Revisiting Bounded-Suboptimal Safe Interval Path Planning

Konstantin Yakovlev,[1,2] **Anton Andreychuk,**[1,3] **Roni Stern**[4,5]

[1]Federal Research Center for Computer Science and Control of Russian Academy of Sciences
[2]National Research University Higher School of Economics
[3]Peoples' Friendship University of Russia (RUDN University)
[4]Ben-Gurion University of the Negev, [5]Palo Alto Research Center (PARC)
yakovlev@isa.ru, andreychuk@mail.com, sternron@post.bgu.ac.il

Abstract

Safe-interval path planning (SIPP) is a powerful algorithm for finding a path in the presence of dynamic obstacles. SIPP returns provably optimal solutions. However, in many practical applications of SIPP such as path planning for robots, one would like to trade-off optimality for shorter planning time. In this paper we explore different ways to build a bounded-suboptimal SIPP and discuss their pros and cons. We compare the different bounded-suboptimal versions of SIPP experimentally. While there is no universal winner, the results provide insights into when each method should be used.

Introduction

Finding a shortest path in a graph is a classical problem in computer science with numerous applications, including robot motion planning, digital entertainment, and logistics. A* (Hart, Nilsson, and Raphael 1968) and Dijkstra's algorithm (Dijkstra 1959) are well-known methods for solving this kind of task. Finding a path becomes more challenging in the presence of dynamic obstacles that move through the environment, blocking vertices or prohibiting moving between some of them at specific time ranges. A solution in such a scenario is a *plan* that consists a sequence of *actions*, where an action is either to move from one vertex to an adjacent one, or to wait in it for some time. Application of textbook A* or Dijkstra's algorithm in this setting is not straightforward. First, the set of actions in a vertex depends on the current time, due to the dynamic obstacles. Second, there are potentially infinite wait actions, depending on how much time one would like to wait.

To address this problem, the *Safe Interval Path Planning* (SIPP) algorithm was introduced (Phillips and Likhachev 2011). SIPP computes for vertices in the graph a set of *safe intervals* in which it is possible to occupy them without colliding with the dynamic obstacles. Then, it runs an A* search in a different graph in which each vertex represents a pair of vertex in the original graph and a safe time interval. SIPP is complete and returns optimal solutions. It has been successfully applied in a range of domains, including robot motion planning and multi-agent path finding (Araki et al. 2017;

Andreychuk et al. 2019; Cohen et al. 2019). In such applications, a common requirement is to tradeoff solution optimality in order to obtain a solution faster. To control this tradeoff, we explore *bounded-suboptimal* versions of SIPP. A bounded-suboptimal algorithm accepts a parameter $w \geq 1$ and returns a solution whose cost is at most w times the cost of an optimal solution. Since SIPP is based on A*, it is natural to apply the same frameworks used for creating a bounded-suboptimal A*. However, the SIPP search space has certain properties that prevent a straightforward application of frameworks such as WA* (Pohl 1970). To address this, Narayanan et al. (2012) proposed a bounded-suboptimal SIPP implementation.

In this work we revisit the assumptions of this algorithm and propose two alternative bounded-suboptimal SIPP algorithms. One based on the WA* framework but allowing node re-expansions (see description later), and another based on the *focal search* framework (Pearl and Kim 1982). We analyze these algorithms and compare them experimentally. The results show that each algorithm has its strengths and weaknesses, and the choice of which algorithm should be use depends mainly on the value of w.

Problem Statement

Consider a mobile agent that navigates in an environment represented by a weighted graph. The vertices of the graph correspond to the locations the agent may occupy, and the edges correspond to allowed transitions. When at a vertex, the agent can either *wait* for an arbitrary amount of time, or *move* to an adjacent vertex along a graph edge. For the sake of simplicity we neglect the inertial effects and assume that agent moves with constant speed such that the duration of a move action equals the weight of the corresponding edge.

