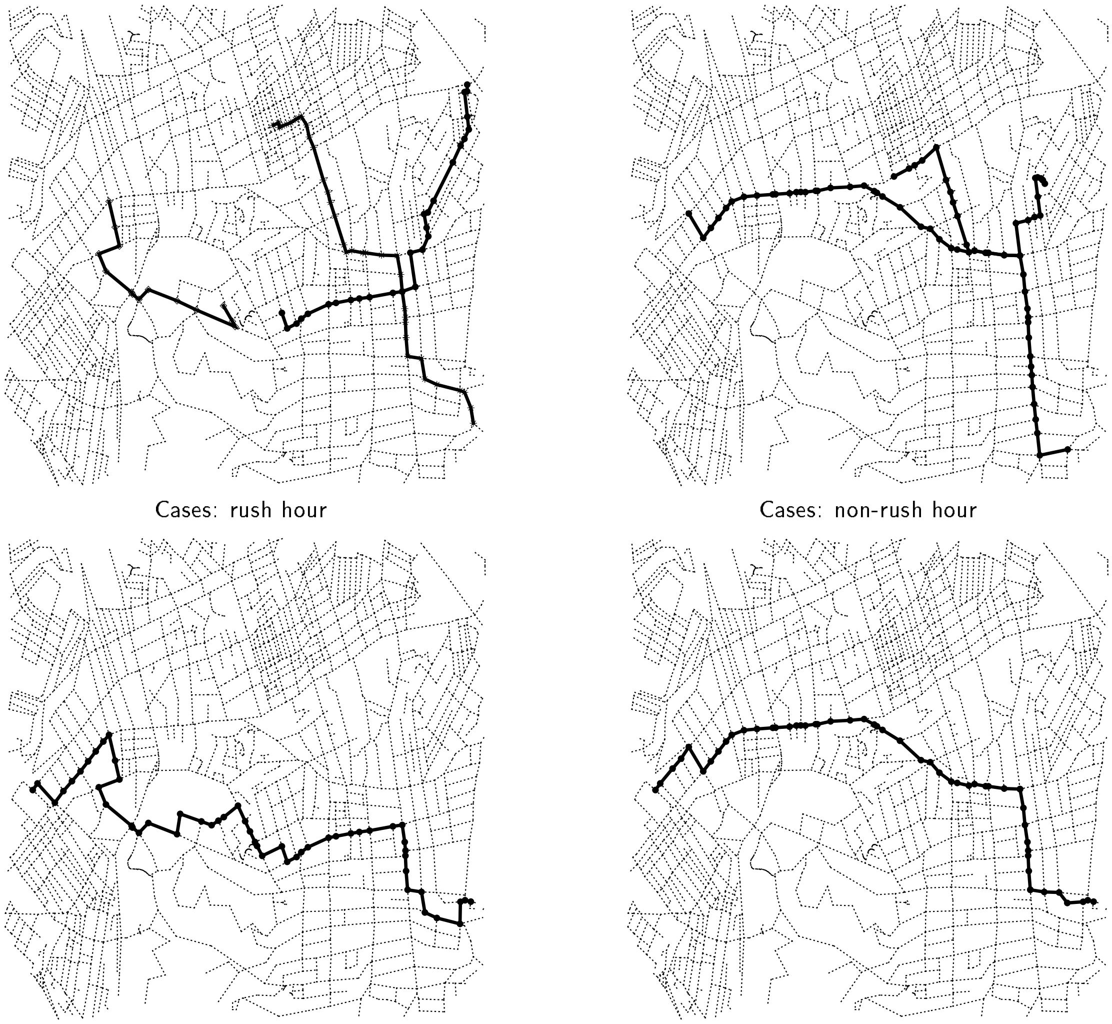
(c) Selected Cases (d) Analogy

Figure 16: (a) The path generated by A*. (b) The path generated by a search heuristic. (c) The cases selected. (d) The path generated by analogy.

the system towards preferred routes. The loss of physical optimality will be regained as the system increases its knowledge of the convenience and reliability of the route through experience. Solutions generated by analogy will therefore be better than solutions generated from scratch.

