Improved Safe Real-time Heuristic Search

Bence Cserna and **Kevin C. Gall** and **Wheeler Ruml**
Department of Computer Science
University of New Hampshire, USA
Durham, NH, 03824, USA
`bence` at `cs.unh.edu`, `kcg245` at `gmail.com`, `ruml` at `cs.unh.edu`

Introduction

A fundamental concern in real-time planning is the presence of dead-ends in the state space, from which no goal is reachable. Providing real-time heuristic search algorithms that are complete in domains with dead-end states is a challenging problem. Recently, the SafeRTS algorithm was proposed for searching in such spaces (Cserna et al. 2018). SafeRTS exploits a user-provided predicate to identify safe states, from which a goal is likely reachable, and attempts to maintain a backup plan for reaching a safe state at all times.

This extended abstract summarizes our recent work in safe real-time search (Cserna, Gall, and Ruml 2019), in which we study the SafeRTS approach, identify certain properties of its behavior, and design an improved framework for safe real-time search. In addition, we prove that the new approach performs at least as well as SafeRTS and present experimental results showing that its promise is fulfilled in practice.

Cserna et al. (2018) introduce the notion of safety as a way of evaluating which states are less likely to lead to dead-ends, they call these *safe* nodes. SafeRTS divides the available time between searches optimizing the independent objectives of safety and finding the goal. This technique for proving safety involves alternating, potentially many times in a search iteration, between expanding the frontier and proving safety. We will argue below that this technique is inherently inefficient.

Improving Safe Real-time Search

We first briefly highlight two deficiencies of SafeRTS.

SafeRTS attempts to prove that some of the nodes in its local search space (LSS) are *safe* by initiating a speedy search (Burns, Ruml, and Do 2013) that prioritizes nodes with low safety distance. Safety distance is a lower bound on the number of nodes between a node and a safe node. There are 3 possible outcomes of a safety proof. It can be non-conclusive, it can prove that a node is safe or show that it is a dead-end. SafeRTS only utilizes the results when the node is proven to be safe. We propose to cache the information of which nodes are dead ends or predecessors of paths that exclusively lead to dead ends so that neither safety nor goal searches will re-expand them. Empirically, this optimization lead to $0.5 – 2.5\%$ savings on expansions in our experiments (Cserna, Gall, and Ruml 2019).

SafeRTS interleaves exploration and safety proofs during its planning phase. As a direct consequence, it attempts safety proofs on nodes that become internal to the LSS by the end of the search iteration. As shown in Cserna, Gall, and Ruml (2019), it would be equally or less difficult to achieve the same or better safety coverage by doing safety proofs after all the LSS expansions. SafeRTS has an anytime behavior but does not effectively utilize the real-time bound given.

A Real-time Framework for Safety

We now summarize the main contribution of Cserna, Gall, and Ruml (2019): a general scheme called Real-time Framework for Safety (RTFS). RTFS composes an algorithm from four elements: a parameter that determines the ratio between goal- and safety-oriented search, and three main functions: an *exploration function*, a *safe target selection function*, and a *proof allocation function*.

RTFS exploits the real-time bound of the problem to pre-allocate the time to spend on exploration and safety proofs by taking an *exploration ratio* as an input parameter. A higher value allows for more aggressive exploration, but decreases the likelihood of completing any safety proofs. The appropriate value should reflect the total available time per iteration and the difficulty of proving that a node is safe in a given domain.

The *exploration function* defines the way the algorithm uses the expansion budget to build the local search space. A trivial example of such a function is A*, but any exploration method that is capable of constructing a search tree could be used, such as wA* (Pohl 1970; Rivera, Baier, and Hernndez 2015), GBFS (Pearl 1984), Beam search (Russell and Norvig 2010), and Speedy (Burns, Ruml, and Do 2013). Using an exploration method that leads to a narrow and deep tree makes each safety proof more consequential as upon a successful proof every ancestor of the node can be marked safe.

Given a search tree, a target selection function, and an expansion budget, the *safety proof allocation strategy* distributes the given budget among the frontier nodes of the tree to prove their safety. It allocates resources based on the ordering provided by the target selection function. This function is highly non-trivial.

Given a search tree in which the safe and dead-end nodes are marked, the *target selection* function selects a node that the agent should commit towards. Cserna et al. (2018) described multiple examples for such target selection strategies.

