Proceedings of the Eighth International Symposium on Combinatorial Search (SoCS-2015)

A Preliminary Selection of Problems in Heuristic Search

Carlos Linares López
Computer Science Department
Universidad Carlos III de Madrid, Spain
carlos.linares@uc3m.es

Abdallah Saffidine
School of Computer Science and Engineering
The University of New South Wales, Australia
abdallahs@cse.unsw.edu.au

Abstract

The Heuristic Search community has been concentrating much effort during the last decades in solving more and more efficiently the SHORTEST PATH problem (SPP). As a result, a valuable body of scientific results has been produced, mostly in the form of heuristics and search algorithms. However, not much attention has been given to other problems even if they result from slight variations of the typical problems addressed by the community. Furthermore, other communities attempt at solving hard combinatorial problems which might be well solved with heuristic search. In this paper, an attempt is presented to introduce a preliminary selection of relevant problems that goes well beyond the classical SPP.

Introduction

A broad definition of *Combinatorial Search* is the study of search algorithms that are used to solve hard problems. All the algorithms considered in the field consist of different strategies for traversing the *state space* of a particular problem as efficiently as possible so that the solution is found with the minimum consumption of computational resources, mainly *time* and *memory*.

This broad definition embraces numerous different (but related) fields such as Operations Research, Graph Theory, and Discrete Optimization. Also, search algorithms are widely used in practice, for instance in Knowledge Engineering, Computer Vision, Machine Learning, and Robotics. In contraposition, the Heuristic Search community focuses on search algorithms themselves over their applicability to either theoretical or practical matters.[1]

The main focus of the Heuristic Search community so far has been the SHORTEST PATH problem (SPP). Although the SPP can be solved in polynomial time on explicitly given graphs, it is often NP-hard on graphs defined implicitly with a set of operators that act over states such as the *sliding-tile puzzle* or the *N-pancake*. Here, the Heuristic Search community has produced a valuable body of expertise, mainly in the form of new algorithms and *heuristics* that guide the search. This approach results in performances that are multiple orders of magnitude superior than when using *uninformed* search algorithms —i.e., when a heuristic function is not available.

Yet, the Heuristic Search community has either implicitly or explicitly made a number of assumptions that have restricted the focus of attention. As a result, the main body of scientific discoveries is mostly conceived to solve the SPP in various forms. However, in the past, a number of papers have suggested different problems. We want to join these efforts by analyzing these assumptions and attempting at motivating research that violates them in one way or another.

This paper is organized as follows: the next section introduces some notions of heuristic search and discusses some of the most notable assumptions in heuristic search. The next two sections show how these decisions affect the correctness of various search algorithms and heuristic functions. The following section introduces some theoretical problems that might be very well suited to search algorithms but which are fairly ignored by the community. The paper ends with some concluding remarks.

Definitions

The importance of graphs is that they can easily represent many problems of different nature. In most cases, vertices represent different states of the problem. A common meaning of an edge (u, v) is a causal relationship between one vertex and the next one —so that from u it is feasible by means of an *operator* to produce/reach v. From this perspective, the ability to efficiently solve graph problems directly relates to the ability to solve real-world problems whose *state space* can be formalized as a graph.

State spaces are often described succinctly using the concepts of state variable and operator. It is always possible to construct the corresponding *underlying* state space. We can thus focus the rest of this article on problems expressed on state spaces directly.

Definition 1 (State space) *A* state space *is a tuple* $\mathcal{S} = \langle V, E, C, s, t\rangle$*, where* V *is a set of states,* $s \in V$ *and* $t \in V$ *are the initial and goal states,* $E \subseteq V \times V$ *is a set of state transitions, and* $C : E \longrightarrow \mathbb{N}$ *is the edge cost function.*[2]

[1] The declaration of interests of the SoCS organization states: "*The purpose of SoCS is to promote the study and understanding of combinatorial search algorithms*".

[2] As usual, $\mathbb{N} = \{1, 2, 3, \dots\}$ and $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$.

The pair $\langle V, E\rangle$ constitutes a *directed graph* which can be *finite* or *infinite*. A similar definition could be given for undirected graphs. As such, states are also called *vertices* and state transitions are *edges*.

When the cost of some edges is equal to zero, pathological cases might arise. In consequence, the Heuristic Search community typically assumes that the cost of every edge is lower bounded by a constant, say 1. When a cost function is not given, it is assumed to map each edge to 1.