A *plan* $\pi(s, g)$ is a sequence of consecutive actions that move the agent from a start vertex s to goal vertex g. The cost of a plan is the sum of durations of its constituent actions. There exist dynamic obstacles that move in the environment and block certain vertices and edges for pre-defined time intervals, preventing the agent from occupying or moving through them. A plan $\pi(s, g)$ is called *valid* if it avoids colliding with all dynamic obstacles. The *path planning with dynamic obstacles problem* is the problem of finding a valid

plan for a given start and goal locations.

An *optimal* solution to this problem is a lowest-cost valid plan from start to goal. The *suboptimality* of a solution is the ratio between its cost and the cost of an optimal solution. In this work we are interested in finding solutions whose suboptimality is bounded by a given scalar $w \geq 1$. E.g. by setting w to 1.1 we aim to find a solution whose cost is no more than 10% over the cost of an optimal solution.

Background

A* is a heuristic search algorithm for finding a path in a state space represented as a graph. It maintains two lists of states, OPEN and CLOSED. OPEN contains all states that were generated but not expanded and CLOSED contains all previously expanded states. Initially, CLOSED is empty and OPEN contains only the initial state. Every state n is associated with two values: $g(n)$, which is the cost of the lowest-cost path found so far from the initial state to n, and $h(n)$, which is a heuristic estimate of the cost of the lowest-cost path from n to a goal. In every iteration the state with the minimal $f(n) = g(n) + h(n)$ is popped out of OPEN and *expanded*. It means generating the *successors*, which are state's neighbors in the state space, and inserting them into OPEN. A* halts when it expands a goal state. Note that if a state n generates a state n' that was already generated, then we must check if its g value can be updated by considering reaching it via n. If this happens and n' is no longer in OPEN, then it must be re-inserted into OPEN. Consequently, a node may be *re-expanded* multiple times. In fact, in some extreme cases the number of expanded states can be exponential in the size of the state space (Martelli 1977).

A* has several attractive properties that are relevant for this paper. First, if $h(n)$ is *admissible*, that is, it always outputs a lower bound on the value it estimates, then A* guarantees finding an optimal solution (if one exists). Second, if heuristic is *consistent*, then A* will never re-expand a state. A heuristic is consistent if for every pair of states n and n' such that n generates n' it holds that $h(n) - h(n') \leq c(n, n')$ where $c(n, n')$ is the cost of the edge from n to n'.

SIPP is a modification of A* for the path planning with dynamic obstacles problem. The core idea of SIPP is to group consecutive time moments into *time intervals* and to associate every vertex with one or more safe intervals. A safe interval for a vertex is "a contiguous period of time ... during which there is no collision and it is in collision one timestep prior and one timestep after the period" (Phillips and Likhachev 2011). SIPP performs an A* search over a state space in which a state is a tuple $(v, [t_i, t_j])$, where v is a vertex in the underlying graph and $[t_i, t_j]$ is a *safe interval* for v. Note that states with different non-overlapping time intervals but the same vertex might exist in the search space.

$g(n)$ in SIPP is the earliest time an agent can reach v in the designated safe interval $[t_i, t_j]$. SIPP uses $g(n)$ to compute the set of successors for n. I.e., when a state $n = (v, [t_i, t_j])$ generates a state $n' = (v', [t'_i, t'_j])$, SIPP first tries to commit a move $v \rightarrow v'$ at time equal to $g(n)$, that is without any waiting at n. If such a transition results in a collision with a dynamic obstacle, then SIPP aug-

ments the move with a wait action of minimal duration. That is, the agent is planned to wait at v no longer than it is needed to avoid the collision. Therefore, $g(n')$ is set to be $g(n) + dur(min_{wait}) + c(v, v')$, where $c(v, v')$ is the cost of the edge (v, v') and $dur(min_{wait})$ is computed taking dynamic obstacles into account.[1]

$h(n)$ in SIPP estimates the lowest-cost plan to move the agent from n to a goal state. SIPP requires $h(n)$ to be both admissible and consistent. Beyond these changes, SIPP uses regular A*: it extract from OPEN the state with the lowest $g(n) + h(n)$ and it halts when a goal has been expanded.

