Intelligent Route Finding: Combining Knowledge, Cases and An Efficient Search Algorithm

Bing Liu [1]

Abstract: One of the major road transportation problems is route finding. With the massive and complicated road network of a modern city (or country), finding a good route to travel from one place to another is not a simple task. In Network Theory, this is a shortest path problem. Shortest path algorithms are often used to solve the problem. However, these algorithms are wasteful in terms of computation when applied to the route finding task. They may also produce routes that are not suitable for human drivers. Furthermore, finding a route only solves part of the route finding problem. Accepting user's feedback and learning from past experiences are also crucial to the success of a route finding system. In this paper, we present an integrated approach of using knowledge about the road network, past cases and an efficient search algorithm for route finding. This technique not only substantially reduces the time and space required in computation, but also produces human-oriented and personalized solutions. Based on this approach, a system has been implemented for route finding in Singapore.

1 INTRODUCTION

Traveling has become an important part of every person's day to day life. With complicated roads of a modern road network, finding a good route to travel is not an easy task. Inefficiency of human route finding also results in a large amount of excessive travel. It was estimated that excess travel amounted to 83.5 billion miles and 914,00 person-year annually at a total estimated cost of more than 45 billion dollars per year in United States alone [7].

Route finding is a shortest path problem [4, 13]. Shortest path algorithms, such as Dijkstra algorithm [3] and *A** [11], are often used for solving this problem. However, these algorithms are wasteful in terms of computation when applied to the route finding task. Most of the roads searched by the algorithms are actually irrelevant as they cannot possibly be part of the solution. Using only these algorithms may also produce solutions that are not suitable for human drivers. For example, human drivers would normally like to drive on major roads, although using some minor roads may give the shortest path. In [9], we proposed a novel approach of using knowledge about the road network to help a shortest path algorithm to perform the route finding task. This approach dramatically reduces the time and space required in computation, and also produces human oriented solutions. The approach is characterized by using knowledge about the road network to isolate the search, and an efficient algorithm to search for the best solution in the isolated areas rather than the whole network. This paper further improves this technique by addressing an important case, which was not properly handled in [9].

Finding a route efficiently only solves part of the route finding problem. After a route is found, the user needs to feedback to the system of his/her driving experiences, and the system in turn should take some appropriate actions. Furthermore, according to human behavior study [1], people also prefer to travel on familiar roads. To meet these requirements, it seems that a case-based reasoning (CBR) [4] approach is required. A number of systems have used the CBR approach [4, 5, 8]. However, we will show that the CBR approach is inefficient for this problem. Then, the challenge is how to realize CBR's effect of choosing familiar roads, while avoiding its difficulties. This paper proposes a simple method that integrates well with the route finding approach above. This method does not have the usual problems associated with CBR. Based on these two techniques, a system, called KB-RFinder, has been implemented for route finding in Singapore.

2 SHORTEST PATH ALGORITHMS, KNOWLEDGE BASED METHOD AND THE CBR APPROACH

2.1. Shortest path algorithms

Shortest path problem is modeled as finding the shortest path between two nodes in a weighted network $G = (N, A)$ with a node set N and an arc set A. In route finding, G is the road network.

Dijkstra algorithm [3, 13] is perhaps one of the most efficient algorithms for the problem. Most of the other algorithms are its variations. In our research, we performed a comparison with eight best known algorithms, i.e., $D1$ (Dijkstra algorithm), $A1$ (A^*) [11], $S1$, $S2$, $D2$, $A2$ [12], $S12$, and $S22$ [6]. $D1$, $A1$, $S1$, and $S2$ all perform uni-directional search [6]. $D2$, $A2$, $S12$, and $S22$ are their bi-directional counterparts respectively [6, 12]. Due to the space limitation, we are unable to show the running time results, Interested reader see [10]. From the results, A^* is found to be the fastest algorithm. The performances of the others are similar.

Although these algorithms are relatively efficient, as discussed earlier, applying them directly to route finding is not appropriate.

2.2. Knowledge-based technique

Knowledge-based problem solving emphasizes the use of human problem solving strategies on a computer. For route finding, a number of behavior studies have resulted in some important findings about people's route finding [1]:

1) People prefer to travel on major roads.
2) People's route finding process is characterized by minimizing the angle between the present direction of travel and the calculated direction to the destination.
3) People like to travel on familiar routes.
4) People learn from driving experiences.