The Heuristic Search community assumes a single goal state, although goal states might not be unique in general. This assumption does not limit the applicability of Heuristic Search results as one can seamlessly transform a representation with multiple goal states to one with a single goal.

The most elementary problem on state spaces is the *reachability problem*, which consists of finding a path between the *initial* and the *goal states*.

Definition 2 *A* path $\pi = \langle n_0, n_1, n_2, ..., n_k\rangle$ *is a sequence of states such that each step corresponds to a valid edge* $(n_i, n_{i+1}) \in E$. *A* prefix *of* π *is a subsequence of states* $\langle n_0, n_1, n_2, ..., n_{k'}\rangle$ *where* $k' \leq k$. *A path is* simple *if no vertices are repeated,* $i \neq j \implies n_i \neq n_j$.

A *solution* to the reachability problem is therefore a path π such that $n_0 = s$ and $n_k = t$. The cost of the solution is defined as the sum of the costs of every edge in the path: $C_\pi = \sum_{\forall 0 \leq i < k} C(n_i, n_{i+1})$. In the forthcoming discussions, we will refer to C_π as the *objective function*.

The Heuristic Search community has developed a large body of algorithms and heuristics to solve problems that exhibit Bellman's "*principle of optimality*", a foundation of *dynamic programming*:

Definition 3 (Optimal policy) *An* optimal policy *is such that no matter the initial state and decision, the remaining decisions must constitute an* optimal policy *with regard to the state resulting from the first decision.*

This principle gives rise to a characterization of state spaces that satisfy it:

Definition 4 (Monotonicity of state spaces) *A search space is said to be* monotone *with respect to a state* s *if for any path* P *in* G *starting from* s *it holds that* P *is not* better *than any of its prefixes, where* better *is defined with regard to the objective function (Stern* et al. *2014).*

For example, the SPP is monotone with respect to the start state. Indeed, recall that individual edge costs are positive $C(u, v) > 0, \forall (u, v) \in E$, and thus any path P issued from s has a cost that is *monotonically increasing*.

The key observation is that the notion of *monotone state spaces* is more restrictive than the definition of the *principle of optimality*. As a matter of fact, some problems can be solved with *dynamic programming* and yet are very hard to solve with search algorithms. Often, the monotonicity of the state space is not even mentioned, yet it dramatically affects the applicability of various search algorithms and heuristic functions. The following section reviews this impact.

Search and Heuristics

We now survey the main mechanisms for generating heuristic functions and show how they are affected by the assumption of monotonicity of state spaces.

Search algorithms

The largest body (if not all) of algorithms developed in the Heuristic Search community are designed to traverse *monotone state spaces*. *Monotonicity* is, indeed, required for Dijkstra's algorithm to be applicable. Other important algorithms also relying on monotonicity include A*, IDA*, Linear-Space Best-First Search (RBFS) and DFBnB.

Heuristics

Another interesting object of study are heuristics, already introduced in the preceding section. The *trivial heuristic* for the SPP, h_0 that maps every state to 0, is admissible if the state space is monotone. Otherwise, when costs are not lower bounded, the trivial heuristic needs to be $h_0 = -\infty$.

Constraint relaxation

In the *constraint relaxation* procedure, an optimization problem is viewed as a collection of constraints, some of which are discarded so that the resulting *relaxed problem* is easier to solve optimally. The relaxation allows new transitions, and the cost of the optimal solution of the relaxed problem lower bounds that of the original problem. Some heuristics obtained this way are still state-of-the-art, e. g. (Zhang and Korf 1996; Junghanns 1999).

Let C_r^* be the cost of an optimal solution to the *relaxed* problem. If it can be proven that no paths with a cost strictly less than C_r^* reaches the goal state, then the cost of the optimal solution of the relaxed problem can be used as an admissible estimate for reaching the target from every node in the original problem. In case the state space is monotone, this verification can be done by extending paths until their cost is larger or equal than C_r^*. However, it this property is not satisfied, the test suggested does not work because there might be paths that are cheaper than their prefixes. In other words, computing the optimal solution to a relaxed problem may be as difficult as solving the original problem.

