PARA*: Parallel Anytime Repairing A*

AAAI 2015 Submission X

Abstract

PARA* is an anytime parallel heuristic search algorithm based on ARA* and PA*SE, which are in turn based on A*.

Introduction

Bread-first and depth-first search are generalized by a class of frontier-based search algorithms, differing mainly in the means by which nodes are selected from the frontier for expansion. In the weighted A* algorithm, the choice combines a greedy goal-directed bias to reduce search time, with a breadth-first bias which guarantees suboptimality by a specified factor. Anytime Reparing A* (ARA*) is an anytime search algorithm, gradually reducing the goal-directed bias to improve solution cost as much as planning time allows. With the advent of multi-core processors, making use of parallelism have become a priority for algorithm designers. Parallel A* for Short Expansions (PA*SE) offers nearly linear speedup in the number of cores, provided the search is dominated by long expansion times.

In this paper, we present Parallel Anytime Repairing A* (PARA*), a simultaneous improvement over both ARA* and PA*SE. Like ARA*, it gradually reduces the weight which biases toward the goal, resulting in incrementally better solutions. Like PA*SE, it often offers approximately linear speedup. However, our design and analysis is tighter than PA*SE, even in the non-anytime setting. This enables several theoretical results, as well as moderate performance gains over PA*SE in certain settings.

Problem Formulation

We wish to find approximate single-pair shortest-paths. That is, given a directed graph with non-negative edge costs $c(s,s') \geq 0$, we must identify a path from s_{start} to s_{goal} whose cost is at most a specified factor $\epsilon \geq 1$ of the true distance $c^*(s_{start}, s_{goal})$. We assume the distances can be estimated by a **consistent heuristic** h, meaning $h(s,s') \leq c(s,s')$ and $h(s,s') \leq h(s,s'') + h(s'',s')$ for all s, s', s''. Of course, consistency implies **admissibility**, meaning $h(s,s') \leq c^*(s,s')$.

Algorithm Design

Algorithm 1 is a skeleton for the PARA* algorithm. It begins by clearing the four main data structures and expanding out all edges coming from the start node. When PARA* is run in an anytime fashion, it then enters a loop in which the data structures are "thawed" in preparation for each search iteration. On a first read-through, we advise ignoring the $FROZEN$ list as well as calls to $thaw()$ which are only needed for anytime operation.

Intuitively, $OPEN$ represents the frontier of candidate states for expansion, initially containing only the direct successors of s_{start}. Once selected for expansion, a state moves from $OPEN$ to $CLOSED$. BE represents the freshly $CLOSED$, i.e. the states which are still being expanded. Its cardinality $|BE|$ can never exceed the number of threads.

Algorithm 1 PARA*

```
$OPEN := BE := CLOSED := FROZEN := \emptyset$
$g(s_{start}) := 0$
expand($s_{start}$, 0)
repeat
    choose $\epsilon \in [1, \infty]$ and $w \in [0, \infty]$
    thaw()
    run search() on multiple threads in parallel
until path is good enough or planning time runs out
```

In Algorithm 2, we see that the main algorithm run by each thread resembles PARA*. $OPEN$ is represented by a balanced binary tree sorted by $f(s) = g(s) + wh(s, s_{goal})$ for some parameter $w \geq 0$. Usually we recommend setting $w = \epsilon$, but alternatives are discussed later.

Each thread begins by attempting to extract an element $s \in OPEN$ which is **safe** for expansion. A state is considered safe if its current distance label $g(s)$ is known to be a good enough approximation of the true distance; that is, $g(s) \leq \epsilon c^*(s_{start}, s)$. Each time a thread finds a safe s, it performs an expansion. The assignable variables $v(s)$ are never used, and exist only to aid the analysis. Intuitively, $v(s)$ is the distance label held by s during its most recent expansion. If $g(s) < v(s)$, s should be a candidate for future expansion.