SIPP shares several of the desirable properties of A*. It guarantees finding a solution if one exists (otherwise, it reports *failure*) and this solution is optimal. Both completeness and optimality rely on the following property: when SIPP expands a state $n = (v, [t_i, t_j])$, then $g(n)$ is the earliest possible arrival time to v in $[t_i, t_j]$ (Theorem 2 in (Phillips and Likhachev 2011)). This property guarantees that set of successors for every generated state is maximal (Theorem 1 in (Phillips and Likhachev 2011)). Consequently, expanding a state with the goal vertex means the lowest cost plan to it has been found and all relevant states have been generated.

Bounded-Suboptimal SIPP

One of the well-known ways to make A* bounded-suboptimal is to use an inflated heuristic function during the search. WA* (Pohl 1970) is a prominent example of this approach. WA* is similar to A* except that it chooses for the expansion a node n that minimizes $g(n) + w \cdot h(m)$, where $w \geq 1$ is the desired suboptimality bound. When a goal state is expanded, WA* is guaranteed to have found a solution whose suboptimality is at most w.

Unlike A*, when WA* expands a state n it may be that $g(n)$ is *not* the lowest cost path to n. Thus, a state may expanded more than once. Nevertheless, if $h(n)$ is consistent, WA* is guaranteed to find a solution with the desired suboptimality bound even without re-expanding a single state (Likhachev, Gordon, and Thrun 2004).

Weighted SIPP (WSIPP)

Let Weighted SIPP (WSIPP) be SIPP that uses an inflated heuristic like WA*. WSIPP is not guaranteed to find a bounded-suboptimal solution, or in fact, any solution, if it does not re-expand states. For example, see Figure 1.[2] A dynamic obstacle moves from A to B, arriving at B at time 10 and staying there forever. Thus, $[0, 10]$ is the only safe interval for B. The safe interval for all states except A and B is $[-\infty, \infty]$. Let $w = 2$. After the first expansion states $n_D = (D, [-\infty, \infty])$ and $n_E = (E, [-\infty, \infty])$ are generated. n_E is chosen for expansion as $f(n_E) = 4 + 7 \cdot 2 < f(n_D) = 3 + 8 \cdot 2$. Next, node $n_C = (C, [-\infty, \infty])$ is generated and its g-value is set: $g(n_C) = 8$. Then n_C is expanded but it has no successors, as one can not reach B within the safe interval $[0, 10]$ from C when $g(n_C) = 8$ (as $8 + 3 > 10$). To find a solution, we must expand n_D and then re-expand n_C to update $g(n_C)$ to 6.

---

[1] The method to compute $dur(min_{wait})$ is domain dependent.

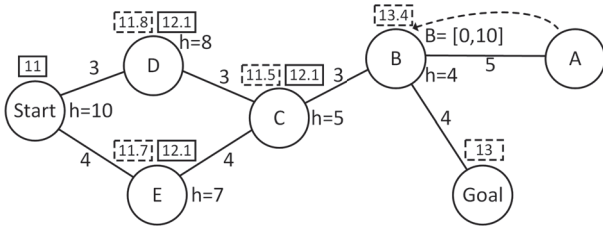[2] The rectangles above the vertices are explained later.

Figure 1: A running example for WSIPP.

Weighted SIPP with Duplicate States The creators of SIPP identified this problem and proposed the following. When the initial state is expanded, we create two copies of every state it generates. An *optimal copy*, which is prioritized in OPEN according to $w \cdot (g + h)$ and a *suboptimal copy*, which is prioritized in OPEN according to $g + w \cdot h$. Throughout the search, whenever an optimal copy is expanded, we again generate two copies of every state it generates. We refer to this algorithm as *WSIPP with Duplicate States* (WSIPP$_d$). WSIPP$_d$ preserves the desirable property of WA*– it guarantees returning a bounded-suboptimal solution while avoiding re-expansions. That is, every copy of a state is expanded at most once.

Weighted SIPP with Re-expansions (WSIPP$_r$) Interestingly, the creators of SIPP did not explore the possibility of allowing unlimited re-expansions instead of duplicating each state. That is, if a state in CLOSED is generated with a lower g value, then it is re-inserted to OPEN. Thus, every state will eventually be re-expanded with the minimal g value that guarantees finding a solution. We call this algorithm *WSIPP with Re-expansions* (WSIPP$_r$).