Through our experiments, which we briefly illustrated above, we have demonstrated that our method of retrieving and reusing multiple cases effectively produces appropriate solutions; they correspond to familiar routes and are not overly sinuous or long. A* and the heuristic algorithm are unable to discriminate between different situations.

With the learning methods described previously, the map information and the knowledge about the quality of each case improves with experience. A* and the heuristic function are at best unreliable when dealing with a fuzzy notion of quality that differs between users and locations. Using analogy from good-quality routes creates a system that will adapt itself to changes in its environment, to particular situations, and to particular users.



Cases: rush hour

Cases: non-rush hour

Route: rush hour

Route: non-rush hour

Figure 17: The relevant cases and the final route generated by PRODIGY/ANALOGY under different conditions.

7 Related Work

A great deal of cognitive research has been conducted on analogical reasoning. Analogy is generally assumed to be a basic process in learning and cognitive development, and to be critical in problem solving. However it is not guaranteed to preserve optimality across solutions. The contributions of cognitive science researchers to analogy are manifold. Different theories of analogical mapping and retrieval of analogs have been proposed that are supported by experimental data of human analogical reasoning and by computational models, as early as in the 1960s (Newell *et al.* 1963). Models have mainly been used to explain human performance and to compare it with a system's performance (*e.g.* Falkenhainer *et al.* 1989, Holyoak & Thagard 1994, Kean 1990). Empirical investigations have proved that analogical reasoning is central to human thinking.

The human process of analogical reasoning has been transferred to machine problem solving methods in the field of case-based reasoning (*e.g.* Schank 1982, Kolodner 1993, Carbonell 1983). General artificial intelligence analogical reasoning systems are not necessarily psychologically plausible. In particular our work also does not aim at capturing the human analogical reasoning process, though we ground it on the evidence that humans rely on case-based experience.

The remainder of our related work discussion focusses on route planning algorithms with emphasis on the few systems that incorporate analogical reasoning.

Most robotics route planners (*e.g.* Dyna (Sutton 1991), COLUMBUS (Thrun 1993), Xavier (Goodwin & Simmons 1992), NavLab (Thorpe 1990)) don't remember paths or path quality, and typically use shortest path, dynamic programming or decision-theoretic algorithms to determine routes. We believe these algorithms need to be extended in the route planning domain because

- They typically examine the entire map and do a blind search that ignores the general direction of the goal. Hence, for a large map, they are unnecessarily slow;
- They do not handle incomplete information;
- They do not interleave planning with execution, or learn from unexpected situations;
- They do not consider situation-dependent information, including path reliability or convenience factors such as time-to-goal, road quality, or user preferences; and
- It is extremely difficult to develop an evaluation function that captures and adapts to a particular users' preferences.

Heuristic search techniques in general are prone to failure in unexpected conditions. Both heuristic and optimal search techniques are limited to the particular implementation or definition of "optimality," and do not adapt to changing conditions. The incremental learning system described in this article allows the system to adapt itself to a particular user and to react to a dynamic environment.

Abstraction planning, such as (Branting & Aha 1995; Holte *et al.* 1994), is another method used in the route planning domain. It improves planning time dramatically, but makes no guarantees about the quality of its generated routes.

ROUTER (Goel *et al.* 1994) and R-Finder (Liu *et al.* 1994) are the only other case-based route-planning systems we are aware of. Both have extremely simple retrieval and modification algorithms, providing less transfer of prior experience than our algorithm. In particular, they cannot retrieve or compose multiple episodes, nor can they identify and exploit partial cases. In addition, they do not revalidate the plans in the current map, making it more likely that in a dynamic world they will return invalid solutions. Nor do they remember the quality of cases in an attempt to improve retrieval and the quality of the plans they generate. Finally, the goal of their work is to speed up planning; we feel that fast planning can be achieved by a combination of Dijkstra's algorithm and goal-oriented heuristics. Our focus, therefore, is instead on the more difficult questions of plan quality and reliability.

8 Conclusion

Our analogical route planner successfully integrates several sophisticated novel algorithms. It includes a novel geometric similarity metric, a sequential case replay mechanism that initiates additional planning to fill gaps not covered by existing cases, and learning algorithms that incrementally update both the geometric and symbolic information stored in the case library. Collectively, these components form a powerful planner capable of adapting to changing conditions, different users, and even a dynamic map.