Abstractions

Abstractions, and especially Pattern Databases (PDBs), are a very powerful means for automatically producing heuristic functions (Korf 1997; Culberson and Schaeffer 1998). In PDBs, states that agree on a subset of variables (the *pattern*) are aggregated into a single one. Since abstractions effectively reduce the size of the state space, an analysis of the abstract space is more tractable (Helmert *et al.* 2014). A simple backwards breadth-first search from the goal pattern allows to compute the distance between each pattern and the goal pattern. If the state space is monotone then the values recorded by a single PDB are necessarily less or equal than the cost to reach any of the states aggregated into the same abstract state, and the PDB can be effectively used as an admissible heuristic. Otherwise, inadmissible estimates might result.

Theoretical problems

The ability to efficiently solve problems over state spaces directly translates into the ability to efficiently solve real problems that can be represented with such graphs. There are different *types* of combinatorial problems, from *optimally* solving to *counting* the number of solutions through *approximating* them or deciding their *existence*. However, we focus on different *classes* of problems defined over state spaces, usually confining our treatment to the optimally solving task. Furthermore, the classes suggested here refer to *classical* (deterministic, offline) heuristic search.

The Longest Path Problem

> Given a state space $\langle V, E, C, s, t\rangle$ find a simple path between s and t such that its cost is as large as possible.

This problem has been already considered in the Heuristic Search community motivated by different applications such as peer-to-peer information retrieval (Wong *et al.* 2005), multi-robot patrolling (Portugal and Rocha 2010), and VLSI design (Tseng *et al.* 2010). The state space is non-monotonic and optimal and suboptimal search algorithms for the SPP have to be modified to deal with this case (Stern *et al.* 2014). Among optimal search algorithms, DFBnB is slightly faster than A* and heuristic search algorithms have been observed to preserve their substantial advantage over uninformed search.

Although, the LONGEST PATH problem (LPP) in unweighted graphs can easily be shown to be NP-hard on arbitrary graphs with a reduction from the HAMILTONIAN PATH problem, the problem can be solved in polynomial time on some specific classes of graphs, including *trees*, block graphs, and cacti graphs (Uehara and Uno 2007), and rectangular grid graphs (Keshavarz-Kohjerdi *et al.* 2012). A *dynamic programming*-based algorithm can solve the LPP on interval graphs in $O(|V|^4)$ (Ioannidou *et al.* 2011), and the same problem has been solved for a larger class of graphs, cocomparability graphs (Mertzios and Corneil 2012).

Shortest Path Problem with Negative Costs

> Given a graph (V, E), two vertices $s, t \in V$ and an edge cost function $C : E \longrightarrow \mathbb{Z} \setminus \{0\}$, find a simple path between s and t with the minimum cost.

Note that the definition requires a *simple* path, as negative cycles would allow paths of arbitrarily low cost. It can be shown with a reduction from the LONGEST PATH problem that the SHORTEST PATH problem with negative costs is also NP-hard.

Negative edges make the state space non-monotonic and Dijkstra's algorithm is not applicable anymore. Indeed, the observation that a shortest path to a node to be expanded has been discovered is not an invariant anymore. When the graph has no negative cycles, the Bellman-Ford algorithm can be used instead and runs in $O(|V| \times |E|)$ (Bellman 1958). The most remarkable contributions are processing the vertices in first-in-first-out order (Yen 1975) and partitioning the input graph (Yen 1970). These ideas have been recently combined and extended with the idea of randomly permuting vertices resulting in a speedup version of the original algorithm (Bannister and Eppstein 2012).

Alternatively, Dijkstra's algorithm can be used by properly modifying the weight of each edge. This is essentially the main idea behind Johnson's algorithm (Johnson 1977) which, nonetheless, uses the Bellman-Ford search algorithm.

All these algorithms are based on *dynamic programming*.

Target-Value Search

> Given a graph (V, E), two vertices $s, t \in V$, an edge cost function $C : E \longrightarrow \mathbb{N}$ and a target value $T \in \mathbb{N}$, find a *simple* path between s and t such that its cost is as close as possible to T.

The problem originates from several significant applications, such as planning a tour of a given duration in a park, or model-based planning and diagnosis (Schmidt *et al.* 2009).