WSIPP$_r$ is complete and is guaranteed to find a bounded-suboptimal solution, but the number of re-expansions performed by WSIPP$_r$ can be exponential (Martelli 1977). Nevertheless, in many domains the number of re-expansions is manageable (Sepetnitsky, Felner, and Stern 2016).

In fact, WSIPP$_r$ may expand fewer states than WSIPP$_d$. Consider Fig. 1. Rectangles over the vertices show the states expanded by WSIPP$_d$ with $w = 1.1$, where the optimal and sub-optimal copies of each state are marked by a solid and dashed line. The values inside the rectangles are f values. After the initial expansion, WSIPP$_d$ expands sub-optimal n_E with $f = 11.7$ and generates sub-optimal n_C with $f = 13.5$. Then sup-optimal n_D ($f = 11.8$) is expanded. This adjusts the f value of sub-optimal n_C to 11.5. Next, WSIPP$_d$ expands the latter and generates sub-optimal n_B with $f = 13.4$. Then optimal n_D and optimal n_E are consequently expanded as their f values (both equal to 12.1) are lower than 13.4. As a result of these expansions optimal n_C is generated and its f value is set to 12.1. It is expanded and optimal n_B with $f = 14.3$ is generated. Then WSIPP$_d$ switches back to sup-optimal n_B (as $13.4 < 14.3$) and expands it generating sub-optimal copy of *Goal* with $f = 13$. Finally, the latter is expanded and the search terminates. Total number of expansions is 9. In the same setting WSIPP$_r$ performs only 6 expansions: $Start, n_E, n_D, n_C, n_B, Goal$,

with no re-expansions at all.

Focal SIPP

Focal Search (Pearl and Kim 1982) is a framework for bounded-suboptimal search that maintains a sublist of OPEN, called FOCAL. FOCAL is the set of every state n for which $g(n) + h(n) \leq w \cdot f_{min}$, where f_{min} is the smallest f value over all states in OPEN. In every iteration, a state n is chosen from FOCAL that minimizes a secondary heuristic $h_F(n)$. The latter does not have to be consistent or even admissible. A well-known secondary heuristics is the number of hops-to-the-goal (Wilt and Ruml 2014), which ignores edge costs. Focal Search terminates when a goal state is in the FOCAL or when OPEN is empty.

We use the name *Focal SIPP* (FocalSIPP) to refer to the SIPP version that uses Focal Search instead of A* and allows unlimited re-expansions. FocalSIPP is complete and guarantees finding a bounded-suboptimal solution. However, due to re-expansions its runtime may be exponential in the number of states in the search space, just like WSIPP$_r$.

Discussion

WSIPP$_d$, WSIPP$_r$, and FocalSIPP are guaranteed to return a bounded-suboptimal solution. However, the cost of the solution they return may differ, since it can be any value between the optimal cost and w times that cost. Perhaps, more interesting, though, is the expected runtime for each algorithm for a given value of w.

Consider setting w to be very close to one. In this case, WSIPP$_d$ is expected to perform poorly, since two copies of every generated state are introduced and both are likely to have nearly the same priority in OPEN, since $w \cdot (g + h)$ and $g + w \cdot h$ are very close if w is close to one, especially in the beginning of the search. In contrast, WSIPP$_r$ is expected to perform well, as it is almost equivalent to A*, which is known to expand the minimal number of states (Dechter and Pearl 1985). On the other hand, consider the behavior of FocalSIPP for very large values of w. In such a case almost all states in OPEN will be in FOCAL and FocalSIPP basically performs a greedy best-first search towards a goal, which is specifically designed to reach a goal state quickly. Thus, we expect in these cases that FocalSIPP will work well. In the experimental results below we confirmed these expectations.