There are many ways to define the auxiliary function $bound(s)$; we present one in Algorithm 4. Its most crucial property is that $bound(s) \leq \epsilon c^*(s_{start}, s)$, so that the condition $g(s) \leq bound(s)$ suffices to ensure s is safe for expansion. In addition, to ensure progress, $bound(s)$ must be defined such that at all times, the condition $g(s) \leq bound(s)$ holds for some $s \in OPEN$. Our implementation guarantees this for the state with minimal $g(s)$.

Algorithm 2 search()

```
while $g(s_{goal}) > bound(s_{goal})$ do
    among $s \in OPEN$ such that $g(s) \leq bound(s)$, remove one with the smallest $f(s)$ and LOCK $s$
    if such an $s$ does not exist then
        wait until $OPEN$ or $BE$ change
        continue
    end if
    insert $s$ into $CLOSED$
    insert $s$ into $BE$ with key $f(s)$
    $v_{expand} := g(s)$
    UNLOCK $s$
    expand($s$, $bound(s)$)
    $v(s) := v_{expand}$
    remove $s$ from $BE$
end while
```

Algorithm 3 is the state expansion procedure. It takes not only the state s, but also the lower bound on $\epsilon c^*(s_{start}, s)$ computed during the safety check. In the non-anytime special case, the first line assigning to g_p can be understood as initializing it to ∞ along with g and v. bp stores back pointers which can be followed from s back to s_{start} to yield a path of cost at most $g(s)$. $c_l \geq 0$ is the best known lower bound on the graph's edge costs.

Atomic locks are used for concurrency; for conceptual clarity, the mechanism presented here is considerably simpler than our implementation. We will not go into detail here, but it bears mentioning that every use of the main data structures is guarded by one global lock.

$g_p(s)$ is perhaps the least intuitive variable in PARA*. Its semantics is similar to $bound(s)$ but somewhat more intricate. $g_p(s)/\epsilon$ is a lower bound on the minimum-cost path from s_{start} to s whose predecessor to s has already been expanded. That is, $g_p(s) \leq \epsilon(c^*(s_{start}, s') + c(s', s))$ for all $s' \in CLOSED$. This inequality should also hold for every s' which was expanded (hence $CLOSED$) during prior anytime iterations.

Finally, Algorithm 4 lists the functions referenced in earlier listings, as well as the $thaw()$ procedure needed for the anytime extension. The following lemma lists some easily checked invariants.

Lemma 1. *At all times, the following invariants hold:*

- $OPEN \cap CLOSED = \emptyset$
- $BE \cup FROZEN \subset CLOSED$
- $s \in OPEN \cup BE \cup FROZEN \Leftrightarrow g(s) < v(s)$
- $s \notin OPEN \cup BE \cup FROZEN \Leftrightarrow g(s) = v(s)$
- $g(bp(s)) + c(bp(s), s) \leq g(s) \leq min_{s'}\{v(s') + c(s', s)\}$
- *Following* $bp(\cdot)$ *from* s *yields a path costing at most* $g(s)$

Algorithm 3 expand(s, g_{bound})

```
for all $s' \in successors(s)$ do
    LOCK $s'$
    if $s' \notin OPEN \cup CLOSED$ then
        if $s'$ has not been generated yet then
            $g(s') := v(s') := \infty$
        end if
        $g_p(s') := g(s') + 2(\epsilon - 1)c_l$
    end if
    $g_p(s') = \min(g_p(s'),\ g_{bound} + \epsilon c(s, s'))$
    if $g(s') > g(s) + c(s, s')$ then
        $g(s') = g(s) + c(s, s')$
        $bp(s') = s$
        if $s' \in CLOSED$ then
            insert $s'$ in $FROZEN$
        else
            insert/update $s'$ in $OPEN$ with key $f(s')$
        end if
    end if
    UNLOCK $s'$
end for
```

