Constrained Route Planning Based on the Regular Expression

Jing Wang, Huiping Liu, and Zhao Zhang(✉)

School of Computer Science and Software Engineering,
East China Normal University, Shanghai, China
{jingwang,hpliu}@stu.ecnu.edu.cn, zhzhang@sei.ecnu.edu.cn

Abstract. Traditional route planning algorithms, which mainly focus on common metrics to find the optimal route from source to destination, are not enough to solve route planning requirements with location constraints like sequence, alternative and avoidance. For example, finding the shortest path passing the whole or a part of user-defined locations or location categories in order or disorder, or not passing some specified locations or categories. Mainly focusing on these scenarios, this paper formalizes the constrained route planning problem based on the regular expression generated by user requirements and gives a general framework for the exact solution. By using different shortest path algorithms, we show how the framework works efficiently with shortest path algorithms. Finally, extensive experiments on real road network datasets demonstrate the efficiency of our proposal.

Keywords: Constrained route planning · The regular expression
The shortest path · Dijkstra's algorithm · A* search algorithm

1 Introduction

Route planning based on metrics such as distance, time, cost, etc. [1–3], has been rapidly developed in recent decades. However, there are some route planning requirements constrained by locations which are common in daily life but cannot be solved by traditional route planning algorithms. Consider a road network in Fig. 1 where each node indicates a Point of Interest (POI), each directed edge represents a one-way route between POIs with an associated number indicating its length. Suppose after working at A, Tom prefers to go to a restaurant to have a meal first and then choose a cinema or bar to relax (he can only visit one of them because of insufficient time), after that he should go home at I, but along the way home he would like to avoid mall F due to traffic jams. Tom wants an ideal route which not only satisfies above location requirements but also has the minimum total length.

Similar problems have received significant attentions. Optimal route queries [4–13] search the shortest route that starts from the query point and passes through a user-defined category or keyword set in order, partial order or disorder. In spite of the significant contributions made by previous works, they are limited to solve route planning problems like the example in Fig. 1.

Therefore, in this paper, we study the problem of finding the optimal route for a fixed source and destination pair with given multiple location constraints like sequence, alternative and avoidance. Considering such route planning requirements always cover rich semantics, the regular expression, a character sequence to represent matching patterns that strings should be in accordance with, is adopted to express route planning queries. By taking advantages of existing operator definitions, the user-defined constraints can be expressed completely and precisely. First, we formalize the Constrained Route Planning Problem (CRPP) based on the regular expression. Then we devise a general framework, which supports shortest path algorithms such as the Dijkstra's algorithm and A* search algorithm, to solve the CRPP.

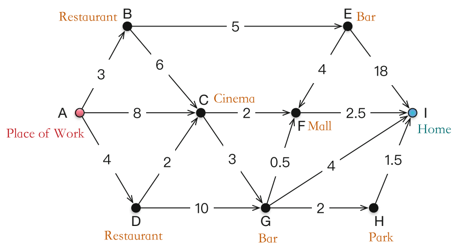


Fig. 1. An example of the constrained route planning problem.

The main contributions of this paper can be summarized as follows:

– We propose and formalize the CRPP based on the regular expression, that is to find the shortest path from the source to destination in accordance with customized location constraints like passing sequence, alternative and avoidance of locations or location categories.
– We devise a general framework to solve the CRPP with the exact solution, and implement it by using the Dijkstra's algorithm and A* search algorithm.
– Extensive experimental evaluations performed on real datasets show the efficiency of the proposed approaches.

The remainder of this paper is organized as follows. In Sect. 2, we briefly review related works. Section 3 gives necessary notations and a formal problem definition of this paper. Section 4 describes the general framework of our solutions, following two implementations: the Dijkstra-based CRPP algorithm and the A* search-based CRPP algorithm. Section 5 reports our extensive experiments. Finally Sect. 6 concludes the paper.