These findings suggest at least two types of AI-based reasoning, i.e., model-based reasoning [4] and case-based reasoning (CBR) [4], are needed. Model-based reasoning in route finding is characterized by a combination of direction finding and route finding [4]. However, we believe that model-based method is not efficient. The reason is that human beings are not efficient route finders. They are seldom able to find the best route unless the distance of travel is very short. Thus, we should not simply copy the human ways of route finding. The CBR approach will be discussed in the next sub-section.

1 Department of Information Systems and Computer Science, National University of Singapore, Lower Kent Ridge Road, Singapore, 119260.

© 1996 B. Liu
ECAI 96. *12th European Conference on Artificial Intelligence*
Edited by W. Wahlster
Published in 1996 by John Wiley & Sons, Ltd.

However, human route finding does have its advantages, i.e., human beings are good at using knowledge about the road network to isolate part of the network that may contain a solution. They are not good at searching for the best solution in the isolated area.

From the above, we can see that although neither the algorithmic approach nor the knowledge-based method provides a good solution for route finding, each of them does have its advantages. Analysis of their advantages and disadvantages shows that they can complement each other very well. Thus, integrating them is the natural solution. In this combined approach, knowledge is used to isolate the search and an algorithm is used to search for the best solution in the isolated areas. Section 3 describes this approach.

2.3. Case-based reasoning (CBR)

Case-based reasoning is an AI technique that "remembers" previous problems and solutions and uses what worked before to solve new problems. This is similar to our human experiences. For example, if one wants to go to a place where he has been before, most likely he will not use the map to find a new route. Instead, he is likely to use the previous route in his memory to guide the present trip. If he wants to go to a place near to where he has been before, he will extend the old solution for the new problem.

Although intuitively appealing, we believe using the CBR approach is not efficient for route finding. The reason is that it is difficult to store and maintain shortest routes in a huge network. The case base could grow very large, which create problems for case matching and retrieval. When there is no complete match of the new problem with an old problem, it is difficult to know what the similar cases are for the new problem, and how to adapt the old solutions and to ensure good solution quality. For example, if partial cases are joined together to form a partial or a complete solution to the new problem, it is difficult to know whether they all go in the same direction as the destination. Even if they do, they still may not guarantee a good solution as the following example shows (Figure 1). In this example, we would like to travel from 1 to 6 and there is a reservoir in between. There are two existing cases, i.e., from 1-2 and from 3-4. They seem to go in the same direction as the destination 6. If we join them together to form a partial solution and leave the rest of the route to be searched by Dijkstra algorithm, it is clear that this partial solution is a bad one because when we reach 4, we must come back to 5 and then go to 6. The best solution will be from 1-5-6.

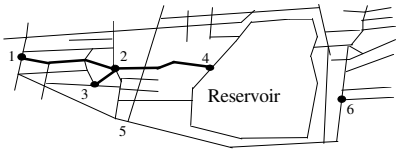

Figure 1. A problem situation with cases

This type of situation could happen very often if a big city (e.g., Singapore) has many hills, reservoirs, big parks, etc., which make many parts of the road network very irregular.

It is these problems that we have encountered in building our earlier system R-Finder [8] that have prompted us to find a better solution technique. However, old cases are useful because people do like to travel on familiar routes. Then, the system must meet this preference. In Section 4, we will present a simple and effective method that meets our needs and also avoids the problems associated with CBR.

3 USING KNOWLEDGE FOR ROUTE FINDING

In [9], we presented the main idea of using knowledge to isolate search in route finding. This section discusses the improvement made to the original approach. For completeness, we review the whole approach here. Along with it, we will point out its problem and present an improvement that overcomes the problem.

To begin with, the findings on people's route choice suggest an ideal situation in route finding. Assume that a driver plans to go from a location *S* on a minor road to another location *D* on another minor road. Ideally, the system should search in a small area around *S* to reach some major roads (to reach the nearest major road may not be sufficient, discussed in Section 6). Within this area, it considers both major and minor roads. After this, it should consider only major roads. When it is approaching *D*, the system again should search in a small area in which both major and minor roads are considered in order to reach *D*. Figure 2 illustrates this scenario. Grids are used to show the isolated areas. This approach not only prunes off a great deal of search space, but also produces routes that satisfy people's preference of traveling on major roads.