When $T \leq h^*(s)$, the target-value search problem is equivalent to the SPP. Otherwise, the problem is NP-hard, via reductions from SUBSET SUM and the LPP (Linares López *et al.* 2014)

The state space behaves non-monotonically for every path whose cost is strictly less than T, and monotonically for all paths with a cost strictly larger or equal to T. Prefixes with different costs might result in different deviations, therefore it is not possible to use a CLOSED list with A*. Additionally, the algorithm cannot be stopped once the goal has been expanded as additional paths have to be generated to prove that none can lower the deviation of the incumbent solution.

The current state of the art, T*, only addresses the case of unitary costs (Linares López *et al.* 2013). T* consists of a breadth-first search from s interleaved with a depth-first search from t, and a domain-independent heuristic based on *dynamic programming*. The searches are continued until the heuristic guarantees that the deviation of the incumbent solution with respect to the target value is minimal.

A related problem is the *bounded-cost search problem* where the goal is to find a solution with a cost smaller than or equal to a given fixed constant (Stern *et al.* 2011).

ANOTHER SOLUTION problem

> Given a state space $\langle V, E, C, s, t\rangle$ and a path π from s to t, find another path from s to t with the same cost.

The ANOTHER SOLUTION problem (ASP) was introduced to derive a finer characterization of previously studied NP-complete problems (Ueda and Nagao 1996). For instance, while both the classic 3SAT and VERTEX COVERING problems are both NP-complete, only the former remains hard under ASP considerations. A natural extension of the ASP is the more general n-ASP which provides n distinct solutions to the problem as input (Yato 2003).

The ASP for path finding remains polynomial for explicit graphs as it can be solved with the following algorithm. For each edge e of the input path, try to solve the SPP for $(V, E \setminus \{e\})$. If a path exists, then it constitutes a valid answer to the ASP. Since the input path is of polynomial length, at most a polynomial number of calls to SPP will be made.

Depending on the precise variant of ASP considered, solutions of differing cost may be accepted. If solutions of lower cost are desired, input solutions of high cost might provide a source of inspiration to find a better solution. For instance, the *Aras* postprocessor uses neighborhood searches to improve plans proposed by Monte Carlo and classical planners with relatively little computational effort (Nakhost and Müller 2010). Another interesting application of the ASP lies in puzzle design where solution unicity is often a desirable property.

K-optimal paths

> Given a state space $\langle V, E, C, s, t\rangle$ and a parameter $K \in \mathbb{N}$ find K simple paths between s and t with the minimum cost.

This problem has been extensively applied to many real-world problems, such as urban rail mass transit (Zhou *et al.* 2014), time-schedule networks (Jin *et al.* 2013), as well as probabilistic model checking when looking for small counter-examples (Han *et al.* 2009).

Depending on the setting, the full graph can be given explicitely or implicitely as in the SPP. In the online version, K is not known beforehand and new paths have to be generated one after another until the user stops the algorithm.

Eppstein's Algorithm (EA) is a notable algorithm to solve the KSP problem (Eppstein 1998) in time $O(|E| + |V| \log |V| + k)$. However, the graph needs to be provided in full from the start. The K* algorithm alleviates this requirement while maintaining low complexity (Aljazzar and Leue 2011) and takes advantage of admissible heuristics.

A typical enhancement of best-first search algorithms, such as Dijkstra's, is to prune paths that visit a node already expanded. Unfortunately, this optimisation is not applicable here as it could discard, say, the second best solutions. Strikingly, no comparative analysis has been performed with regard to depth-first searches (i. e., IDA*or DFBnB) or search algorithms that do not use a CLOSED list which do not have this limitation, the main problem being that transpositions would be re-expanded as many times as necessary.

Number of paths between two vertices

> Given a graph (V, E) and two vertices $s, t \in V$ find the number of paths from s to t.

Finding the number of paths between a given pair of vertice (s, t) is a #P-complete problem (Valiant 1979): it is as hard as counting the number of satisfying assignments in a SAT formula. It is quite remarkable that although finding a path in an explicit graph is (presumably) significantly easier than finding a single solution to SAT, counting the number of solutions is just as hard for both problems. While the hardness of the path counting problem might raise interesting theoretical questions, developing practical algorithms to address it is a pressing matter, especially in the field of network and routing protocols. Indeed, this problem is directly connected to the computation of the *two-terminal reliability* of a given pair of nodes in a network which quantifies the likelihood of there remaining a path between nodes in the event of link failures.