2 Related Work

The optimal route query with multiple constraints has been extensively investigated. Li et al. [4] propose the Trip Planning Query (TPQ) in spatial databases where each spatial object has a location and a category. The aim of the TPQ is to find the shortest path from the source to the destination that passes through at least one object from each category in a defined set of categories. In the TPQ, users cannot specify the passing order of categories, and [4] only gives approximate solutions. Sharifzadeh et al. [5] study an Optimal Sequenced Route (OSR) query which strives to find the shortest path from a fixed source that passes through at least one node from given category sets in a specified order. [5] gives solutions in Euclidean space - the LORD and R-LORD algorithms as well as in metric space - the PNE algorithm. By preprocessing given category sequences, a series of Additively Weighted Voronoi Diagrams (AWVD) are constructed to solve OSR queries effectively [6,7]. The Generalized Shortest Path (GSP) query [8] is essentially the OSR with a given destination and a strict order. By using dynamic programming formulation, [8] is more suitable for larger graphs. Different from the OSR, Chen et al. [9] consider the Multi-Rule Partial Sequenced Route (MRPSR) query where only partial ordered categories are specified. [10] solves the optimal route query from the source and passes through a user-defined set of categories, supporting partial order restrictions between some specific categories of the set.

[11] proposes the Multi-Approximate-Keyword Routing (MAKR) query, which defines not only the source and destination, but also a set of (keyword, threshold) pairs to find the shortest path that passes through at least one matching object per keyword with the matching degree higher than the corresponding threshold. Keyword-aware Optimal Route (KOR) query [12,13] gives approximate solutions to find a path from the source to the destination, covering a set of user-specified keywords, satisfying the travel budget and optimizing the route popularity. However, MAKR and KOR don't take keyword sequences into consideration, and adding keyword matching degree, path popularity, etc. is only suitable for specific scenarios that contain these information.

Also using the regular expression to define constrained route problems, main concerns of the formal-language-constrained shortest path problem [14] are edges where labels of edges on the route should satisfy the regular expression.

To the best of our knowledge, existing works mainly focus on constraints of location categories, while some do not allow the sequence constraints, some just give approximate solutions, some take additional consideration into factors such as the route popularity which only applies to specific scenarios, none of them solve the alternative and avoidance constraints. For more general constrained route planning problems, this paper takes a comprehensive consideration of sequence, alternative and avoidance constraints as well as gives the exact solution.

3 Preliminary

Definition 1 (Graph). A directed weighted graph $G(V, E)$ consists of a set of nodes V and a set of edges E. $v \in V$ represents a location with a category $v.c$. $e = (v_i, v_j)$ denotes a directed edge from $v_i \in V$ to $v_j \in V$ with an nonnegative weight $w(e)$. We define G as a general graph, it can be a road network where V is the set of POIs and E is the set of shortest paths between POIs.

Definition 2 (Constrained Route Query). The constrained route query, denoted as q, is a regular expression that makes up of five kinds of elements: the source s, the destination t, a set of locations in V, a set of location categories in V and six operators $\cdot$, $|$, $\wedge$, $*$, $+$, ( ) (see Table 1.).

Table 1. Operators in the Constrained Route Query

Operator	Meaning
Join $\cdot$ (can be omitted)	Indicates concatenation between locations or categories.
Alternative $\|$	Separates alternative locations or categories.
Non-operator $\wedge$	Connects avoidance locations or categories.
Kleene closure $*$	Indicates zero or more occurrences of locations or categories.
Positive closure $+$	Indicates at least one occurrences of locations or categories.
Brackets ( )	Defines the range of operators.

Generated by the combination of above elements, q can cover almost all kinds of route planning requirements constrained by locations. Thus the constrained route query of Tom can be represented as $q = work \cdot Restaurant \cdot (Cinema|Bar) \cdot (^\wedge mall) \cdot home$, where $work$, $home$ and $mall$ are specific locations while $Restaurant$, $Cinema$ and Bar are categories. Since a location can be regarded as a single category, locations can be solved by transferring them into categories. So following sections only discuss the processing of categories.

