RESEARCH PAPERS OF THE WROCŁAW UNIVERSITY OF ECONOMICS
No. 25 2008
Knowledge Acquisition and Management

Sebastian Ernst, Antoni Ligęza

AGH University of Science and Technology
Kraków, Poland
{ernst,ligeza}@agh.edu.pl

IMPROVEMENT OF ROUTE-PLANNING SYSTEMS USING INTELLIGENT AND KNOWLEDGE-BASED METHODS[1]

Abstract: Route planning is one of the major successful applications of automated planning (AI planning), a branch of Artificial Intelligence. However, classical planning methods are often too general when it comes to adaptation to real-life conditions and situations. This article provides an overview and suggestions for improvement of route-planning systems, particularly those used in urban conditions, with the help of knowledge engineering and other intelligent techniques. It also describes a new approach to route planning, based on the granular set theory, aimed at maintaining a set of paths instead of a single one. Such approach can provide significant improvements in agility, as well as introduce interaction with the user. Numerous enhancements of existing informed search algorithms are provided, along with commentary and references. Furthermore, the article suggests new methods and approaches for urban route planning, such as pre-computation of alternative routes, including guidelines for formal representation, solution robustness analysis and map abstraction with regard to decision-making points.

1. Introduction

Route planning and satellite navigation systems have been in use for several years, and it is hard to overestimate their usefulness in the modern world. Their performance is near-perfect in case of long journeys, but they suffer significant problems when used in urban conditions. The main reason is that they do not take the traffic conditions and other characteristics of urban travel into account. Most systems only use a limited set of criteria for finding the optimal route – usually they are limited to finding the shortest route or finding the theoretically quickest one – which is performed by assuming the estimated travel time for route segments, according to the road class, etc. This particular problem can be dealt with in two ways. One method is to create an infrastructure for monitoring the traffic and use that real-time

[1] This work is supported by MNiSW research grant N516 228635.

information to calculate a route that is optimal in practice. However, on-line traffic monitoring systems are still very rare, and that approach does not provide prediction of traffic changes as the travel progresses. Therefore, instead of monitoring the traffic in real time, it was decided to develop a set of methods for predicting conditions using artificial intelligence methods and knowledge describing the traffic characteristics of a given city [Ernst, Ligęza 2006].

Another drawback of route planning systems, particularly noticeable in mobile devices which are bound to provide real-time instructions for the driver, is the problem of *re-planning*. If the driver is forced to leave the previously calculated route, either due to a sudden event, change of traffic conditions or simply failure to exercise the instructions, the system has to calculate a new route. This has to be done quickly, before the vehicle gets to the next decision point (junction).

While re-planning cannot be avoided whatsoever, measures can be taken to avoid some situations which make re-planning necessary, and to assure that re-planning will be feasible.

2. Prediction of traffic conditions

As already mentioned in the Introduction, travel time for route fragments are usually computed using the length of a given fragment and the estimated travel speed, depending on the type of road (residential streets, collector roads, arterial roads).

Traffic camera footage and real-life observations have shown that the level of traffic can be quite accurately approximated using two normal distribution functions joined together (Fig. 1). That function has two peaks, one during the morning rush hours, as lots of people drive to work, and another one in the afternoon, representing

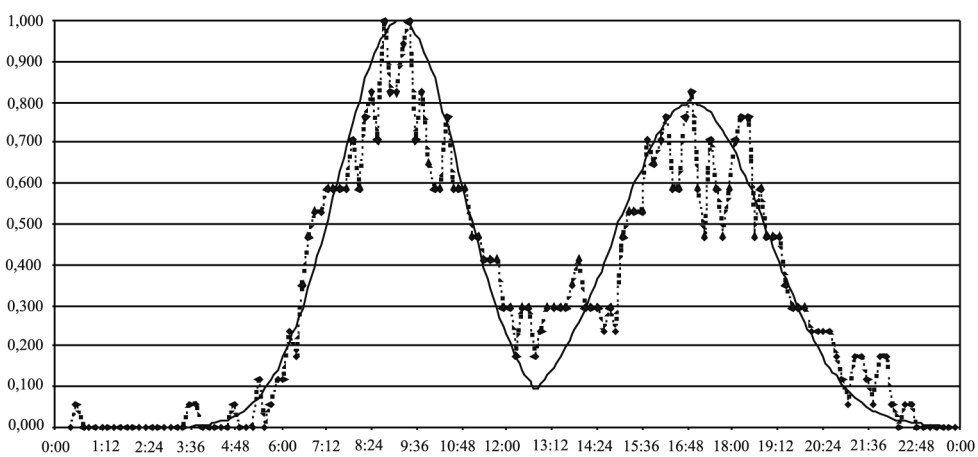

Fig. 1. An example traffic intensity graph. Dots represent real-life data

Source: [Ernst, Ligęza 2006].

their return home. Traffic data for the most important places in a city can then be interpolated for other streets using flow simulation or queue network algorithms [Helbing 1998].