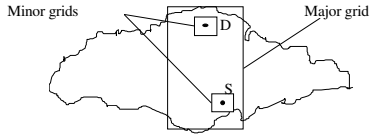

Figure 2. An ideal route finding situation

Obviously, this ideal case does not guarantee the best solution. However, the solution may be good enough as people do prefer to drive on major roads.

Now, the problem is how to form these grids. The next sub-section describes the technique that uses knowledge about the road network to form these grids.

3.1 Using road network knowledge to isolate search

3.1.1 Knowledge about the road network

Before presenting our grid formation technique, we first describe the key types of knowledge we use. There are two main types:

- **Road types:** In the typical road network of a city, there are three types (or levels) of roads, minor roads, major roads and expressways. Road type knowledge is important for route finding because people prefer to drive on expressways and major roads. A practical system should meet this preference.
- **Major roads naturally partition the whole road network**: When we look at the road map of a city, an important feature that can be observed is that major roads and expressways themselves also form a network (we call it the major road network) and they naturally partition the whole network into many small areas. We group major roads and expressways into one class and call them major roads because expressways normally make up only a small part of the road network in a city, which is our primary concern. Figure 3 (major roads are in thick lines and minor roads are in thin lines) shows a small part of Singapore road network.

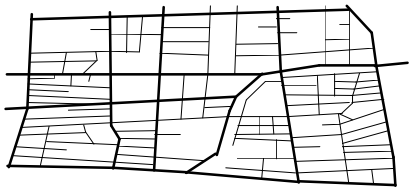

Figure 3. A part of the road network of Singapore

3.1.2. Using road network knowledge to form grids

Now we present the grid formation. We do not need to form the major grid as in Figure 2 as when *A** is used instead of Dijkstra algorithm, it automatically searches in the direction of the destination. Thus, only the two minor grids need to be formed to link the source and destination to the major road network.

It was noted that the major road network partitions the whole network into many small areas. All the minor roads in each area can be accessed from the major roads around them. Hence, we can statically assign each node (or junction) in the small area that area as its grid. Thus, the road network can be represented as a major road network and many small sub-networks (or grids). The main cases of grid formation are as follows (see [10] for details):

1) Major roads surrounding a set of minor roads form a grid for every node (or junction) in the minor roads. Figure 4 shows such a grid. The minor roads inside the grid are connected.

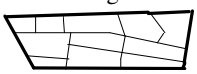

Figure 4. A grid sub-network formed by major roads

2) Although major roads naturally form many grids, the minor roads in a grid may not be connected. For example, Figure 5(A) shows a natural grid that has three unconnected parts (only connected via major roads). It is obvious that traveling to or from any node in one part will not need to pass the minor roads in any other part unless the source or the destination is in that part. Hence, three smaller grids may be formed (see B, C and D in Figure 5).

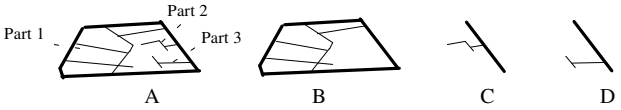

Figure 5. An area surrounded by some major roads

3) In a normal road network, there are underpasses and fly-overs. They make a road network non-planar. However, this issue was not properly handled in [9]. The old grid formation algorithm could not detect underpasses and fly-overs, which results in much bigger grids being formed. For example, in Figure 6(A), the road segment 1-2 does not meet 3-4. 1-2 is a underpass. Then, the old grid formation algorithm will form only one grid instead of two. In the worst case, a single grid could be formed containing the whole network because all the minor roads in the network could be connected via underpasses or fly-overs. This defeats the purpose of forming grids to isolate search. In Figure 6(A), actually two grid sub-networks could be formed as shown in Figure 6(B) and (C) with 1-2 appearing in both grids. In grid 1 (Figure 6(B)), the road 1-2 is represented as an arc from 2-1, while in grid 2 (Figure 6(C)) it is represented as an arc from 1-2. The reason for doing so is to ensure that from minor roads in grid 1 (or 2), the search will not go into grid 2 (or 1) via the underpass 1-2 unless the source and the destination of travel appears in grid 1 (or 2) and grid 2 (or 1) respectively. In general, there could be a number of underpasses between two grids.