However, closure operators $*$ and $+$ can be simplified: (1) a certain category appearing zero or more times has no influence on route planning results and can be removed from q; (2) passing a certain category one or more times can be simplified as once according to properties of the shortest path. Therefore, we mainly focus on other four operators.

In traditional regular expression theory, a string (can be seen as a route of characters) is accorded with a regular expression only if they have a complete matching. For example, only 'ACFI' and 'ACGI' are satisfied for $AC(F|G)I$, one more or one less character are all unacceptable. But in CRPP, q only contains categories that users care about, also it is insignificant to define all passed categories in q. It is a key issue when applying the regular expression to the CRPP and the solution will be stated in 4.2.

Definition 3 (Constrained Route Planning Problem). Given a directed weighted graph $G(V, E)$ and a constrained route query q, constrained route planning problem based on the regular expression CRPP(G, q) intends to find the shortest path that satisfies q in G.

So the CRPP of Tom is: to find the shortest path that satisfies $q = work \cdot Restaurant \cdot (Cinema|Bar) \cdot (^\wedge mall) \cdot home$ in the graph illustrated in Fig. 1. Actually, his ideal result is $A \to D \to C \to G \to H \to I$ with the length of 12.5.

4 Solutions

4.1 The Framework

In order to solve the CRPP, a general processing framework is proposed by taking the shortest path algorithm as the main line while the constrained route planning query assists in filtering routes which don't meet location constraints.

First of all, q is parsed to a Deterministic Finite Automaton (DFA) represented by the transition table T, which is recorded as $(state, location/category) \to nextstate$. See an example in Figs. 2 and 3 for q of Tom. Particularly, different from traditional regular expression that $^\wedge$ only constrains the next character of a string, in CRPP one can't pass operand categories of $^\wedge$ until arriving at the next constrained categories in q. For example, Tom wants to avoid passing *mall* along his way from *Cinema* or *Bar* until getting *home*. Above definition of avoidance constraints needs a special processing: first the non-operator and its operands are stored together as a transition condition, for example, $(state_4, ^\wedge mall) \to state_6$; then add into T a new transition from the added transfer result state to itself with the non-condition, for example, $(state_6, ^\wedge mall) \to state_6$ (showed in dashed boxes in Figs. 2 and 3). Now there are two transitions of $state_6$ whose conditions have a overlap that *home* is satisfied both for *home* and $^\wedge mall$. In order to get the shortest qualified route as soon as possible, that is, to forward the route as much as possible, we give a lower priority to the non-condition so that every time facing transitions like $(state_6, home)$, it will first move to $state_7$, thus keep the determinacy of DFA. As a result, we can check the qualification of a route through T: a route is qualified if and only if the start state can be led to an accepting state in the order of nodes of this route.

(1, work) -> 2
(2, Restaurant) -> 3
(3, Cinema) -> 4
(3, Bar) -> 5
(4, ^mall) -> 6
(5, ^mall) -> 6
(6, ^mall) -> 6
(6, home) -> 7

Fig. 2. The state transition table.

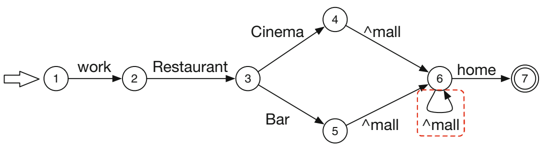


Fig. 3. The state transition diagram.

In this paper, we use two classical shortest path algorithm - Dijkstra's algorithm and A* search algorithm to show the implementation of our framework.

Because Dijkstra's algorithm is the basis of many shortest path algorithms and can be easily extended to them, while A* search algorithm is one of the most popular and effective target-directed methods by using a heuristic estimation.