The graph edge weights representing the travel times have to meet several conditions. Firstly, the travel times must be greater than zero, to ensure that there are no cycles with a zero or negative cost in the graph. In extreme cases, travel within the city can take several hours. As traffic conditions may change during that period, the algorithm has to use traffic level values for that exact moment. It is impossible for the traveller to save time by waiting at any location without moving, the travel times must satisfy the so-called *FIFO property* [Ahuja et al. 2002]. It is satisfied if for each pair t_1, t_2 ($t_1 < t_2$):

$$t_1 + d_{ij}(t_1) \leq t_2 + d_{ij}(t_2)$$

where $d_{ij}(t_1)$ is the time needed to travel from vertex i to vertex j at time t.

3. Existing algorithms for route planning

There are two different route planning situations in urban conditions. One occurs when the planning process is conducted before the travel begins. This is usually done by means of a route planning application or a web-based application. The other involves usage of mobile real-time satellite navigation systems.

3.1. A priori planning

Existing AI algorithms can be used for *a priori* urban route planning. Traffic conditions are incorporated into the edge weights, so it is possible to employ literally any shortest path algorithm to do the job [Ghallab et al. 2004].

It is, of course, possible to use any general shortest path search procedure, although for performance considerations we shall focus on informed search methods. Of those, the most widely known algorithm – A* search – seems well-suited to this purpose [Ramalingam, Reps 1991]. It is, however, important to notice that as the travel progresses, time advances as well, and that edge weights have to be updated so that they represent the traffic conditions at any given moment.

Selection of an appropriate heuristic function is very important for informed search methods. In our case, the weights of the graph edges ought to represent real-life travel time through a road segment. A heuristic function should therefore estimate the estimated travel time from a given point to the destination with maximum possible accuracy. The heuristic function does not take the traffic characteristics of streets into account, as that would require solving a route-planning sub-problem and therefore impact the performance. The nature of heuristic functions is not to use the search domain data itself, but instead to utilize some other kind of knowledge.

If the search domain is a map, the most obvious heuristic function refers to the distance between points, such as the straight-line or Manhattan distance metrics. For the A* algorithm to provide an optimal result, the heuristic function must be *admissible* – it must not overestimate the time required to get to the destination [Russell, Norvig 1995]. Because our heuristics are based on *distance*, they also have to take the *speed* into account, as the result must reflect the travel *time*. In order to obtain an admissible – optimistic – heuristic, the speed used for the calculations has to be the maximum allowed speed in the region where the search is performed.

3.2. Adaptive search

For the classic re-planning problem, the route needs to be constantly adjusted as the travel progresses. There are two common conditions causing this. One – as mentioned in the Introduction – occurs when the driver of the vehicle, for some reason, leaves the previously calculated route. The other cause for adjustments is the flow of time. The planning algorithm assumes that travel through respective route segments takes a certain amount of time. If the times are longer or shorter, the estimated times of arrival at certain points are no longer synchronized with the real arrival times, hence the need to alter the graph edge weights as well.

However, performance of repeated heuristic search can be improved. The idea here is to modify the heuristics in every subsequent search, so that it becomes more accurate, but maintains the properties of the original one – it remains admissible and monotonic.

One such algorithm, proposed by Koenig and Likhachev [2006], is called *Adaptive A**. Here, $gd[s]$ represents the goal distance of vertex s. Predictably, $f^* = gd[s_{start}]$ is the length of the shortest path from s_{start} to s_{goal}. In the original A* algorithm. Therefore, $f[s] = g[s] + h[s]$, so $f^*–g[s]$ can be used as the new, more accurate heuristic. As it is never smaller than the original $h[s]$ heuristic, it dominates it and should result in reduction of the number of states visited in subsequent searches. The authors further prove that the new heuristics are monotonically non-decreasing, and therefore become more "informed" over time, while maintaining the features of being consistent and admissible [Koenig, Likhachev 2006].