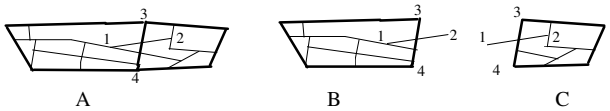

Figure 6. Grids with underpasses

Much of the grid formation can be done automatically except (3), which is currently done semi-automatically (with minimal human assistance). This appears to be quite amount of pre-processing. This is not a problem. It should be recognized that this only needs to be done once. Once constructed, it is used again and again.

3.2. Route finding: *disk-and-memory* design

We now describe a route finding design, called *disk-and-memory*. Before going into details, let us define some terms.

A ***grid sub-network*** is a grid formed in Section 3.1.2 without its major roads attached, i.e., only the minor road network in it. After extracting and storing all the grids, we are left with the skeleton of the major road network. This network contains (1) junctions of major roads, called ***major junctions***, and (2) junctions of both major and minor roads, called ***mixed junctions***. Between two major junctions, there may exist a number of ***mixed junctions***. Figure 7 shows the ***major road network*** after extracting all the grids from the network in Figure 3.

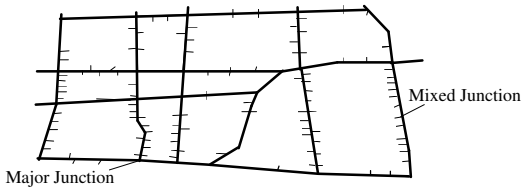

Figure 7. Major road network

If we remove the mixed junctions from the major road network, we obtain a condensed network that consists of only major junctions, which we call the ***major junction network*** **(MJN)**. The string of mixed junctions between two adjacent major junctions is called a ***detailed major road*** **(DMR)**.

3.2.1. Network representation and storage

According to our ideal situation, the network to be searched for solving a problem only includes the MJN, and at most two grids and the DMRs connecting the grids to the MJN. Thus, it is possible to store only the MJN in memory, while storing all the grids and the DMRs in the hard disk. When a query requires a node in a grid, that grid (sub-network) along with its relevant DMRs will be read in.

This method saves a great deal of memory because most of the roads in a typical road network are minor roads. The method is very useful in situations where it is difficult to store the whole network in memory. The bottleneck of the method is the slow i/o accesses.

3.2.2. Problem solving algorithm

Here, we present the problem solving algorithm for the *disk-and-memory* design. Due the handling of underpasses and fly-overs, the algorithm in [9] has to be modified to consider them. This is reflected in line 12 below.

```
1   Step 1. Link the source grid sub-network to the MJN:
2     If the source is a major junction then go to Step 2
3     elseif the source is a mixed junction then
4           retrieve and link the DMR containing the source to the MJN
5     else retrieve and link the source grid sub-network and its relevant
            DMRs to the MJN
6   Step 2. Link the destination grid sub-network to the MJN:
7     If the destination is a major junction then go to Step 3
8     elseif the destination is a mixed junction then
9       if the destination is on the same DMR as the source then go to Step 3
10      else retrieve and link the DMR containing the destination to the MJN
11    elseif the destination grid is the same as the one for the source then
            go to Step 3
12    else retrieve and link the destination grid sub-network and its
            relevant DMRs to the MJN (if an underpass or fly-over in the
            current grid represents the same road segment as in the previous
            grid, the two segments will be merged)
13  Step 3. Run the shortest path algorithm (e.g., A*) with the augmented MJN
14  Step 4. Remove the sub-networks added into the MJN.
```

More details about the algorithm can be found in [10]. The *disk-and-memory* design saves 90% of memory compared to a system that loads the entire network into memory with the Singapore road network. However, its bottleneck is the slow i/o accesses. When memory is not a limitation, all the various networks can be stored in memory. Then, the grid sub-networks and DMRs can be connected to the MJN very quickly with no i/o access. Two alternative designs were proposed and implemented, which are *Disjointed-networks* design *Grid-testing* design. Details on them can be found in [9, 10].