4.2 The Dijkstra-Based CRPP Algorithm

Instead of the original Dijkstra's algorithm [15], we use a variant of it because its result is sure the shortest route but may be unqualified for location constraints. When updating the distance from s to v, we keep all partial routes not merely the shortest one, so that all possible qualified routes are taken into consideration. Particularly, a route will be discarded if some nodes on it are avoidance locations or belong to avoidance categories for the current state in T.

As stated in Definition 2, traditional regular expressions are limited in CRPP for they are a complete matching. To solve this, we define an arbitrary state *arb* to represent states not in T, *nowstate* and *paststate* to indicate the current state and its last non-arbitrary state, respectively. If a node satisfies transition conditions, it will be transferred directly; otherwise, since its subsequent nodes are likely to meet transition conditions, we mark its *nowstate* as *arb* and expand it according to *paststate*. A local priority queue LQ_v is used to store candidate routes from s to v, denoted as $rt(path, dist, nowstate, paststate)$ where *path* is the shortest path while *dist* is its length. Routes in LQ_v are sorted by *rt.dist* in an ascending order. Then a global priority queue GQ is used to preserve nodes during processing ordered incrementally by the minimal *rt.dist* in their LQs.

Algorithm 1. CRPP(G, T)

```
Input: graph $G$, transition table $T$
Output: the shortest path in $G$ satisfied $q$ and its length
1  Get the $startState$ and $endState$ in $T$;
2  Create $rt_s(rt_s.path.add(s),\ 0,\ T(startState, s),\ startState)$;
3  $LQ_s$.enqueue($rt_s$); $GQ$.enqueue($s$);
4  while $GQ$ is not empty do
5      $nd_i \leftarrow GQ$.dequeue(); $rt_i \leftarrow LQ_{nd_i}$.dequeue();
6      if $rt_i.nowstate{=}endState$ then
7          return $rt_i$;
8      else
9          if $LQ_{nd_i}$ is not empty then
10             $GQ$.enqueue($nd_i$);
11         GetLinks($G, T, GQ, nd_i, rt_i$);
```

However, above process will produce a lot of redundant routes, which surely cannot be the shortest qualified route. It is a waste of memory and computation resources to consider redundant routes, more importantly, they may be expanded in later steps, which will result in continuous unnecessary calculations. Based on above observations, we design two pruning rules to filter redundant routes.

Pruning Rules. When considering two candidate routes rt_i and rt_j of v:

(1) if $rt_i.nowstate = rt_j.nowstate \neq arb$, the longer route should be pruned;
(2) if $rt_i.nowstate = rt_j.nowstate = arb$ and $rt_i.paststate = rt_j.paststate$, the longer route should be pruned.

Proof. Suppose $rt_i.dist < rt_j.dist$, the shortest qualified path from s to t is $rt_j.path{+}p(v,t)$ where $p(v,t)$ is the path from v to t (+ means to concatenate two routes). That is, *endState* can be reached from $rt_j.nowstate$ (if $rt_j.nowstate \neq arb$) or $rt_j.paststate$ (if $rt_j.paststate = arb$) through $p(v,t)$. As rt_i and rt_j share the same state ($nowstate$ if $nowstate \neq arb$, otherwise $paststate$), rt_i can reach *endState* through $p(v,t)$ too. Because $rt_i.dist < rt_j.dist$, then $rt_i.path{+}p(v,t) < rt_j.path + p(v,t)$, which is a contradiction. If $rt_i.path + p(v,t)$ is not qualified, neither can $rt_j.path + p(v,t)$. So rt_i is always better than rt_j and we can safely discard rt_j without changing the correctness of algorithms. □

Algorithm 1 shows the details of Dijkstra-based CRPP (G,T) algorithm. It starts expanding nodes from s, then continuously takes the shortest path until its *nowstate* reaches *endState*. If GQ is empty, there is no qualified result at all. Otherwise the route will be expanded by invoking Algorithm 2 Getlinks(G,T,GQ,nd_i,rt_i) and during expanding, attributes of expanded routes will be set using aforementioned transition rules. Before adding to the priority queue, pruning rules are employed to check if it is a redundant route.