3.3. Incremental heuristic search

Incremental search is a common technique used for improving the performance of subsequent similar search operations. The basic idea of incremental search is that when changes are introduced in the search graph (i.e., the topology is altered or the edge weights are modified), the new search is not started "from scratch." Instead, the *start distances* (costs of the shortest path from the starting vertex to a given goal vertex) from the previous search are analyzed in order to identify those which could have changed after the graph had been altered.

One of the most popular uninformed incremental search algorithms is the DynamicSWSF-FP algorithm [Ramalingam, Reps 1991]. However, as it was already pointed out, urban route planning should be based on an informed (heuristic) search algorithm. Article [Koenig et al. 2004b] describes an algorithm, based on the classic A*, utilizing incremental search.

Lifelong Planning A* (LPA*) repeatedly searches for shortest paths between two vertices in graphs with dynamically assigned edge weights. As the modifications of weights are arbitrary, the algorithm seems well-suited for application in urban route planning. It maintains two estimates of the start distance: $g(s)$ and $rhs(s)$ (based on the g estimates of the successors) for each graph vertex s.

Initially, the shortest path is computed by a standard A* algorithm, generating pairs of equal estimates for each vertex. When the weight of any edge is changed, the *UpdateVertex* procedure is called for the destination vertex of that edge. The *rhs* estimate for that vertex is updated to reflect the updated weight, and, if changed, the vertex is added to a priority queue, according to the f and g values of a vertex in A*. When all the edges have been processed, the shortest path is recomputed. This time, only the vertices from the priority queue are taken into account, in order of priority. Estimates for those vertices are again made equal, and all the successors of each vertex are added to the priority queue [Koenig et al. 2004a].

Authors of the algorithm admit that LPA* could prove less efficient than A*, especially in the case when a large percentage of edges have had their weights modified, i.e. when the overlap between the old and the new search trees is small. In order to improve computational performance, it seems a good idea not to re-compute the shortest path every time there is a time mismatch. There are two ways of doing this. One – quite obvious – is to define the minimum time gap length triggering the edge weight update. However, that could be a problem for route segments with highly dynamic characteristics. Therefore, it is more advisable to compute the difference of the previous and the current travel time for each street segment, and update it only if it exceeds a certain value.

4. Alternative routes and solution robustness

To avoid the necessity of re-planning, multiple alternative routes can be calculated in advance. Generation of a set of suboptimal plans can be performed using informed search methods, such as A*, IDA*, etc., provided that multiple solutions, instead of just the best one, are maintained in the memory. When considering two or more paths, the paths can intersect at certain nodes, creating a possibility to switch between plans during plan execution. Therefore, no re-computation will be necessary, as the alternative plans have already been determined. It is therefore necessary to develop a formal way of representing a set of suboptimal plans. A *robust plan* is a bundle of plans with numerous switching points, providing the possibility to dynamically alter the selected plan, thus ensuring that the goal is reached in a time as close to the optimal solution as possible.

Let us assume that we have a set of paths (plans), intersecting at certain nodes. The basic element of any plan is a set of one or more paths joining two of such nodes. Such a structure will be called a *bunch*. An *elementary bunch* is a set of graph edges, connecting two selected nodes. A *bunch* in general will therefore be a set of paths between two graph nodes.

Any node which is the beginning of two or more edges will be referred to as a *branching node* or a *switching node*. Branching nodes provide the possibility to switch to an alternative route. The chances to achieve the goal node for such nodes can be roughly estimated as the sum of the numbers of paths of a certain length divided by the path length. That number shall be called the *robustness factor* for a given node, and can be used in the decision-making process when choosing from among the calculated alternative routes.

5. Multiple levels of map abstraction

Map abstraction and refinement is a technique widely used in computer game logic programming, in order to optimize the doings of the characters controlled by the computer. Theoretically, this approach also can be used for urban route planning.

5.1. Existing algorithms

Authors of [Sturtevant, Buro 2005] proposed a method of improving the performance of path-finding by using several levels of abstraction. The algorithm, *Partial Refinement A** (PRA*), begins with the most abstract version of input data. Then it executes a loop, reducing the abstraction level in each iteration. The loop performs an A* search and then truncates the obtained path to a given length. This way, only part of the search graph is refined at a time, hence the name.

The biggest problem here is the selection of abstraction levels and preparation of abstract data. Grid-type maps can be easily subjected to abstraction, e.g. by joining adjacent cells. Commonly used map file formats incorporate a system of levels, normally used to reduce the number of details displayed at lower zoom levels. This usually amounts to removal of "less significant" streets.

5.2. Application in route planning

The approach of removing streets (edges) of lower rank in higher abstraction levels is somehow contrary to the idea of calculating alternative routes, optimal in situations of high levels of traffic.