The problem can be easily solved using *dynamic programming* in Directed Acyclic Graphs (DAGs). The problem of counting the number of (s, t)-paths in both directed and undirected graphs has been addressed by the Monte Carlo community who proposed a stochastic algorithm (Roberts and Kroese 2007).

K edge-disjoint paths

> Given a state space $\langle V, E, C, s, t\rangle$ and a parameter $K \in \mathbb{N}$ find K edge-disjoint paths from s to t such that the sum of their costs is minimal.

Finding sets of edge-disjoint $s-t$ paths has numerous applications, including computing the size of the largest such set allows one to lower bound the *two-terminal reliability* of a graph (Papadimitratos *et al.* 2002).

Finding a shortest pair ($K = 2$) of edge-disjoint paths is a special case of minimum-cost network flow and a linear time algorithm for DAGs has been recently proposed (Tholey 2012). However, if length constraints are imposed over an undirected graph the problem is NP-complete (Tragoudas and Varol 1997). The problem has been studied in a wide variety of forms, including generalizations to k restricted edge-disjoint paths (Guo 2014). The node-disjoint variant has been applied to object tracking (Berclaz *et al.* 2011).

This problem concludes our preliminary selection of search problems and it appears to us as a perfect fit for the Heuristic Search community as long as it does not violate the monotonicity of the state space. Let a state describe K edge-disjoint paths from s. Extending these paths results in a larger value of the objective function than any of its prefixes provided that all edge costs are strictly positive. Furthermore, admissible heuristics based on either constraint relaxation or abstraction seem to be easy to derive as simple combinations of the heuristics for every path.

Conclusions

Search algorithms have been vastly applied to many different problems, both theoretical and practical. Although different problems have been studied by the Heuristic Search community, most efforts have concentrated around the SHORTEST PATH problem. This focus has resulted in relying consistently on various assumptions (such that the edge costs are strictly positive) and remarkably, that the state space is monotonic.

We first examined the consequences of dropping the monotonicity assumption on search algorithms and heuristics. Non-monotonicity affects the algorithms' termination and node re-expansion conditions and the heuristics' admissibility or tractability. We then proposed a selection of problem classes built on the same formalism as the SPP. They have important applications, have been studied in other communities, and pose refreshing challenges to our community.

Acknowledgements

The first author has been partially supported by the Spanish MINECO project PlanInteraction TIN2011-27652-C03-02. The second author has been supported by the Australian Research Council (project DE 150101351).

References

Husain Aljazzar and Stefan Leue. K*: A heuristic search algorithm for finding the k shortest paths. *Artificial Intelligence*, 175(18):2129–2154, 2011.

Michael J. Bannister and David Eppstein. Randomized speedup of the bellman-ford algorithm. In *ANALCO*, pages 41–47, Kyoto, Japan, January 2012.

Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

Jérôme Berclaz, François Fleuret, Engin Türetken, and Pascal Fua. Multiple object tracking using k-shortest paths optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(9):1806–1819, 2011.

Joseph C. Culberson and Jonathan Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

David Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1998.

Longkun Guo. *Frontiers in Algorithmics*, volume 8497 of *Lecture Notes in Computer Science*, chapter Improved LP-rounding Approximations for the k-Disjoint Restricted Shortest Paths Problem, pages 94–104. Springer International Publishing, 2014.

Tingting Han, Joost-Pieter Katoen, and Berteun Damman. Counterexample generation in probabilistic model checking. *IEEE Transactions on Software Engineering*, 35(2):241–257, 2009.

Malte Helmert, Patrik Haslum, Jörg Hoffmann, and Raz Nissim. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM*, 61(3):1–63, 2014.

K. Ioannidou, G. B. Mertzios, and S. Nikolopoulos. The longest path problem is polynomial on interval graphs. *Algoritmica*, 61:320–341, 2011.

Wen Jin, Shuiping Chen, and Hai Jiang. Finding the k shortest paths in a time-schedule network with constraints on arcs. *Computers & Operations Research*, 40(12):2975–2982, 2013.

Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM*, 24(1):1–13, 1977.

Andreas Junghanns. *Pushing the Limits: New Developments in Single-Agent Search*. PhD thesis, University of Alberta, Edmonton, Alberta, Canada, 1999.