The main technical contributions of this work include

- the exploitation of the geometric properties of the domain,
- the use of partial cases, multiple cases, and case ordering information in analogical route planning, and
- the capture not only of previous planning, but also of execution experience in the indexing of the case library. By coupling analogical reasoning with feedback from execution experience, our planner can incrementally improve both the efficiency of route planning and the quality of the routes generated.

The domain of route planning has proven to be a good testbed for planning by analogy because of the necessity of dealing with incomplete and changing real-world domain information, because of the large scale of the task and its accompanying computational demands, and most of all because of its geometric properties. These features have collectively led us to several interesting extensions of the analogical reasoning method. As we have noted, there are a variety of reasons why traditional methods of route planning are not equipped to handle real-world navigation problems; the most debilitating is their inability to adapt to incomplete information and changing conditions. Hence, we believe that learning must be a part of a robust practical route planner.

Accordingly, our similarity metric can learn and exploit both geometric and symbolic situation-dependent features of a map and the routes it represents. Because the similarity metric exploits geometric features of the domain, it can find partial and multiple cases and suggest an appropriate case ordering, and can do so with an efficiency that purely symbolic planners cannot easily match. Furthermore, its continuous-valued character allows it to avoid common problems that occur when values are forced into discrete ranges to aid case retrieval. Because the similarity metric exploits symbolic features, it can adapt to user preferences, changes in road conditions, and changes in the map itself. Because execution experience is captured in the case efficiency values (for the similarity metric) and the case histories (for the sequential replay mechanism), the planner learns to select cases and routes that are good in practice. As a final bonus, the incremental nature of our triangulation-based case indexing method makes it inexpensive to update both the geometric and symbolic information.

Our sequential case replay mechanism takes advantage of the features provided by our similarity metric. The replay mechanism is given a set of cases with a suggested order, as well as suggested entry and exit points for partial cases. Using this information as a guide, the replay mechanism sequentially reviews the cases and weaves them together into a complete route, performing extra planning to fill gaps between the retrieved cases and the gaps that appear within cases when previously successful steps are invalidated by changing map information. The latter is possible because the system updates the domain knowledge based on execution feedback, and thus the replay mechanism can respond to unexpected impediments. By taking advantage of the cases recovered by the similarity metric, the planner can create a final plan with considerably less search than if it were planning from scratch.

Our experimental data supports the effectiveness of the algorithms we have described herein. By using past experience with good-quality cases as a starting point, our planner achieves not only efficiency, but also the ability to adapt to nasty surprises, to multiple users, and to changing conditions of the real world.

Notes

1. Pathological cases exist where up to $\mathcal{O}(n^3)$ vertices must be added, but these are of theoretical interest only and never occur in realistic maps; Edelsbrunner and Tan (1993) discuss how to construct conforming Delaunay triangulations within this bound.

2. A single problem can consist of multiple goals, but the similarity metric is only invoked for a single source/goal pair. PRODIGY/ANALOGY is capable of handling multiple goals in the map domain. Multiple goals can be decomposed according to their geometric location into a sequence of one goal problems.

Code Availability

The majority of the code for this system and instructions on how to use it are available from the Web page `http://www.cs.cmu.edu/~khaigh/map.html`.

Acknowledgments

This research is sponsored in part by (1) the Natural Sciences and Engineering Research Council of Canada, (2) the National Science Foundation under grant CMS-9318163, and (3) the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant number F33615-93-1-1330.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of NSERC, the NSF, Wright Laboratory or the U. S. Government.

A Execution Model

The cost of a route was calculated by the execution model using the following formula:

$$route\_cost = \frac{\sum_{segments \in route} length(segment) * cost(segment)}{length(route)}$$

where $cost(segment)$ is determined by selecting a value from a Gaussian distribution with a mean and deviation determined from:

- whether the segment is known to be bad,
- the segment's location,
- weekday,
- time of day,
- turning angle (*i.e.* routes with few turns are better than those with many).