Experimental Results

We evaluated the considered algorithms on a range of different grid maps including empty 64×64 map, 64×64 map containing 10 rectangular obstacles that resemble a warehouse (*Warehouse*), 64×64 map composed of square rooms connected by the passages (*Rooms*), 257×256 game map.[3] Grid connectivity varied from 8- to 32-connected. Each map was populated with 250 dynamic obstacles that move between random cells. 100 different pairs of start and goal locations for an agent were chosen on each map randomly. Two action models were considered – one that assumes only

---

[3]The Rooms map and the game map (den520) were taken from the movingai repository (Sturtevant 2012). Our code can be found at https://github.com/PathPlanning/SuboptimalSIPP.
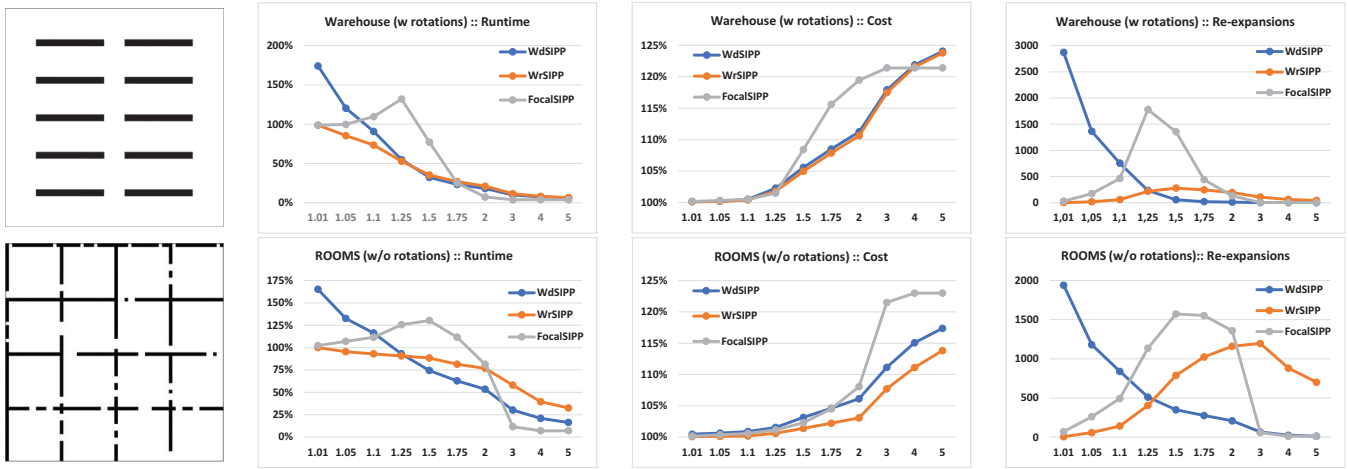
Figure 2: From left to right: map of the environment, runtime, solution cost, number of re-expansions. Top row: Warehouse w rotations. Bottom row: Rooms w/o rotations.

translations (denoted "w/o rotations") and one that also considers rotations (denoted "w rotations"), i.e., if moving from one cell to the other requires aligning the heading, the agent has to rotate and it takes time. Moving speed was 1 cell per 1 time unit, rotation speed was $\pi/2$ per time unit. We used the Euclidean distance, scaled properly with the agent's speed, as a heuristic. The secondary heuristic for FocalSIPP, $h_F(n)$, was set to be the shortest path from n to the goal ignoring all dynamic obstacles and edge costs, which we computed offline. The sub-optimality bound w varied from 1.01 to 5.

In each run we measured the algorithm's runtime and solution cost, relative to SIPP, and the number of re-expansions. While WSIPP_d prohibits re-expansions, it eventually generates two copies of the same state. If both of them got expanded, we count this as a re-expansion.

Fig. 2 shows the results of our experiments on two representative setups: *Warehouse w/o rotations* (32-connected) and *Rooms w rotations* (32-connected). We choose to present results for only these two setups due to space limitations. These specific maps represent two possible "types" of worlds. Warehouse is a relatively open environment populated with the isolated obstacles (similar to city maps, empty maps etc.), Rooms is a corridor-like environment with a large number of narrow passages (similar to mazes, indoor maps, etc.). The heuristic we used is relatively accurate for Warehouse but it is not accurate for Rooms, allowing us to show the impact of heuristic accuracy. Moreover, in the *w rotations* model, the heuristic is even less accurate.