Fatemeh Keshavarz-Kohjerdi, Alireza Bagheri, and Asghar Asgharian-Sardroud. A linear-time algorithm for the longest path problem in rectangular grid graphs. *Discrete Applied Mathematics*, 160:210–217, 2012.

Richard E. Korf. Finding optimal solutions to Rubik's cube using pattern databases. In *AAAI*, pages 700–705, 1997.

Carlos Linares López, Roni Stern, and Ariel Felner. Target-value search revisited. In *IJCAI*, pages 601–607, Beijing (China), August 2013.

Carlos Linares López, Roni Stern, and Ariel Felner. Solving the target-value search problem. In *SoCS*, pages 202–203, Prague, Czech Republic, August 2014.

George B. Mertzios and Derek G. Corneil. A simple polynomial algorithm for the longest path problem on cocomparability graphs. *SIAM Journal on Discrete Mathematics*, 26(3):940–963, 2012.

Hootan Nakhost and Martin Müller. Action elimination and plan neighborhood graph search: Two algorithms for plan improvement. In *ICAPS*, pages 121–128, Ontario, Canada, May 2010.

Panagiotis Papadimitratos, Zygmunt J. Haas, and Emin Gün Sirer. Path set selection in mobile ad hoc networks. In *MobiHoc*, pages 1–11, Lausanne, Switzerland, June 2002.

D. Portugal and R. Rocha. MSP algorithm: multi-robot patrolling based on territory allocation using balanced graph partitioning. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 1271–1276, Sierre, Switzerland, March 2010.

Ben Roberts and Dirk P. Kroese. Estimating the number of $s - t$ paths in a graph. *Journal of Graph Algorithms and Applications*, 11(1):195–214, 2007.

Tim Schmidt, Lukas Kuhn, Bob Price, Johan de Kleer, and Rong Zhou. A depth-first approach to target-value search. In *SoCS*, Lake Arrowhead, Canada, July 2009.

Roni Stern, Rami Puzis, and Ariel Felner. Potential search: A bounded-cost search algorithm. In *ICAPS*, pages 234–241, Freiburg, Germany, June 2011.

Roni Stern, Scott Kiesel, Rami Puzis, Ariel Felner, and Wheeler Ruml. Max is more than min: Solving maximization problems with heuristic search. In *SoCS*, pages 148–156, Prague, Czech Republic, August 2014.

Torsten Tholey. Linear time algorithms for two disjoint paths problems on directed acyclic graphs. *Theoretical Computer Science*, 465:35–48, 2012.

Spyros Tragoudas and Yaakov L. Varol. *Graph-Theoretic Concepts in Computer Science*, volume 1197 of *Lecture Notes in Computer Science*, chapter Computing disjoint paths with length constraints, pages 357–389. Springer Verlag Heidelberg, 1997.

I-Lun Tseng, Huan-Wen Chen, and Che-I Lee. Obstacle-aware longest-path routing with parallel MILP solvers. In *WCECS*, volume II, San Francisco, United States, October 2010.

Nobuhisa Ueda and Tadaaki Nagao. NP-completeness results for NONOGRAM via parsimonious reductions. Technical Report TR96-0008, Department of Computer Science, Tokyo Institute of Technology, May 1996.

R. Uehara and Y. Uno. On computing longest paths in small graph classes. *International Journal of Foundations of Computer Science*, 18(5):911–930, 2007.

Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

W. Y. Wong, T. P. Lau, and I. King. Information retrieval in P2P networks using genetic algorithm. In *WWW*, pages 922–923, Chiba, Japan, May 2005.

Takayuki Yato. Complexity and completeness of finding another solution and its application to puzzles. Master's thesis, University of Tokyo, Japan, January 2003.

J. Y. Yen. An algorithm for finding shortest routes from all source nodes to a given destination in general networks. *Quarterly of Applied Mathematics*, 27:526–530, 1970.

J. Y. Yen. *Shortest Path Network Problems*. Mathematical Systems in Economics (Book 18). Verlag Anton Hain, 1975.

Weixiong Zhang and Richard E. Korf. A study of complexity transitions on the asymmetric traveling salesman problem. *Artificial Intelligence*, 81:223–239, 1996.

Wei Teng Zhou, Bao Ming Han, and Hao Dong Yin. Study on the k-shortest paths searching algorithm of urban mass transit network based on the network characteristics. *Applied Mathematics and Materials*, 505-506:689–697, 2014.