Take for example a particular segment in a residential area. If the route makes a left turn onto the segment on the weekend, the Gaussian mean is $(0.9 + 90.0/45.0)$, whereas if the route goes straight onto the segment on a weekday during rush hour, the mean is $(0.3 + 1.0 + 0.0/45.0)$. The full function is shown below:

BAD_ROUTES = known poor quality routes

1. if (*segment* ∈ BAD_ROUTES) or (*segment*-direction == WRONG_WAY_ONE_WAY)
   then *mean* = 3.0
   else if (*today is weekend*)
   then case *segment*-type == RESIDENTIAL: *mean* = 0.9
   segment-type == BUSINESS: *mean* = 0.7
   segment-type == BOULEVARD: *mean* = 0.5
   segment-type == HIGHWAY: *mean* = 0.3
   else case *segment*-type == RESIDENTIAL: *mean* = 0.3
   segment-type == BUSINESS: *mean* = 0.5
   segment-type == BOULEVARD: *mean* = 0.7
   segment-type == HIGHWAY: *mean* = 0.9
   case *time* == NON_RUSH_HOUR: *mean* = *mean* - 0.1
   time == RUSH_HOUR: *mean* = *mean* + 2.0
2. *mean* = *mean* + (turning-angle / 45.0)
3. if ($mean \geq 1.0$)
   then *deviation* = 0.1
   else *deviation* = 0.3
4. *segment*-beta = Gaussian(*mean*, *deviation*)

B Proofs

Theorems B.1 through B.3 characterize optimum routes and prove the correctness of our optimal algorithm.

Theorem B.1 *Any optimal route is a sequence of line segments whose endpoints all fall on case graph vertices or segments.*

Proof. Suppose R is a route which has a section R' that does not intersect the case graph (except possibly at the endpoints of R'), and R' is not a straight line segment. Let a and b be the endpoints of R'. We construct a new route $Q = R - R' + \overline{ab}$. Q is strictly shorter than R, so R is not optimal. ■

In our exact algorithm, it is not necessary that every edge of the complete graph G be labeled with the cost of the shortest route between its endpoints (though it is necessary for the shortest edges). If a shortest route between two vertices x and z contains another vertex y, and the costs of (x, y) and (y, z) are calculated accurately, then Dijkstra's algorithm will find this (or another) optimal route even if the cost associated with (x, z) is greatly overestimated. Therefore we only need to assign the correct cost to an edge (x, z) if all minimal routes between x and z contain no other vertices. Our exact algorithm takes advantage of this fact, aided by the following theorem.

We say that a segment S is *travelled* by a route R if R intersects a nonzero length of S. (Hence, R does not travel S if R intersects S in only a finite number of points.)

Theorem B.2 *Let R be a minimal route from x to z that travels two or more case segments, and that intersects no other vertex than x and z. Then either there is another minimal route R' from x to z that travels at most one case segment, or there is another minimal route R' from x to z that passes through some intermediate vertex y.*

Proof. Refer to the examples in Figure 18a and 18c. Let $S_1 = \overline{a_1 b_1}$ and $S_2 = \overline{a_2 b_2}$ be the first two case segments travelled by R. Let c_1 be the point where R leaves S_1, and c_2 the point where R enters S_2. Note that c_1 and c_2 are not endpoints of S_1 or S_2 (because we specified that R intersects no vertex except x and z). Let R_c be the portion of R between c_1 and c_2. R_c does not intersect any vertex or travel any other case segment, so by Theorem B.1, $R_c = \overline{c_1 c_2}$.

Let R_c^+ and R_c^- be segments that run parallel to R_c on either side of R_c, separated from R_c by a distance ϵ. Choose $\epsilon > 0$ to be sufficiently small that R_c^+ and R_c^- intersect the segments S_1 and S_2; the intersection points are labeled c_1^+, c_2^+, c_1^-, and c_2^-, and serve as the endpoints of the parallel segments, so that $R_c^+ = \overline{c_1^+ c_2^+}$ and $R_c^- = \overline{c_1^- c_2^-}$. Let R^+ and R^- be routes that are similar to R, but traverse R_c^+ and R_c^- respectively instead of R_c.

Denote the cost of route R by $C(R)$. From an examination of the segments that make up each of the three routes R^-, R, and R^+, it is apparent that $C(R^+) - C(R) = C(R) - C(R^-)$ (regardless of what efficiency coefficients are assigned to S_1 and S_2). In other words, if the route R^+ costs more than R, then R^- must cost less; and vice-versa. However, R is a minimal route, so neither R^+ nor R^- may cost less than R; it follows that all three routes have the same cost. Furthermore, there is an infinite set of minimal routes, chosen by varying ϵ within some valid range.