Algorithm 2. GetLinks(G,T,GQ,nd_i,rt_i)

```
Input: graph $G$, transition table $T$, global priority queue $GQ$, current node
       $nd_i$, the optimal candidate route to be extended $rt_i$
1 foreach directly linked node $u$ of $nd_i$ in $G$ do
2     Create $rt_k(rt_i.path.add(u), rt_i.dist + w(nd_i,u), nowstate, paststate)$;
3     if $u$ isn't in $GQ$ then
4         $LQ_u$.enqueue($rt_k$); $GQ$.enqueue($u$);
5     else if there exists a $rt_j$ in $LQ_u$ s.t. $rt_j.nowstate = rt_k.nowstate$ then
6         Pruning rules are used to decide which one should be held in $LQ_u$;
7     else
8         $LQ_u$.enqueue($rt_k$);
```

Since above algorithm is equivalent to continually find the next shortest path from s to t and check whether it is qualified for constraints, its validity is obvious. Also, the use of pruning rules only reduces the search space and improves the efficiency of the algorithm without deteriorating the correctness, so the validity of Dijkstra-based CRPP algorithm is proved.

4.3 The A* Search-Based CRPP Algorithm

In essence, A* search algorithm [16] works like Dijkstra's except that at each step it expands a node v with the minimal evaluation length $\hat{f}(v) = g(v)+\hat{h}(v)$, where $g(v)$ is the actual shortest length of the route from s to v and $\hat{h}(v)$ represents the estimated length from v to t. Hence, the process of A* search-based CRPP algorithm is nearly the same as Dijkstra-based only to add a new attribute *esdist* representing $\hat{f}(v)$, and redefine the priority of LQ as an ascending order of *esdist*. Intuitively, the correctness of A* search-based CRPP algorithm is proved.

5 Experimental Evaluation

5.1 Experimental Settings

All algorithms are written in Java and run on an Intel Core i5-4460 3.20 GHz CPU PC with 16 GB memory and a Windows platform.

Datasets. We use a real road network of California [4] with 68,345 nodes of 64 different categories and 137,980 edges. In order to investigate the impact of different graph sizes and graph structures on our solutions, we randomly choose a proportion of nodes and their relevant edges from the original graph to generate different sizes of subgraphs. The ratio of selected nodes is represented as the parameter *num*.

Queries. In order to ensure the generality of q, we set the normalized form of q as $s(c_1|c_2|...|c_k)_1()_2...()_j(\hat{}(c_1|c_2|...|c_l))t$. Assuming that each query has j categories that need to be connected sequentially, k choices for each of these categories, and after the current route there are l categories that cannot be passed (j, k, and l are all positive integers), generally, all constrained route queries can be expressed as this normalized form. So for each experiment, we generate 100 basic query instances with randomly chosen s, t and categories.

Parameter settings of the experiment are presented in Table 2.

Table 2. Parameter settings

Parameter	Range	Default value
num	25%, 50%, 75%, 100%	50%
j	1, 2, 3	2
k	1, 2, 3	2
l	0, 1, 2	0

5.2 Performance

We mainly focus on the response time and the number of expanded nodes which indicate the processing efficiency and the size of the search space, respectively.

In particular, we observe the number of nodes which are reduced to be extended by non-operators. To the best of our knowledge, there's no previous works that solve the same constrained route planning problem as ours, so we only compare Dijkstra-based CRPP algorithm and A* search-based CRPP algorithm, focusing on their performance differences. Note that below 'D-CRPP' and 'A-CRPP' both represent algorithms with pruning rules.

Effect of the Graph. We study the influence of G's scale and structure on our algorithms by varying *num*. Figures 4(a) and 5(a) show that the more number of nodes in a graph, the more search space and processing time the algorithm needs. It is because the average number of nodes of the shortest path in large graphs is often more than that in small graphs.