4 USING PAST EXPERIENCES AND CASES

Finding a good route efficiently only solves part of the route finding problem. It is also crucial for the system to learn from past experiences. If the system is to be used by individual users, it should produce personalized solutions that meet different users' preferences. Thus, two additional objectives need to be met:

1) Accepting user's feedback.
2) Ensuring that familiar routes (or roads) are used whenever possible and ensuring a reasonable compromise between using old routes and finding new ones.

The second objective suggests that we need to keep the past routes as cases. When a new problem is given, the system should try to make use of these past cases. Only when these cases cannot be used, will the search be performed. However, using old cases may not be that straightforward because the user may not like certain road segments of a route. Thus, a past case may not be easily used because the road level selection rather than the route level selection may be more important. This and the other reasons discussed earlier suggest the standard CBR approach may not be desirable.

Here, we present a simple approach that integrates well with the technique presented in Section 3. It does not have an explicit case base. The cases are implicitly represented in the network as follows: During travel, the user can feedback to the system of his/her experiences. If he/she dislikes a particular road, he/she can inform the system, which has a set of default actions to increase (or decrease) the road length (travel time or distance) in the network. The user can also change the length directly. If he/she does not express dissatisfaction with a road, the system will reduce the length of the road to reflect that this road has been visited before.

If we use the *disk-and-memory* design for route finding, when a major road's length needs to be increased or decreased, it will be reflected in the MJN (in memory) and its related DMRs in the hard disk. When a minor road in a grid sub-network is modified, the modification is reflected in its hard disk copy. In the other two designs, the modifications will be directly reflected in the MJN and sub-network(s) in memory.

This simple approach meets our two objectives. The first objective is met because the user is allowed to feedback to the system, and the system can modify the lengths of the related roads to reflect his/her personal preferences. The second objective is met because the visited roads have shorter lengths than they have originally. Then, a reasonable compromise between using old roads and finding new ones is automatically ensured.

Thus, we have avoided the use of CBR, and we do not have the problems associated with it. In building our first system R-Finder [8], we meet all those difficult problems. The technique used in R-Finder was so complicated, and yet it does not work well. This simple method integrated with the route finding technique in Section 3 solves all the problems. In addition, this method does not impose any memory overhead as a normal case base does.

5 EVALUATION

The aim of this project is to build a vehicle guidance system in Singapore. A prototype system, called KB-RFinder, has been built based on the *disk-and-memory* design. It uses the whole Singapore road network, which has a total of 12697 junctions and 30867 road segments. From the network, 958 grids and 503 detailed major roads are extracted. The MJN consists of 1096 major junctions and 2574 major road segments.

KB-RFinder is implemented in C on PC. Due to the space limitation, we will not describe it in detail. Here, we focus on the evaluation of the proposed techniques.

For our testing, we implemented four systems. The first one is based on the *disk-and-memory* design (*D-A-M*), the second based on the *disjointed-networks* design (*D-N*), and the third based on the *grid-testing* design (*GT*). For comparison, we also implemented a system that performs the search with no knowledge (*ZK*).

In Table 1, the first four sets of results show the performances of the four systems (using a 486 66MHz PC with 16MB of memory) without feedback and modification to the network. The cost measure is travel distance. The number of problems solved for each distance category is 100. All the results are the sums for solving the 100 problems. The three distance categories are *short*, *medium* and *long*, ranging from 0 - 15 km, 15 - 30 km and 30 - inf. km respectively.

Number of problems each category = 100				Cost measure: Travel distance	
		Dijkstra	A*		
		Running	Running	Travel	Travel
Method	Category	Time (sec.)	Time (sec.)	Dist. (m)	Time (min)
Without modification to the network.					
D-A-M	short	30.00	28.00	608120	749.43
	medium	29:00	27.00	2169321	2581.77
	long	27.00	24.00	3443898	4038.19
ZK	short	9.61	4.230	563404	754.42
	medium	30.27	14.72	1951254	2534.61
	long	39.55	27.47	3019790	3996.35
D-N	short	2.03	1.09	608120	749.43
	medium	2.64	2.03	2169321	2581.77
	long	3.10	3.00	3443898	4038.19
GT	short	2.85	1.98	608120	749.43
	medium	6.32	4.56	2169321	2581.77
	long	8.46	6.81	3443898	4038.19
With modification to the network.					
D-A-M	short	44.00	43.00	608120	749.43
	medium	44.00	42.00	2169321	2581.77
	long	42.00	40.00	3443898	4038.19