Given these definitions, we will show how to construct two different minimal routes from x to z; each of these routes either travels one less segment than R, or intersects some intermediate vertex. Arbitrarily choose either R_c^+ or R_c^-; each will yield a different minimal route. For illustrative purposes, suppose we have chosen R_c^-. Increase ϵ until one of the following conditions occurs.

- Point c_1^- coincides with an endpoint a_1 or b_1 of case segment S_1,
- Point c_2^- coincides with an endpoint a_2 or b_2 of case segment S_2 (illustrated in Figure 18b),
- Point c_1^- coincides with the entry point of R onto S_1 (illustrated in Figure 18d), or
- Point c_2^- coincides with the exit point of R from S_2.

In the first two cases, call the coincidental endpoint y. If $y = x$ or $y = z$, then R^- is a minimal route that travels one less segment than R; see the next paragraph. Otherwise, R^- is a minimal route R' that travels through an intermediate vertex y, which is one of the segment endpoints. (Note that we can decompose R'

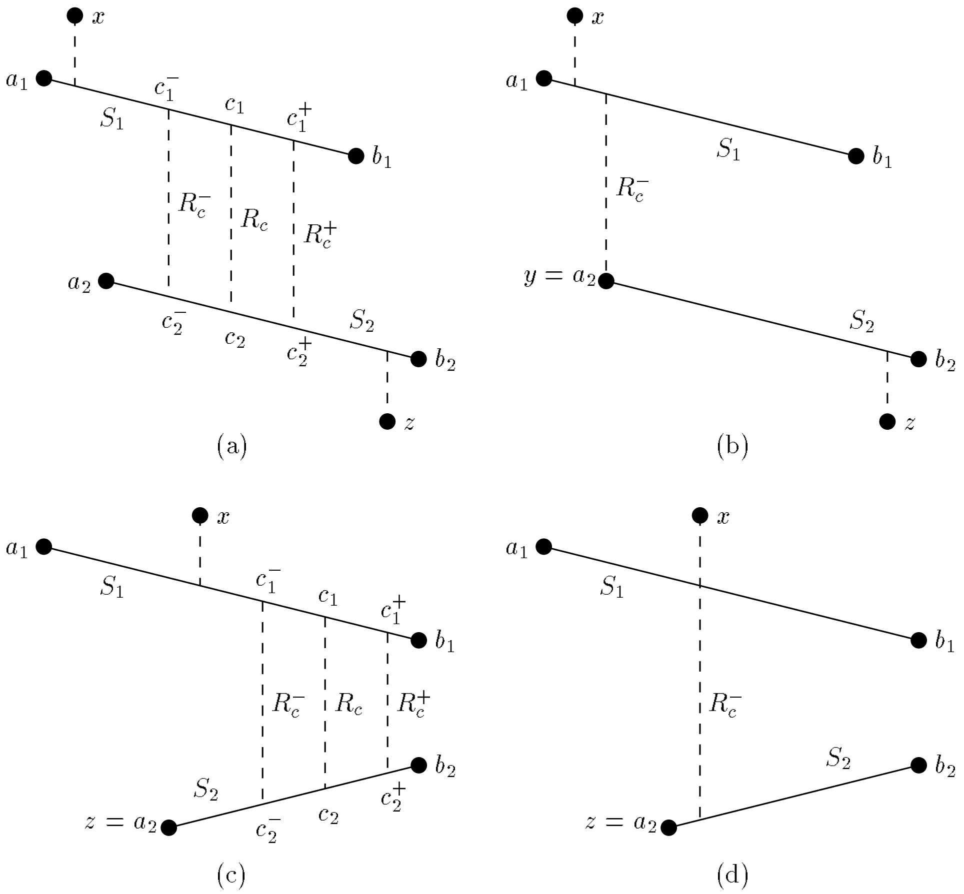


Figure 18: (a) If R is a minimal route from x to z (passing through R_c), then R^+ and R^- (passing through R_c^+ and R_c^-) must also be minimal routes from x to z. (b) R^- is a minimal route that passes through an intermediate vertex y. (c) Another example where R, R^+, and R^- are all minimal routes. (d) R^- is a minimal route that travels one less segment than R.

into two minimal subroutes; one from x to y traveling only S_1, and one from y to z traveling S_2 and possibly other case segments. If the latter subroute travels more than one case segment, we can recursively apply the procedure described herein.)