RTFS first constructs the full local search space (LSS) using the *exploration function*, then carries out the safety proofs according to the *safety proof allocation function*, and commits to the target selected by the *target selection function*. Deferring the safety proofs after the construction of the LSS allows RTFS to make a more informed allocation of the available time, based on an evolved LSS. Thus, RTFS is likely to prove more promising nodes than SafeRTS and to avoid redundant work in safety proofs.

The learning and safety propagation is identical to those in SafeRTS with the addition of dead-end propagation that removes all nodes from the local search tree that were found to be unsafe or whose successors are all unsafe.

Empirical Evaluation

To ascertain the performance gain of RTFS, we create a configuration RTFS-0 with *target selection* and *safety proof allocation* functions that mimic SafeRTS. SafeRTS allocates at least 50% of its expansions towards the construction of the LSS, hence we set the *exploration ratio* of RTFS-0 to 0.5. Though both algorithms select the node on the open list with the lowest f value at the time the proof is attempted, RTFS-0 differs from SafeRTS as it only attempts safety proofs after expanding the LSS. As such, its proofs are only limited by the iteration bound. If a target node is proven to be an implicit dead-end, RTFS-0 removes it and all other discovered dead-end nodes from the graph, then attempts to prove the next best node on the open list.

In our experiments we include two additional oracles, A* and Safe-LSS-LRTA*, to provide reference points. A* is executed offline and its execution time is not included in its GAT. This serves as a lower bound on the GAT and an upper bound on the velocity. Safe-LSS-LRTA* is a version of LSS-LRTA* that has access to an ideal dead-end detector and thus only considers nodes that are safe. This offers the behavior of an agent-centered real-time search method that only has to focus on reaching the goal.

Real-time planning algorithms construct a solution iteratively and start to move the agent immediately after the first completed iteration. In the chain of decisions, the starting point of each decision is the result of all prior decisions. We argue that it is important for benchmark problems to balance the impact of each decision on the overall GAT. As an illustration of how to reduce the long term impact of actions in benchmark domains, we introduce a new benchmark domain called Airspace. One of the key principles behind Airspace is to avoid allowing any single decision to have an outsized effect on a planner's overall performance. Additionally, Airspace guarantees that the agent will not visit a state more than once, thus it eliminates scrubbing (Sturtevant and Bulitko 2016), focusing the benchmark on exploration and safety rather than on learning.

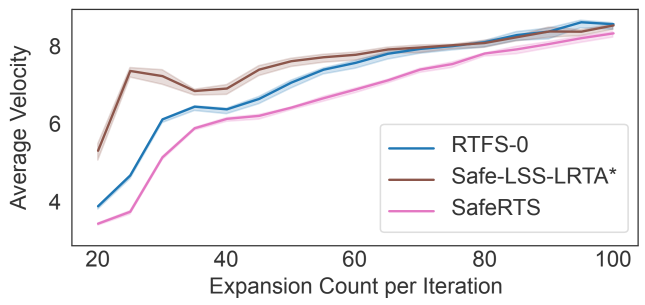

Figure 1: Average velocity on the Airspace domain.

We discovered multiple undesirable phenomena in the commonly used racetrack and traffic domains, thus this summary of our empirical evaluation focuses on the Airspace domain. In the Airspace domain the velocity of the agent directly corresponds to the GAT, thus it is used as the primary benchmark. Figure 1 shows the convergence of methods towards the oracle real-time search, and the average velocity shows a clear increasing trend as the time available per iteration increases. The maximum achievable velocity by A* was 13. RTFS-0 has faster average velocity and it closes the gap faster than SafeRTS.

Additional Results

For the detailed description of the RTFS method, the Airspace domain, the dead-end propagation, and for definitions, proofs, additional results and a discussion of benchmark domains please see Cserna, Gall, and Ruml (2019).

References

Burns, E.; Ruml, W.; and Do, M. B. 2013. Heuristic search when time matters. *Journal of Artificial Intelligence Research* 47:697–740.

Cserna, B.; Doyle, W. J.; Ramsdell, J. S.; and Ruml, W. 2018. Avoiding dead ends in real-time heuristic search. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*.

Cserna, B.; Gall, K. C.; and Ruml, W. 2019. Improved safe real-time heuristic search. arXiv.

Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence* 1:193–204.

Rivera, N.; Baier, J. A.; and Hernndez, C. 2015. Incorporating weights into real-time heuristic search. *Artificial Intelligence* 225:1 – 23.

Russell, S., and Norvig, P. 2010. *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition.

Sturtevant, N. R., and Bulitko, V. 2016. Scrubbing during learning in real-time heuristic search. *Journal of Artificial Intelligence Research* 57:307–343.