Shortest Path for K Goals

Roni Stern
Ben Gurion University of the Negev
Be'er Sheva, Israel

Meir Goldenberg
The Jerusalem College of Technology
Jerusalem, Israel

Ariel Felner
Ben Gurion University of the Negev
Be'er Sheva, Israel

The *k-goal* problem is a generalization of the Shortest Path Problem (SPP) in which the task is to solve k SPP problems, such that all the problems share the same start vertex. kGP was introduced to the heuristic search community for building an Incremental Roadmap Spanner technique (Dobson and Bekris 2014), which is a useful construct in motion plannin for robotics. But kGP has many other applications, e.g., when path planning for multiple drones flying from a central dispatcher location to k target locations.

Problem Definition

Let $G = (V, E, w)$ be a weighted directed graph, where V is the set of states, E is the set of edges, and w is the edge cost function. A kGP problem is defined by a tuple $\langle G, s, \mathbf{g} \rangle$ where G is a graph s is state in G, and $\mathbf{g} = (g_1, g_2, \ldots, g_k)$ is a vector of k states. A solution to the kGP problem is a vector of k paths $\mathbf{p} = (p_1, \ldots p_k)$ such that for every $i \in [1, k]$ it holds that p_i is a lowest-cost path from s to g_i.

kGP is also known as a one-to-many shortest path query, and has been addressed in domains in which the graph can be stored in memory and processed upfront (Delling, Goldberg, and Werneck 2011). kGP is similar but different from the problem of finding the k best solutions to the same goals (Flerova, Marinescu, and Dechter 2016).

k Independent Searches vs. one k-Goal Search

A trivial algorithm for solving a kGP problem, which we call k×A*, is to run a SSP solver k times, one for each goal. k×A* explores parts of the search space multiple times, introducing potential redundancies. For example, consider the kGP problem in Figure 1(a), where k×A* would generate $1 + 2 + 3 + \cdots + (k + 1) = \frac{k(k+1)}{2}$ states while it is easy to see that generating k states is sufficient to solve this kGP problem. To avoid some of these redundancies, we propose kA*, an alternative algorithm in which all the k goals are searched together in a single pass. kA* is a generalization of A* designed to search for k goals in a single pass of the search space. There are several key aspects that differentiate between kA* and A*.

Maintaining the set of active goals When a goal is expanded in A*, the search halts. By contrast, in kA* the search does not halt until a lowest-cost path to each of the k goals has been found. To this end, kA* tracks the set of goals for which the lowest-cost path has not been found. This set of goals is referred to as the set of *active goals*, and the search halts when it is empty.

Multiple heuristics per state. When a state n is generated, kA* computes a k-ary vector $\mathbf{h}(n) = (h_1(n), \ldots h_k(n))$, where $h_i(n)$ is the heuristic estimate of the cost to get from n to goal g_i. These k heuristic values are used to form a k-ary vector $\mathbf{f}(n) = (f_1(n), \ldots f_k(n))$, where $f_i(n) = g(n) + h_i(n)$. We discuss later how the elements of this k-ary vector of f values are aggregated to a single value denoted $F(n)$ – which is used to prioritize OPEN. That is, in every iteration kA* selects from OPEN a state with the lowest F value. Note that the computational effort of generating a state in kA* is larger than that of regular A*, as it requires computing up to k heuristics. In practice, fewer computations may be needed, since we only need to compute the heuristics for the set of goals that are still active.

Conditions for Optimality

kA* is complete, i..e, if it does not find a solution then indeed no solution exists. The optimality of kA* depends on how the F value of a state is computed. We assume that $h_1, \ldots, h_k$ are all admissible heuristic functions, and consider two options: $F_{min}(n) = \min_{i \in [1,k]} f_i(n)$ and $F_{max}(n) = \max_{i \in [1,k]} f_i(n)$. kA* that uses $F_{min}(n)$ as the state evaluation function ($F(n)$) is referred to hereinafter as kA*$_{min}$, and similarly kA* that uses $F_{max}(n)$ is referred to as kA*$_{max}$. The following theorems state the relation between properties of the k heuristics and optimality. Proofs are omitted due to space constraints.

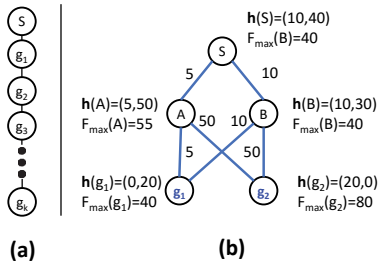

Figure 1: (a) k×A* inefficiency (b) kA*$_{max}$ suboptimal.

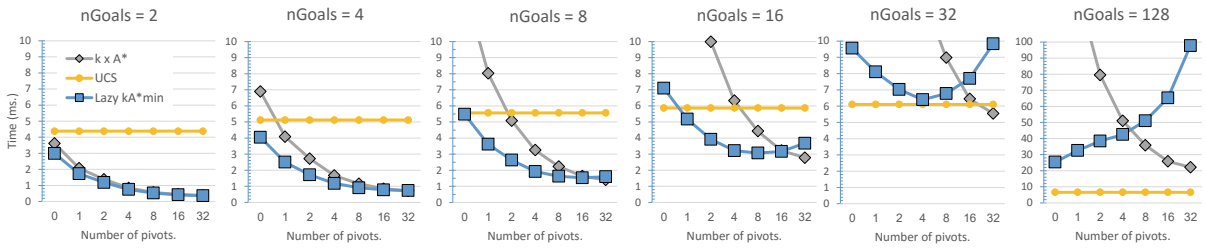

Figure 2: The runtime of Lazy kA$^*_{min}$, k×A*, and UCS for different number of goals (the different plots) and difference number of pivots using in the heuristic computation. With more pivots, the heuristic is more accurate but slower to compute.