In the last two cases, R^- is a minimal route that travels one less segment than R; if R travels more than two segments, the procedure must be applied repeatedly, removing one case segment at a time until one obtains a minimal route R' that either travels only one case segment or passes through an intermediate vertex. In any case, any minimal route can be recursively converted into a minimal route that travels at most one case segment between any two consecutive vertices. ■

Note that we never actually apply the procedure described in the proof above to find a route. Rather, we are interested in the fact that there exists an optimal route that travels at most one segment between any two consecutive vertices, because it means that we can find that optimal route without working very hard to assign edge costs. We formalize this as follows.

Theorem B.3 *Assign a weight to each edge (x, y) in G as described in Section 3.2. If an optimal route from an initial vertex i to a goal vertex g in the case graph has cost C, then there is a corresponding shortest path from i to g in G that has cost C and traverses the same vertices (and possibly others).*

Proof. Suppose R is a minimal route which traverses the sequence of vertices $R^u = \langle u_1, u_2, \ldots, u_m \rangle$, where $i = u_1$ and $g = u_m$. Let R_j be the subroute of R with endpoints $\langle u_j, u_{j+1} \rangle$. Each subroute R_j is itself a

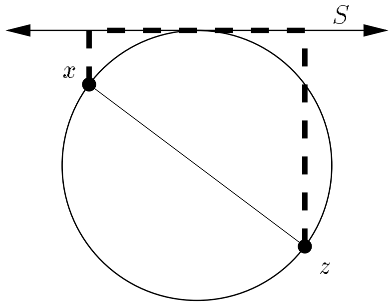


Figure 19: The minimal route from x to z that travels S is illustrated as bold dashed lines. If travel along S costs nothing, then the cost of this route is equal to the diameter of the circle, so $\overline{xz}$ is an equally good route. If travel along S incurs any cost, or if S is moved further away, then $\overline{xz}$ is a better route.

minimal route, so by Theorem B.2, there is a route R'_j from u_j to u_{j+1} that has the same cost as R_j and, although R'_j may pass through additional vertices, R'_j travels at most one case segment between any two vertices.

Let R' be the union of the subroutes R'_j; clearly, R' is a minimal route with the same cost as R. Suppose R' traverses the sequence of vertices $R^v = \langle v_1, v_2, \ldots, v_n \rangle$, where $i = v_1$, $g = v_n$, and R_v includes all the vertices in R^u as well as all the intermediate vertices in the subroutes R'_j. Let R''_j be the subroute of R' with endpoints $\langle v_j, v_{j+1} \rangle$. Because R''_j travels at most one case segment, R''_j is one of the routes considered when a weight is assigned to the edge (v_j, v_{j+1}) (by the procedure described in Section 3.2), so $C(v_j, v_{j+1}) = C(R''_j)$.

It follows that the cost of the path R^v in G is $C(R)$. Edge weights in G are never underestimated, so R^v is a shortest path in G. ■

Theorem B.3 implies that finding a shortest path in G will produce a correct solution to the shortest route problem.

We claim in Section 3.3.2 that when we assign a cost to an edge e, we need only examine case segments that intersect the interior of the diametral circle of e. We prove the fact now.

Theorem B.4 *Let S be a segment that does not intersect the interior of the diametral circle of (x, z). The length of $\overline{xz}$ is less than or equal to the cost of any route from x to z that travels only S.*

Proof. Suppose that an optimal route from x to z travels S. (By Theorem B.3, we need not consider routes that travel more than one segment.) Without loss of generality, assume that S has efficiency zero (so that travel along S incurs no cost). If S extends infinitely in both directions and is tangent to the diametral circle, then the sum of the distance from x to S and the distance from S to z is exactly equal to the length of $\overline{xz}$ (see Figure 19); hence, $\overline{xz}$ is also an optimal route. Even if S is finite or is not tangent to the diametral circle, there clearly cannot be a route travelling S having lower cost than $\overline{xz}$. ■

References

Bern, M., and Eppstein, D. (1992). Mesh generation and optimal triangulation. In Du, D.-Z., and Hwang, F. (eds), *Computing in Euclidean Geometry*, volume 1 of *Lecture Notes Series on Computing*. (Singapore: World Scientific). pp. 23–90.