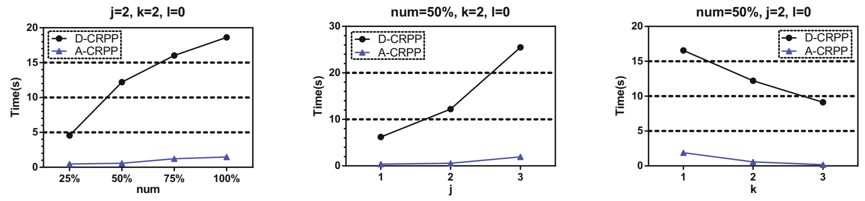
Effect of the Constrained Route Query. Figures 4(b) and 5(b) plot the performance when varying j. It can be seen that with the increase of j, the algorithm performance has a decline, that is because the more sequenced categories in q, the more calculations algorithm needs to check constraints.

In Figs. 4(c) and 5(c), the performance of algorithms falls with the increase of k. The reason is that when $k = 1$, there is a fixed category should be passed so it's equivalent to narrow search scope for algorithms; then when k increases, algorithms can only choose one among categories separated by $|$, which will accelerate the processing. Consider an extreme case that all categories in G are added as an alternative in each bracket of q, that is, $k = 64$, then every possible route is qualified.

Another important concern is l. See Fig. 6, with no surprise, more nodes are reduced to be extended because more candidate routes are excluded when l increases. It is interesting to note that there is a performance difference between D-CRPP and A-CRPP. With the increase of l, D-CRPP needs less processing time as well as less expanded nodes while A-CRPP needs more. This is because Dijkstra's algorithm only expands routes with the minimum length based on the current route, but A* search algorithm takes future conditions into account to expand routes with the shortest evaluation length from s to t. While we assume that the avoidance constraints appear at the end of the normalized form of q, many unqualified routes are used to estimate the shortest route in A*. And if a node on an expanded route belongs to the avoidance categories, this route will be discarded instead of transferred to an indeterminate state, that's why this phenomenon only appears when varying l rather than other parameters.

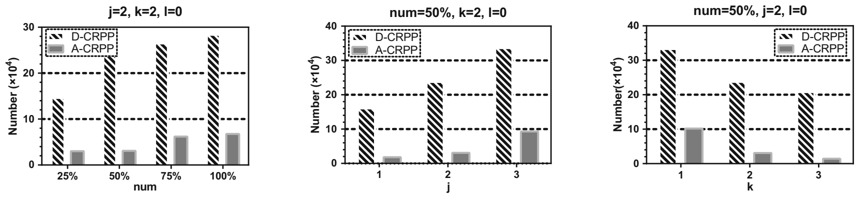
Effect of Pruning Rules. Assume the response time more than 3 min is unacceptable, in the process of experiment, we find that unacceptable query rates of both Dijkstra-based and A* search-based CRPP algorithms without pruning rules are nearly 100%, which indicates the good efficiency of pruning rules.

Comparison of Dijkstra-Based and A*-Search Based Algorithms. As shown in above diagrams, clearly A-CRPP outperforms D-CRPP in all cases. For the same kind of query, the performance of A-CRPP can be one order of magnitude better than D-CRPP. Also, as the size of G or constraints in q varies, the performance change of D-CRPP is faster than A-CRPP. However, since the



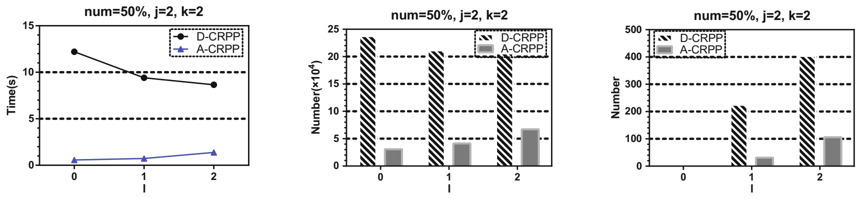
(a) Vary *num*. (b) Vary j. (c) Vary k.