Algorithm 4 Auxiliary Functions

```
FUNCTION $successors(s)$
return $\{s' \mid c(s, s') < \infty\}$
FUNCTION $f(s)$
return $g(s) + wh(s, s_{goal})$
FUNCTION $g_{back}(s', s)$
if $s' = NULL$ then
    return $\infty$
else if $w \leq \epsilon$ then
    return $g(s) + f(s') - f(s) + (2\epsilon - w - 1)c_l$
else
    return $\frac{\epsilon}{w}(g(s) + f(s') - f(s)) + (\epsilon - 1)c_l$
end if
FUNCTION $bound(s)$
$g_{front} := g_p(s)$
$s'$ := first node in $OPEN \cup BE$
while $g_{back}(s', s) < g(s) \leq g_{front}$ do
    $g_{front} := \min(g_{front},\ g_p(s') + \epsilon h(s', s))$
    $s'$ := node following $s'$ in $OPEN \cup BE$
end while
return $\min(g_{front},\ g_{back}(s', s))$
PROCEDURE $thaw()$
$OPEN := OPEN \cup FROZEN$ with keys $f(s)$
$CLOSED := FROZEN := \emptyset$
for all $s \in OPEN$ do
    $g_p(s) := g(s) + (\epsilon - 1)\min(g(s),\ 2c_l)$
end for
```

- $s \in OPEN \cup CLOSED \Rightarrow g(s) + (\epsilon - 1)c_l \leq g_p(s) \leq \epsilon g(s)$
- $s \in OPEN \cup CLOSED$ *iff we had* $g(s) < v(s)$ *sometime during the current main() loop iteration*

Proof. Induction on time. □

Analysis

We begin by looking at the properties of $bound(s)$ as listed in Algorithm 4.

Lemma 2. *At all times, for all states* s *and* $s' \notin \{s_{start}, s\}$*:*

$$g_{back}(s', s) \leq g_p(s') + \epsilon h(s', s).$$

Proof. If $w \leq \epsilon$, then

$$\begin{aligned} & g(s) + f(s') - f(s) + (2\epsilon - w - 1)c_l \\ = \ & g(s') + w(h(s', s_{goal}) - h(s, s_{goal})) + (2\epsilon - w - 1)c_l \\ \leq \ & g(s') + wh(s', s) + (2\epsilon - w - 1)c_l \\ \leq \ & g(s') + \epsilon h(s', s) + (w - \epsilon)c_l + (2\epsilon - w - 1)c_l \\ = \ & g(s') + (\epsilon - 1)c_l + \epsilon h(s', s) \\ \leq \ & g_p(s') + \epsilon h(s', s) \end{aligned}$$

On the other hand, if $w > \epsilon$, then

$$\begin{aligned} & \frac{\epsilon}{w}(g(s) + f(s') - f(s)) + (\epsilon - 1)c_l \\ = \ & \frac{\epsilon}{w}(g(s') + w(h(s', s_{goal}) - h(s, s_{goal}))) + (\epsilon - 1)c_l \\ \leq \ & g(s') + \epsilon(h(s', s_{goal}) - h(s, s_{goal})) + (\epsilon - 1)c_l \\ \leq \ & g(s') + (\epsilon - 1)c_l + \epsilon h(s', s) \\ \leq \ & g_p(s') + \epsilon h(s', s) \end{aligned}$$

□

Lemma 3. *For all* $s \in OPEN \cup BE$*,* $bound(s) \leq \min_{s' \in OPEN \cup BE} g_p(s') + \epsilon h(s', s)$*. Furthermore,* $g(s) \leq bound(s)$ *iff* $g(s) \leq \min_{s' \in OPEN \cup BE} g_p(s') + \epsilon h(s', s)$*.*

Proof. By construction, $bound(s)$ is bounded above by $g_p(s') + \epsilon h(s', s)$ for $s' = s$ as well as for the other states s' which are checked in the loop. As for the remaining states $s' \in OPEN \cup BE$, the algorithm ensures that $bound(s) \leq g_{back}(s', s)$ for these by using a minimum representative. By Lemma 2, it follows that

$$bound(s) \leq \min_{s' \in OPEN \cup BE} g_p(s') + \epsilon h(s', s).$$

To prove the second claim, note that the loop in $bound(s)$ terminates under only two conditions. Either $g(s) > g_{front}$, in which case we have $g(s) > g_p(s') + \epsilon h(s', s) \geq bound(s)$ for the s' which began the final iteration; or $g(s) \leq g_{back}(s', s)$, in which case $g(s) \leq bound(s)$ iff $g(s) \leq g_{front}$ iff $g(s) \leq g_p(s') + \epsilon h(s', s)$ for all $s' \in OPEN \cup BE$. □