Branting, L. K., and Aha, D. W. (1995). Stratified case-based reasoning: Reusing hierarchical problem solving episodes. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95).* (San Mateo, CA: Morgan Kaufmann), pp. 384–390.

Broos, P., and Branting, K. (1993). Compositional instance-based acquisition of preference predicates. In *Case-Based Reasoning: Papers from the 1993 Workshop.* (Menlo Park, CA: AAAI Press), pp. 70–75. Available as Technical Report WS-93-01.

Bruegge, B.; Blythe, J.; Jackson, J.; and Shufelt, J. (1992). Object-oriented system modeling with OMT. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '92).* (ACM Press), pp. 359–376.

Carbonell, J. G.; Knoblock, C. A.; and Minton, S. (1990). PRODIGY: An integrated architecture for planning and learning. In VanLehn, K. (ed.), *Architectures for Intelligence.* (Hillsdale, NJ: Erlbaum). Also Available as Technical Report CMU-CS-89-189.

Carbonell, J. G. (1983). Learning by analogy: Formulating and generalizing plans from past experience. In Michalski, R. S.; Carbonell, J. G.; and Mitchell, T. M. (eds), *Machine Learning, An Artificial Intelligence Approach.* (Palo Alto, California: Tioga Press), pp. 137–162.

Cormen, T. H.; Leiserson, C. E.; and Rivest, R. L. (1990). *Introduction to Algorithms.* (Cambridge, MA: MIT Press).

Delaunay, B. N. (1934). Sur la sphere vide. *Bull. Acad. Science USSR VII: Class. Sci. Math.*, pp. 793–800.

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

Edelsbrunner, H., and Tan, T. S. (1993). An upper bound for conforming Delaunay triangulations. *Discrete & Computational Geometry*, **10**:197–213.

Eppstein, D. (1996). Spanning trees and spanners. Technical Report 96-16, Department of Information and Computer Science, University of California at Irvine.

Falkenhainer, B.; Forbus, K. D.; and Gentner, D. (1989). The Structure-Mapping Engine: Algorithms and examples. *Artificial Intelligence*, **41**:1–63.

Fortune, S. (1992). Voronoï diagrams and Delaunay triangulations. In Du, D.-Z., and Hwang, F. (eds), *Computing in Euclidean Geometry*, volume 1 of *Lecture Notes Series on Computing*. (Singapore: World Scientific). pp. 193–233.

Fredman, M. L., and Tarjan, R. E. (1985). Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, **34**(1):209–221.

Goel, A.; Ali, K.; Donnellan, M.; de Silva Garza, A. G.; and Callantine, T. (1994). Multistrategy adaptive path planning. *IEEE Expert*, **6**(6):57–65.

Goodwin, R., and Simmons, R. (1992). Rational handling of multiple goals for mobile robots. In Hendler, J. (ed.), *Artificial Intelligence Planning Systems: Proceedings of the First International Conference (AIPS-92).* (San Mateo, CA: Morgan Kaufmann), pp. 70–77.

Guibas, L., and Stolfi, J. (1985). Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, **4**(2):74–123.

Guibas, L. J.; Knuth, D. E.; and Sharir, M. (1992). Randomized Incremental Construction of Delaunay and Voronoï Diagrams. *Algorithmica*, **7**(4):381–413. Also available as Stanford University Computer Science Department Technical Report STAN-CS-90-1300.

Haigh, K. Z., and Shewchuk, J. R. (1994). Geometric similarity metrics for case-based reasoning. In *Case-Based Reasoning: Working Notes from the AAAI-94 Workshop.* (Menlo Park, CA: AAAI Press), pp. 182–187.

Haigh, K. Z., and Veloso, M. (1995). Route planning by analogy. In *Case-Based Reasoning Research and Development, Proceedings of ICCBR-95.* (Sesimbra, Portugal: Springer-Verlag), pp. 169–180.

Hammond, K. J. (1990). Case-based planning: A framework for planning from experience. *Cognitive Science*, **14**(3):385–443.