A slightly different approach to map abstraction is proposed here. In practical environments the planning process should be made hierarchical. It is important to

determine alternative landmark nodes through which a solution must pass. Such landmarks can be specified by bridge over a river to pass, frontier passes, tunnels, etc. To accomplish that, the granular set approach can be applied [Ligęza 2003].

Application in urban conditions makes it easy to determine natural divisions of the map (city). Usually, the number of ways of getting from one section to the other is limited – if a city is divided with a river and the start of the journey is on another side of the river than the goal, the driver will have to choose one of the available bridges to cross it. An example of a very abstract graph, used to find the possible connections between two sides of a river in a real city, is shown on Fig. 2.

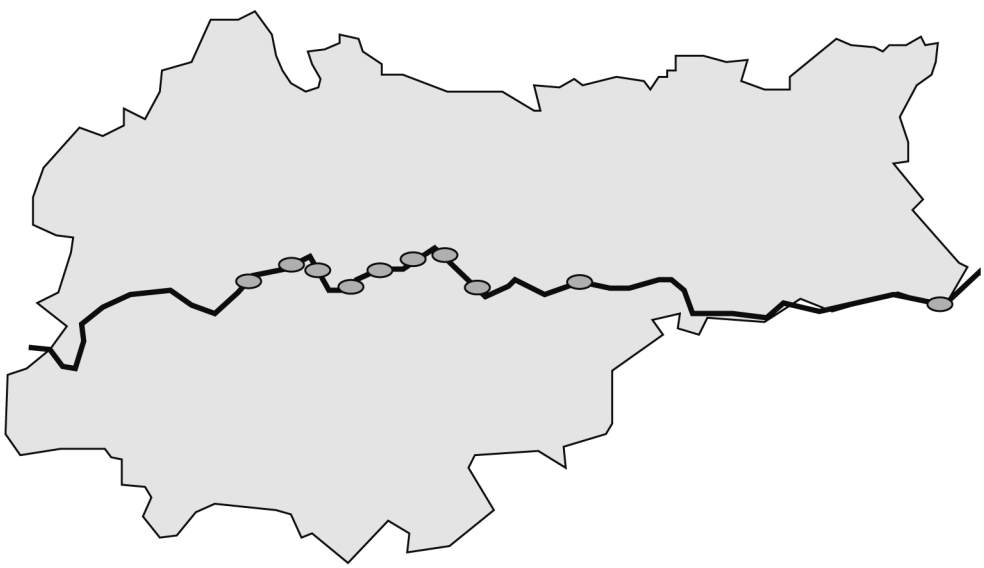

Fig. 2. A highly abstract map of Kraków, showing the relatively low number of possible places for crossing the river

Source: own elaboration.

Landmark nodes can be explicitly specified by the end-user as places of preference, or they can be computed automatically using certain heuristic functions (e.g. functions based on the concept of distance) or external rules.

6. Rule-based planning

The next step for the aforementioned approaches is the introduction of rule-based planning. This approach involves partitioning of the graph, similarly to the approach described in section 5, but with the use of formal definitions.

The approach includes definitions of partitions, partition links, neighboring partitions and traversal plans. The concept is quite self-explanatory and intuitive, but it

is worth to note that a partition link is a node connecting two neighboring partitions, and thus is the way of getting from one partition to another.

Currently, it is assumed that the process of map partitioning is performed manually, using expert knowledge. The primary guideline for partitioning the map is appropriate selection of abstraction levels and partition boundaries. This means that significant obstacles, such as landmarks, should be omitted using proper partitioning schemes. A classical example of this case is an irregular lake, which has to be avoided by going on either side of it. If the obstacle itself is not considered a partition boundary (i.e., a partition is missing), the selection of the partition link can be sub-optimal and result in a non-optimal solution.

After the input domain has been divided into partitions and sub-partitions, input data is compiled into traversal rules. A single traversal rule includes the partition ID, enter and exit link IDs, the computed traversal cost, optional validity values and the path to follow.

A traversal plan is therefore a series of traversal rules, leading from the start partition to the destination partition. Such approach allows pre-calculation of route segments while maintaining reasonable storage requirements. Figure 3 shows an example division of a planning problems into sub-problems.