Theorem 1. *For all* $s \in OPEN \cup BE$*,* $bound(s) \leq \epsilon g^*(s)$*. Hence, for all* $s \in CLOSED$*,* $g(s) \leq v(s) \leq \epsilon g^*(s)$*.*

Proof. We proceed by induction on the order in which states are expanded.

Let $\pi = \langle s_0, s_1, \ldots, s_N \rangle$ be a minimum-cost path from $s_0 = s_{start}$ to $s_N = s \in OPEN \cup BE$. Choose the minimum i such that $s_i \in OPEN \cup BE$. If $i = 1$, then

$$g_p(s_i) \leq \epsilon g(s_i) = \epsilon g^*(s_i)$$

If $i \geq 2$, there are two cases to consider, depending on whether $s_{i-1} \in CLOSED$.

If so, then $expand(s_{i-1})$ has assigned to $g_p(s_i)$. Hence by the induction hypothesis,

$$\begin{aligned} g_p(s_i) & \leq v(s_{i-1}) + \epsilon c(s_{i-1}, s_i) \\ & \leq \epsilon g^*(s_{i-1}) + \epsilon c(s_{i-1}, s_i) \\ & = \epsilon g^*(s_i) \end{aligned}$$

On the other hand, suppose $s_{i-1} \notin CLOSED$. Choose the maximum $j < i$ such that $s_j \in CLOSED$, or $j = 0$ if there is no such j. Then $j \leq i - 2$ and, by the induction hypothesis, $g(s_j) \leq \epsilon g^*(s_j)$. Furthermore, $g(s_k) = v(s_k)$ for all $j < k < i$. Let $g_{old}(s_i)$ denote the value of $g(s_i)$ at the start of the current main() loop iteration. Then,

$$\begin{aligned} g_p(s_i) & \leq g_{old}(s_i) + 2(\epsilon - 1)c_l \\ & \leq v(s_j) + c^*(s_j, s_i) + 2(\epsilon - 1)c_l \\ & \leq \epsilon g^*(s_j) + c^*(s_j, s_i) + 2(\epsilon - 1)c_l \\ & = \epsilon(g^*(s_j) + c^*(s_j, s_i)) + (\epsilon - 1)(2c_l - c^*(s_j, s_i)) \\ & \leq \epsilon g^*(s_i) \end{aligned}$$

In all three cases, we found that

$$g_p(s_i) + \epsilon h(s_i, s) \leq \epsilon g^*(s_i) + \epsilon c^*(s_i, s) = \epsilon g^*(s).$$

Therefore, by Lemma 3,

$$bound(s) \leq \min_{s' \in OPEN \cup BE} g_p(s') + \epsilon h(s', s) \leq \epsilon g^*(s).$$

□

Corollary 1. *At the end of a main() loop iteration, the path obtained by following the back-pointers* $bp(\cdot)$ *from* s_{goal} *to* s_{start} *is* ϵ*-suboptimal.*

Proof. The termination condition of PARA* implies $g(s_{goal}) \leq bound(s_{goal})$. By construction, the path given by following back-pointers costs at most $g(s_{goal})$. The claim now follows from Theorem 1. □

Performance Guarantees - Blind PARA*

By deleting the while loop in bound(s), we arrive at a simplified version of the algorithm which we call Blind PARA*. g_p values are no longer used, so their computation can be ommitted. Blind PARA* can only expand states which would be proved safe in PARA* using zero iterations of the bound(s) loop. Thus, every performance guarantees that we prove for Blind PARA* also holds for PARA*.