Harandi, M. T., and Bhansali, S. (1989). Program derivation using analogy. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89).* pp. 389–394.

Holte, R. C.; Mkadmi, T.; Zimmer, R. M.; and MacDonald, A. J. (1994). Speeding up problem-solving by abstraction: A graph-oriented approach. Technical Report TR-95-07, Department of Computer Science, University of Ottawa.

Holyoak, K. J., and Thagard, P. (1989). Analogical mapping by constraint satisfaction. *Cognitive Science*, **13**:295–355.

Keane, M. T. (1990). Incremental analogizing: Theory and model. In Gilhooly, K. J.; Keane, M. T.; Logie, R.; and Erdos, G. (eds), *Lines of Thinking: Reflections on the Psychology of Thought, Vol I.* (Wiley).

Keil, J. M., and Gutwin, C. A. (1992). Classes of graphs which approximate the complete Euclidean graph. *Discrete & Computational Geometry*, **7**(1):13–28.

Kolodner, J. L. (1993). *Case-Based Reasoning.* (San Mateo, CA: Morgan-Kaufmann).

Lawson, C. L. (1977). Software for C^1 surface interpolation. In Rice, J. R. (ed.), *Mathematical Software III.* (New York: Academic Press). pp. 161–194.

Lee, D.-T., and Schachter, B. J. (1980). Two algorithms for constructing a Delaunay triangulation. *International Journal of Computer and Information Sciences*, **9**(3):219–242.

Liu, B.; Choo, S.-H.; Lok, S.-L.; Leong, S.-M.; Lee, S.-C.; Poon, F.-P.; and Tan, H.-H. (1994). Integrating case-based reasoning, knowledge-based approach and Dijkstra algorithm for route finding. In *Proceedings of the Tenth Conference on Artificial Intelligence for Applications.* (Los Alamitos, CA: IEEE Press), pp. 149–155.

Mitchell, J. S. B., and Papadimitriou, C. H. (1991). The weighted region problem: Finding shortest paths through a weighted planar subdivision. *Journal of the Association for Computing Machinery*, **38**(1):18–73.

Newell, A.; Shaw, J. C.; and Simon, H. A. (1963). Empirical explorations with the logic theory machine: A case study in heuristics. In Feigenbaum, E., and Feldman, J. (eds), *Computers and Thought.* (New York, NY: McGraw-Hill).

Ruppert, J. (1995). A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, **18**(3):548–585.

Schank, R. C. (1982). *Dynamic Memory.* (Cambridge, MA: Cambridge University Press).

Shewchuk, J. R. (1996). Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *First Workshop on Applied Computational Geometry.* (Association for Computing Machinery), pp. 124–133.

Sutton, R. S. (1991). Planning by incremental dynamic programming. In *Machine Learning: Proceedings of the Eighth International Workshop on Machine Learning (ML91).* (San Mateo, CA: Morgan Kaufmann), pp. 353–357.

Thorpe, C. E. (ed.). (1990). *The CMU Navlab.* (Boston, MA: Kluwer Academic Publishers).

Thrun, S. B. (1993). Exploration and model building in mobile robot domains. In *Proceedings of the IEEE International Conference on Neural Networks.* (San Francisco, CA: IEEE Press), pp. 175–180 vol. 1.

Veloso, M. M.; Carbonell, J.; Pérez, M. A.; Borrajo, D.; Fink, E.; and Blythe, J. (1995). Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, **7**(1):81–120.

Veloso, M. M. (1991). Variable-precision case retrieval in analogical problem solving. In *Proceedings of the 1991 DARPA Workshop on Case-Based Reasoning.* (San Mateo, CA: Morgan Kaufmann), pp. 93–106.

Veloso, M. M. (1992). Automatic storage, retrieval, and replay of multiple cases. In *Preprints of the AAAI 1992 Spring Symposium Series, Workshop on Computational Considerations in Supporting Incremental Modification and Reuse.* (Menlo Park, California: AAAI Press), pp. 131–136.

Veloso, M. M. (1994). *Planning and Learning by Analogical Reasoning.* (Berlin, Germany: Springer Verlag). PhD Thesis, also available as Technical Report CMU-CS-92-174, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.