Table 1. Results with and without modification to the network

- It can be seen that *D-A-M*, *D-N* and *GT* produce the same paths as they all search the same roads. *ZK* finds the shortest paths as both *A** and Dijkstra algorithm guarantee the best solution.
- Regarding the travel distance, *D-A-M*, *D-N* and *GT* produce routes around 9% longer than the optimal routes produced by *ZK* (the travel times are similar). But, their routes meet people's preferences. These results can actually be greatly improved by making use of shortcuts. See [10] on how to do this.
- When Dijkstra algorithm is used, *D-N* improves the running time by 11 times comparing to *ZK*, and *GT* improves the running time by 4.5 times comparing to *ZK*. When *A** is used, the improvement is also substantial. As expected, *D-N* outperforms *GT* as *GT* needs to test each node along the search.
- The performance of *D-A-M* cannot be compared with *D-N* and *GT* since disk accesses are involved. However, it has similar performances as *ZK* for medium and long distance travels.

  It can be observed that the running times for different distance categories are not that different for *D-A-M*. This is because disk accesses dominate the computation in this design.

  It is also noted that the running time for long distance category is slightly shorter than for short distance category. The reason is that for short distance travels many grids could be in the city center, where a grid tends to have more minor roads in it than the outskirts of the city. For long distance travels, the grids are mainly in the outskirts of the city.

When using travel time as the cost measure, the running time performance comparison of various systems still applies. Due to the space limitation, we will not show the results here.

In Table 1, the last set of results shows the performances with modification to the network due to old cases. Only the results from *D-A-M* is shown as for the other systems there is little change from their original results. It can be seen that more time is taken when modification of the network is performed due to disk accesses. Note that travel distances and times are the same as those without modification because in our testing, we did not change travel distances or times, which will be done in real use.

6 RELATED WORK

Shortest path problem is a well-known problem. It has been studied for decades, and many algorithms have been produced [3, 4, 6, 11, 12]. However, limited research has been done in adapting those algorithms for practical route finding.

[14] proposed a technique to use level structure (which is similar to our major and minor road structure) of the road network to help searching for a path efficiently. Applying their technique to our problem, it will be like this: Assuming the source *S* and the destination *D* are both on some minor roads. The algorithm will first search for the nearest entry point *u* to the major road network from *S*. It will then do the likewise to *D* to find a node *v*. After that, it uses Dijkstra algorithm to search only the major roads to find a path linking *u* and *v*. This technique indeed minimizes the time traveling on minor roads and also greatly reduces the search space. However, the problem is that the two nodes *u* and *v* connecting *S* and *D* to the major roads could be on the wrong major roads. Then, the path found would be quite far from the shortest path.

Our approach does not have the problem of landing on the wrong major roads because all the major roads surrounding *S* and *D* are considered. Our approach also offers a way to partition the whole network into small sub-networks that can be stored separately.

A related AI-based system is ROUTER [4]. It uses both model-based method and case-based method (CBR) for route finding. The road network in ROUTER is represented as neighborhoods in a space hierarchy that is based on spatial proximity, level of detail and relative directions. Route finding consists of direction finding and route finding. The CBR method makes use of past experiences.

KB-RFinder is different from ROUTER in a number of ways. KB-RFinder integrates knowledge, an algorithm, and past cases for route finding while ROUTER uses only knowledge and the CBR approach. Using knowledge-based method together with CBR normally cannot find the best solution. It is also inefficient comparing to a good algorithm. The types of knowledge used in ROUTER are also different from those in KB-RFinder. KB-RFinder does not use the knowledge of neighborhoods based on spatial proximity and relative directions. This type of information is not required by KB-RFinder because of its search algorithm. ROUTER uses the traditional CBR approach, but not KB-RFinder. ROUTER's performance [4] also cannot compare with those presented in Section 5. Last but not least ROUTER stores the whole network in memory, but not KB-RFinder.