Fig. 4. Average response time when varying *num*, j, k.



(a) Vary *num*. (b) Vary j. (c) Vary k.

Fig. 5. Average number of expanded nodes when varying *num*, j, k.



(a) Average response time. (b) Average number of expanded nodes. (c) Average number of reduced expanded nodes.

Fig. 6. Performance when varying l.

estimated length in A* search may base on routes which are in conflict with avoidance constraints, A-CRPP tends to get lower performance while D-CRPP gets better when avoidance constraints increase.

6 Conclusion

This paper investigates the constrained route planning problem, which is first formalized by defining the constrained route query as the regular expression, then gives a general framework for exact solutions. Subsequently, a Dijkstra-based CRPP algorithm and an A* search-based CRPP algorithm are proposed while utilizes pruning rules to prune redundant routes. Finally, we demonstrate the efficiency of our approaches through an extensive experimental evaluation.

Acknowledgements. Our research is supported by the National Key Research and Development Program of China (2016YFB1000905), National Natural Science Foundation of China (61370101, 61532021, U1501252, U1401256 and 61402180), Shanghai Knowledge Service Platform Project (No. ZF1213) and the Funding Program of International Academic Conference for Graduate Students at East China Normal University.

References

1. Liu, H., Jin, C., Zhou, A.: Popular route planning with travel cost estimation. In: DASFAA, (2), pp. 403–418 (2016)
2. Zhang, S., Qin, L., Zheng, Y., et al.: Effective and efficient: large-scale dynamic city express. IEEE TKDE **28**(12), 3203–3217 (2016)
3. Lu, H.C., Lin, C.Y., Tseng, V.S.: Trip-Mine: an efficient trip planning approach with travel time constraints. In: IEEE International Conference on Mobile Data Management, pp. 152–161 (2011)
4. Li, F., Cheng, D., Hadjieleftheriou, M., et al.: On trip planning queries in spatial databases. In: SSTD, pp. 273–290 (2005)
5. Sharifzadeh, M., Kolahdouzan, M., Shahabi, C.: The optimal sequenced route query. VLDB J. **17**(4), 765–787 (2008)
6. Sharifzadeh, M., Shahabi, C.: Additively weighted Voronoi diagrams for optimal sequenced route queries. In: Workshop on Spatio-Temporal Database Management. ACM (2009)
7. Sharifzadeh, M., Shahabi, C.: Processing optimal sequenced route queries using Voronoi diagrams. GeoInformatica **12**(4), 411–433 (2008)
8. Rice, M.N., Tsotras, V.J.: Engineering generalized shortest path queries. In: IEEE International Conference on Data Engineering, pp. 949–960 (2013)
9. Chen, H., Ku, W.S., Sun, M.T., et al.: The multi-rule partial sequenced route query. In: ACM GIS, pp. 1–10 (2008)
10. Li, J., Yang, Y., Mamoulis, N.: Optimal route queries with arbitrary order constraints. IEEE TKDE **25**(5), 1097–1110 (2013)
11. Yao, B., Tang, M., Li, F.: Multi-approximate-keyword routing in GIS data. In: ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 201–210 (2011)
12. Cao, X., Chen, L., Cong, G.: Keyword-aware optimal route search. PVLDB **5**(11), 1136–1147 (2012)
13. Cao, X., Chen, L., Cong, G., et al.: KORS: keyword-aware optimal route search system. In: IEEE International Conference on Data Engineering, pp. 1340–1343 (2013)
14. Barrett, C., Jacob, R., Marathe, M.: Formal language constrained path problems. Computing **30**(3), 234–245 (1998)
15. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1**(1), 269–271 (1959)
16. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for heuristic determination of minimum path cost. IEEE Trans. Syst. Sci. Cybern. SSC4 **4**(2), 100–107 (1968)