In general, we observed similar trends across all the considered domains and action models. These trends, supported by the Fig. 2, are the following: (1) WSIPP_r is better for small values of w that are close to 1, (2) FocalSIPP is better or the same on higher w, and (3) WSIPP_d is better or the same for mid-range w. For mid-range sub-optimality bounds the results also show a notable "spike" in the number of re-expansions and runtime for FocalSIPP. We hypothesize that this occurs because the bound is large enough to cause finding suboptimal paths to generated nodes but not large
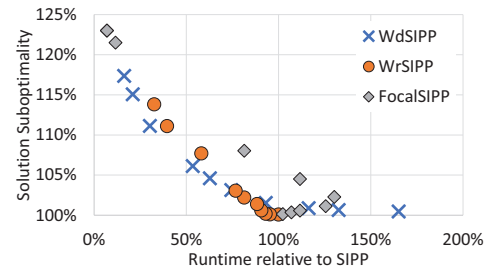
enough so that bounded-suboptimal solutions can be found without re-expanding these nodes.

Figure 3 shows the tradeoff between solution cost and runtime obtained by WSIPP_d, WSIPP_r, and FocalSIPP. Each data point represents the results of an algorithm with a specific w value, where the x value is the average runtime and the y value is the solution suboptimality. The results again show the same trends as above. FocalSIPP allows the fastest solution when solution cost is the worst. WSIPP_r is preferable if one wants very close to optimal solution, while WSIPP_d is more suitable for mid-range suboptimality.

Figure 3: Solution suboptimality vs. relative runtime, for Room maps w/o rotations.

Summary

We explored three bounded-suboptimal versions of SIPP and analyzed their pros and cons. Experimental evaluation on different settings show that the previously proposed bounded-suboptimal SIPP – WSIPP_d – is frequently outperformed by the other algorithms, e.g. by WSIPP_r, which runs WA^* but allows re-expanding. An appealing direction of future research is to explore more sophisticated techniques for bounded-suboptimal SIPP, as well as explore its applications in an anytime planning and multi-agent path finding.

Acknowledgments

References

Andreychuk, A.; Yakovlev, K.; Atzmon, D.; and Stern, R. 2019. Multi-agent pathfinding with continuous time. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 39–45.

Araki, B.; Strang, J.; Pohorecky, S.; Qiu, C.; Naegeli, T.; and Rus, D. 2017. Multi-robot path planning for a swarm of robots that can both fly and drive. In *IEEE International Conference on Robotics and Automation (ICRA)*, 5575–5582.

Cohen, L.; Uras, T.; Kumar, T. S.; and Koenig, S. 2019. Optimal and bounded-suboptimal multi-agent motion planning. In *Symposium on Combinatorial Search*.

Dechter, R., and Pearl, J. 1985. Generalized best-first search strategies and the optimality of a. *Journal of the ACM (JACM)* 32(3):505–536.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1(1):269–271.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics* 4(2):100–107.

Likhachev, M.; Gordon, G. J.; and Thrun, S. 2004. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in neural information processing systems (NIPS)*, 767–774.

Martelli, A. 1977. On the complexity of admissible search algorithms. *Artificial Intelligence* 8(1):1–13.

Narayanan, V.; Phillips, M.; and Likhachev, M. 2012. Anytime safe interval path planning for dynamic environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 4708–4715.

Pearl, J., and Kim, J. H. 1982. Studies in semi-admissible heuristics. *IEEE transactions on pattern analysis and machine intelligence* (4):392–399.

Phillips, M., and Likhachev, M. 2011. SIPP: Safe interval path planning for dynamic environments. In *Proceedings of The 2011 IEEE International Conference on Robotics and Automation (ICRA 2011)*, 5628–5635.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial intelligence* 1(3-4):193–204.

Sepetnitsky, V.; Felner, A.; and Stern, R. 2016. Repair policies for not reopening nodes in different search settings. In *Symposium on Combinatorial Search (SOCS)*, 81–88.

Sturtevant, N. R. 2012. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games* 4(2):144–148.

Wilt, C. M., and Ruml, W. 2014. Speedy versus greedy search. In *Symposium on Combinatorial Search (SoCS)*.