[5] describes a sophisticated case-based route finding system. The cases are connected as a planar case graph. For route finding, the system first triangulates the case graph together with the source and the destination into a Delaunay triangulation [2], which is then used for search to find the path in the triangulation. Since the solution path in the triangulation is only an approximation, the path has to be given to a separate planner to generate the final solution. This is a quite complex process. When the network is very irregular, the approximated solution could also be quite far from the optimal solution. In contrast, our system is much simpler. It guarantees good and human-oriented solutions. In addition, our *disk-and-memory* design can save a great deal of memory space.

Our earlier system R-Finder [8] uses Dijkstra algorithm, knowledge and the CBR approach. KB-RFinder has made substantial improvements over R-Finder. The most important improvements are on the grid formation and on the use of old cases.

In R-Finder, grids are dynamically assigned at run time for each problem, rather than formed statically for each node as in KB-RFinder. Each grid used in R-Finder is a rectangle that encloses a small area of the network. This approach has a number of problems. Firstly, it requires a great deal of network knowledge to decide the size and location of the grid (at run time). Secondly, R-Finder has the difficult problems associated with CBR. Thirdly, R-Finder does not save any space, and actually requires more due to the case base. Finally, the performance of R-Finder cannot compare with KB-RFinder if the *D-N* or *GT* design is used.

The improvements of this paper over [9] have been discussed earlier. There are two main aspects: handling of underpasses and fly-overs; allowing the user to feedback to the system so that it can produce personalized solutions.

7 CONCLUSION

This research proposes a novel approach for route finding, from finding routes to learning from past cases and user's feedback. It is computationally much more efficient than the existing approaches, and also produces human-oriented and personalized solutions. The proposed approach is general as the types of knowledge it uses are simple and generic to most modern road networks. It can be readily applied to build route finding systems in other places.

ACKNOWLEDGMENTS: I would like to thank Jimmy Tay and Teng-Hui Teo for implementing various parts of the system. I would also like to thank Land System Support Unit, Ministry of Law of Singapore for providing the Singapore road map data.

REFERENCES

[1] P. Bovy and E. Stern, *Route Choice: Wayfinding in Transport Networks, Studies in Industial Organization*. Kluwer Academic Publisher, 1990.

[2] L. P. Chew, "Constrained delaunay triangulations." *Algorithmica* 4, 97-108, 1989.

[3] G. Gallo, and S. Pallottino, "Shortest path methods: a unified approach," *Mathematical Programming Study*, 26,38-64, 1986.

[4] A.K Goel, K S. Ali, M W. Donnellan, A G de Silva Garza, and T J. Callantine. "Multistrategy adaptive path planning," *IEEE Expert*, 9(6), 57-64, 1994.

[5] K.Z. Haigh and M. Veloso, "Route planning by analogy," *Proceedings of International Conference on Case-based Reasoning*, 169-180, 1995,

[6] R.V. Helgason, J.L Kennington, "The one-to-one shortest-path problems: an empirical analysis with the two-tree Dijkstra algorithm," *Compu. Opt. and Appl*, 1, 45-75, 1993.

[7] G. F King and T M Mast, "Excess travel: causes, extent, and consequences," *Transport. Research Record*, 1111, 126-134.

[8] B. Liu, *et al*, "Finding the shortest route using cases, knowledge, and Dijkstra's algorithm," *IEEE Expert*, 9(5), 149-155, Oct. 1994.

[9] B. Liu, "Using Knowledge to Isolate Search in Route Finding," *IJCAI-95*, pp. 119-125, 1995.

[10] B. Liu, *Route finding by using knowledge, past cases and an search algorithm*, DISCS Technical Report, 1995.

[11] J. Pearl, *Heuristics.* Addison-Wesley, Reading, MA, 1988.

[12] I. Pohl, "Bi-directional search," *Machine Intelligence*, B Meltzer and D. Michie (eds), 6, 127-140, 1971.

[13] R. Sedgewick, *Algorithms*, Addison-Wesley Publishing Company, Chapter 11, 145-163, 1988.

[14] J. Shapiro, J. Waxman, and D. Nir, "Level graphs and approximate shortest path algorithm," *Networks*, 22, 1992.