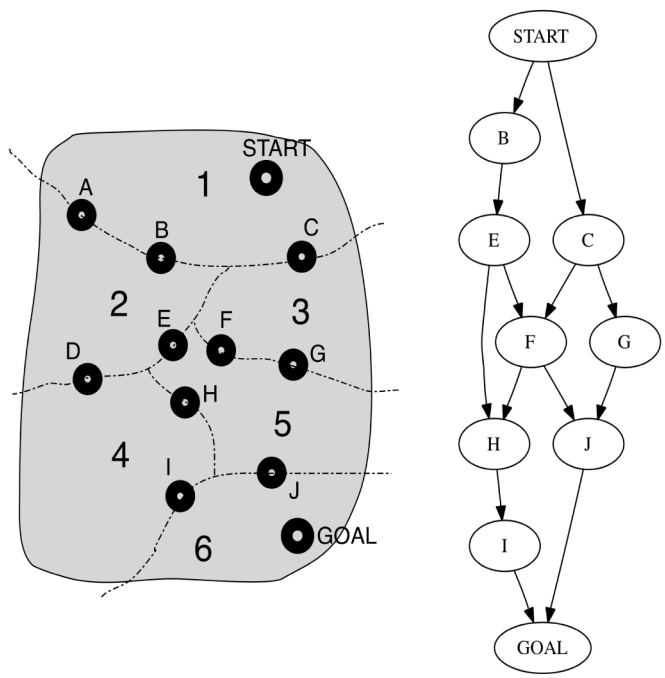

Fig. 3. An example division of a planning problem into subproblems

Article [Ernst, Ligęza 2008] decribes an algorithm which can be used to solve a planning problem using the rule-based method. The algorithm consists of two procedures, *Calculate point-to-point route* and *Traverse entire partition*. For the example shown on Fig. 3, the execution is as follows: first, the *Calculate point-to-point route* algorithm is used to solve the following subproblems: START→B, START→C, I→GOAL, J→GOAL; then, use the *Traverse entire partition* algorithm to solve the following subproblems: B→E, C→F, C→G, E→F, F→H, F→J, G→J, H→I. The algorithms have been described in detail in the aforementioned article.

7. Conclusion

Current route planning systems perform quite well in long-distance journey planning, but often provide unfeasible solutions in urban conditions. In this article, we have provided an overview of possible enhancements of route planning systems by utilising artificial intelligence and knowledge engineering methods. Application of those methods should make it possible to avoid some shortcomings of urban navigation systems.

Obviously, the methods presented herein require refinement and calibration before they are put to use in practical applications. Numerous combinations of the presented approaches are possible, for instance bunches described in section 4 can be enhanced with additional rules used to derive the plan execution strategy.

Moreover, pre-computation of alternative routes can be joined with map abstraction methods described within the article to divide the problem into smaller sub-problems, and therefore enable buffering of sub-problem solutions as well as make the plan generation problem feasible for parallel computing.

References

Ahuja R.K., Orlin J.B., Palottino S. and Scutellá M.G (2002). Dynamic shortest paths minimizing travel times and costs. In: *MIT Sloan Working Paper No. 4390-02*, Research Paper Series, MIT Sloan School of Management.

Helbing D. (1998). *Fundamentals of Traffic Flow*. University of Stuttgart.

Ernst S., Ligęza A. (2006). Application of knowledge engineering methods for urban route planning. In: *Proceedings of the "Inżynieria wiedzy i systemy ekspertowe" conference*. Wrocław, Poland.

Ernst S, LigęzaA. (2008). A rule-based approach to robust granular planning. In: *Proceedings of the International Multiconference on Computer Science and Information Technology.* Wisła, Poland.

Ghallab M., Nau D., Traverso P. (2004). *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers, San Francisco.

Koenig S., Likhachev M. (2006). A new principle for incremental heuristic search: Theoretical results. In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, Cumbria, UK.

Koenig S., Likhachev M., Furcy D. (2004a). Lifelong planning A*. *AI Magazine*, vol. 155, no. 1-2, pp. 93-146.

Koenig S., Likhachev M., Liu Y., Furcy D. (2004b). Incremental heuristic search in Artificial Intelligence. *AI Magazine*, vol. 25, no. 2, pp. 99-112.

Ligęza A. (2003). Granular sets and granular relations for algebraic knowledge management. In: *Smart Engineering Systems Design*. Eds. C.H. Dagli et al. ASME Press, New York, vol. 13, pp. 169-174.

Ramalingam G., Reps T. (1991). *On the computational complexity of incremental algorithms*. Computer Sciences Department, University of Wisconsin.

Russell S.J., Norvig P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall, New Jersey, USA.

Sturtevant M., Buro M. (2005). Partial pathfinding using map abstraction and refinement. In: *Twentieth National Conference on Artificial Intelligence*. Pittsburgh.