Theorem 2. *If* $w \leq 1$*, the parallel depth of Blind PARA* is bounded above by*

$$\min\left(\frac{\epsilon g^*(s_{goal})}{(1 - w)c_l}, \frac{(\epsilon g^*(s_{goal}))^2}{(4\epsilon - 2w - 2)c_l^2}\right).$$

Proof. We prove the two bounds separately. For the first, note that if the lowest f-value is f_{min}, every state with f-value up to $f_{min} + (2\epsilon - w - 1)c_l$ can simultaneously be expanded. Since h is consistent, the successors' f-values is at least $f_{min} + (1 - w)c_l$. Therefore, the depth is at most

$$\frac{\epsilon g^*(s_{goal})}{(1-w)c_l}$$

For the other bound, write t for $2\epsilon - w - 1$. Notice that since f-values never decrease along paths, once the minimum f-value in $OPEN$ surpasses f_{min}, from then on all nodes with f-value up to $f_{min} + tc_l$ are always safe to expand. And during each iteration of the simultaneous expansions, the g-value of all such nodes increases by at least c_l. Since g cannot exceed f, this continues for at most $(f_{min} + tc_l)/c_l = f_{min}/c_l + t$ iterations, after which every node in $OPEN$ has f-value $\geq f_{min} + tc_l$. Continuing this process until f_{min} exceeds $\epsilon g^*(s_{goal})$, a bound on the total iteration count is (TODO: fix this analysis)

$$\begin{aligned}
& t + 2t + 3t + ... + \epsilon g^*(s_{goal})/c_l \\
\leq \quad & \epsilon g^*(s_{goal})/(tc_l)(2 + \epsilon g^*(s_{goal})/c_l + t)/2 \\
\leq \quad & (\epsilon g^*(s_{goal})/c_l)^2(tc_l/(\epsilon g^*(s_{goal})) + 1/(2t) + c_l/(2\epsilon g^*(s_{goal}))) \\
\leq \quad & (\epsilon g^*(s_{goal})/c_l)^2(t + 1/(2t) + 1/2)
\end{aligned}$$

□

Edgewise Supobtimality

Let $k(s)$ be the least number of edges used in a minimum-cost path to s and fix $\delta > 0$. If g_{front} and g_{back} are each increased by 2δ, then by similar arguments to the proofs earlier in the paper, we find that, upon expanding s, $g(s) \leq \epsilon g^*(s) + \delta k(s)$.

Here's an extension inspired by (Klein and Subramanian 1997): suppose the mean edge cost c_m along the optimal path is known to be much greater than the lower bound c_l. In such a case, the bound in Theorem 2 scales poorly. To remedy the situation, we "grow" the small edges, effectively running PARA* with $c'_l = c_l + \delta$ and $c'(s, s') = \max(c(s, s'), c'_l)$.

Theorem 3. *If the mean cost of the edges along the minimum-cost path to s is at least c_m, then upon expansion, $g(s) \leq \epsilon(1 + \delta/c_m)g^*(s)$. Therefore, to get the same optimality factor as ϵ, we can set $\delta = (\epsilon - 1)c_m$.*

Proof. We assumed $c_m \leq g^*(s)/k(s)$, so $k(s) \leq g^*(s)/c_m$. It follows from Lemma 1 that $g'(s) \leq \epsilon g'^*(s) \leq \epsilon(g^*(s) + \delta k(s)) \leq \epsilon(1 + \delta/c_m)g^*(s)$. □

Corollary 2. *If $w \leq 1$ and $c_m \leq g^*(s)/k(s)$, the parallel depth of Blind PARA* can be improved to*

$$\frac{\epsilon g^*(s_{goal})}{(1-w)(c_l + (\epsilon - 1)c_m)}.$$

If in addition $c_m \geq g^*(s)/(mk(s))$, the depth is at most

$$\frac{\epsilon mk(s)}{(1-w)(\epsilon - 1)}$$

In other words, if we know the mean edge cost up to a small constant factor, we can find approximately optimal paths in a depth which is within a small factor of the "omniscient" algorithm that expands only along the optimal path.

Experiments

Conclusion

References

Klein, P. N., and Subramanian, S. 1997. A randomized parallel algorithm for single-source shortest paths. *Journal of Algorithms* 25(2):205–220.