Theorem 1 (kA$^*_{min}$ is admissible)**.** *kA$^*_{min}$ returns the lowest-cost path to each of the k goals.*

Running kA$^*_{max}$, however, may return a path π_i from s to g_i that is not optimal. For an example, see Figure 1(b), in which kA$^*_{max}$ halts with the suboptimal path to g_1 via B. However, if $h_1, \ldots, h_k$ are consistent, we can show that kA$^*_{max}$ does returns optimal solutions. However, in our experiments kA$^*_{max}$ was almost always worse than kA$^*_{min}$, so we focus the experimental results presented in this abstract on kA$^*_{min}$.

Experimental Results

We report here a small subset of our experimental results, focusing on grid path finding problems from the Dragon Age: Origin video game (Sturtevant 2012). We compared k×A*, an optimized version of kA$^*_{min}$ called Lazy kA$^*_{min}$, and Uniform Cost Search (UCS, i.e., Dijkstra's algorithm). As a heuristic, we used differential heuristics (DH) (Goldberg and Harrelson 2005; Ng and Zhang 2002; Sturtevant et al. 2009) instead of Octile distance, which is a sophisticated memory-based heuristic for grid pathfinding. DH can be tuned: adding more *pivots* to it results in a more accurate and more costly to compute heuristic. Figure 2 plots the average runtime in ms. as a function of the number of pivots, for 4, 8, 16, 32, 64, and 128 goals. The results show that the benefit over UCS diminishes as the number of goals grow. Eventually, when solving for 128 goals, UCS dominates both kA$^*_{min}$ and k×A*. This is reasonable To explain this, consider the most extreme case, where every state is a goal. In this case, clearly UCS will be the most effective, as it will expand every state at most once and will never spend any time on computing a heuristic value. By contrast, A* and kA* will compute at least one heuristic for every state, and most likely will compute even more heuristics per state. This heuristic computation time is not spent by UCS.

Next, consider the impact of adding pivots – i.e., using a stronger but more costly heuristic. For k×A*, we observe that adding pivots always helps reducing runtime. For kA$^*_{min}$, however, adding pivots improves runtime only for cases when the number of goals was not very large. For example, when $k = 32$, using more than 4 pivots actually degrades the performance of kA$^*_{min}$. This highlights a limitation in kA$^*_{min}$: it is sensitive the to cost of heuristic computation. In our ongoing research we are currently studying exactly how each operation in k×A* and kA$^*_{min}$ affect the overall runtime analysis, in an effort to provide useful guidelines for choosing the appropriate algorithm for a given problem.

This preliminary work is, to the best of our knowledge, the first study of using heuristic search techniques to solve kGP. There are several exciting directions for future work. First, we only explored in this work how to avoid some of the redundant computations done when performing k individual searches. A different way to benefit from searching for multiple goals is to learn valuable information about the underlying graph while searching for one goal, and use this information to to improve the search for the other goals. Another approach is to utilize concepts from incremental search algorithms such as Path-Adaptive A* (Hernández et al. 2015) for kGP.

Acknowledgments. Financial support for this research was in part provided by Israel Science Foundation (ISF) grant #417/13 and by the Cyber Security Research Center at Ben-Gurion University.

References

Delling, D.; Goldberg, A. V.; and Werneck, R. F. 2011. Faster batched shortest paths in road networks. In *OASIcs-OpenAccess Series in Informatics*, volume 20.

Dobson, A. J., and Bekris, K. E. 2014. Improved heuristic search for sparse motion planning data structures. In *the Annual Symposium on Combinatorial Search (SOCS)*.

Flerova, N.; Marinescu, R.; and Dechter, R. 2016. Searching for the m best solutions in graphical models. *Journal of Artificial Intelligence Research* 55:889–952.

Goldberg, A. V., and Harrelson, C. 2005. Computing the shortest path: A search meets graph theory. In *the annual ACM-SIAM Symposium on Discrete algorithms (SODA*, 156–165.

Hernández, C.; Uras, T.; Koenig, S.; Baier, J. A.; Sun, X.; and Meseguer, P. 2015. Reusing cost-minimal paths for goal-directed navigation in partially known terrains. *Autonomous Agents and Multi-Agent Systems* 29(5):850–895.

Ng, T. E., and Zhang, H. 2002. Predicting internet network distance with coordinates-based approaches. In *The Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 1, 170–179. IEEE.

Sturtevant, N. R.; Felner, A.; Barer, M.; Schaeffer, J.; and Burch, N. 2009. Memory-based heuristics for explicit state spaces. In *IJCAI*, 609–614.

Sturtevant, N. 2012. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games*.