Enhanced Parallel A* for Slow Expansions

AAAI 2015 Submission 2327

Abstract

In order for heuristic searches to take advantage of modern CPU architectures, the algorithms must be parellelized. wPA*SE is a recent parallel variant of A*, which expands each state at most once and guarantees a solution cost not exceeding a specified factor of the optimal. wPA*SE can achieve a nearly linear speedup in the number of processor cores if expansions are sufficiently time-consuming to dominate the search runtime. Much of the overhead of wPA*SE is due to careful selection of states to expand, as required to meet the theoretical guarantees. In this work, we present extensions to wPA*SE that maintain speedups at faster expansion rates than wPA*SE allows. They reduce the overhead of selecting states for expansion by maintaining tighter bounds on the suboptimality of each state. On the theoretical side, we provide the same guarantees on completeness and solution quality as wPA*SE. Experimentally, we show comparable performance to wPA*SE when expansions are slow, and better performance as expansions become faster and the number of cores increases. Finally, we present an extension to the anytime setting.

Introduction

Breadth-first and depth-first search are generalized by a class of frontier-based search algorithms, differing mainly in the means by which states 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 bounded by a specified factor. With the advent of multi-core processors, making use of parallelism has become a priority for algorithm designers. Parallel A* for Slow Expansions (PA*SE) and its weighted generalization wPA*SE offer nearly linear speedup in the number of cores, provided the search time is dominated by time-consuming expansions.

In this paper, we present Enhanced PA*SE (ePA*SE). Its performance at least rivals wPA*SE in general, and surpasses it when expansions times are faster or a lot of processor cores are available. These improvements are achieved by tightening the analysis of wPA*SE. This enables several theoretical results, which we also present. We show how ePA*SE generalizes single-source shortest paths, providing performance bounds in the massively parallel limit.

Finally, we present Parallel Anytime Repairing A* (PARA*), a simultaneous improvement over both Anytime Repairing A* (ARA*) and wPA*SE. ARA* and PARA* are anytime search algorithms, gradually reducing the goal-directed bias to improve solution cost as much as planning time allows.

Related Work

Some early parallel versions of Dijkstras algorithm (Quinn 1986) and A* (Irani and Shih 1986) (Leifker and Kanal 1985) worked by simultaneously expanding some number of states in the frontier with the lowest g or f values. These methods do not bound suboptimality on the first expansion, so they must allow re-expansions. The approach of (Vidal, Bordeaux, and Hamadi 2010) performs well by increasing the number of threads if the goal is not found within some number of expansions, but they offer no guaranteed bounds on solution suboptimality.

There are several methods which split the frontier into multiple pieces for parallel processing. Parallel Retracting A* (PRA*) (Evett et al. 1995) uses a hash function to map each state to a processor. Parallel Structured Duplicate Detection (PSDD) (Zhou and Hansen 2007) uses a state abstraction function to group states into “nblocks (Zhou and Hansen 2007). Each processor takes an entire nblocks and can expand all its states without locking by provided that neighboring nblocks are not undergoing expansion. Parallel Best-NBlock-First (PBNF) (Burns et al. 2010) fuses the two latter approaches, running PRA* with a hashing function based on PSDD state abstraction. This avoids much of the thread locking in PRA*. This approach has also been extended to weighted (bounded sub-optimal) and anytime search. A more recent algorithm based on PRA* hasing is Hash Distributed A* (HDA*) (Kishimoto, Fukunaga, and Botea 2009), which uses asynchronous message passing to deliver hashed states between processors without blocking the sending thread.

Instead of parallelizing a single search algorithm, (Valenzano et al. 2010) run a variety of planning algorithms in parallel. Their solution quality is thus bounded by the worst so-

Copyright © 2014, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

lution quality bound from the set of algorithms run in parallel. Since the planners run independently of one another, the same state may be expanded multiple times. Likewise, the preceding approaches must allow states to be re-expanded, perhaps exponentially many times, in order to guarantee bounded suboptimality. Weighted Parallel A* for Slow Expansions (wPA*SE) (Phillips, Likhachev, and Koenig 2014) is a recent approach which, like weighted A*, expands each state at most once while guaranteeing a specified solution suboptimality factor. At the price of taking more time to select states for expansion, this approach reduces the total number of expansions, often improving planning times. In this paper, we extend wPA*SE to speed up the selection of states of expansion, while also increasing the number of states which are eligible for simultanous expansion.

Since these algorithms seldom provide bounds on runtime, anytime algorithms such as Anytime Reparing A* (Likhachev, Gordon, and Thrun 2003) and Anytime Window A* (Aine, Chakrabarti, and Kumar 2007) are an active area of research. We also contribute an anytime extension of wPA*SE along the lines of ARA*.

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

A parallel view of wA*

Many A* variants work by maintaining a set of upper bound estimates $g(s)$ of the optimal cost $g^*(s) = c^*(s_{start}, s)$ of reaching s from s_{start}. The estimates are constructive: every state s in the search tree has a back-pointers $bp(s)$, and these can be followed back from s to s_{start} to yield a path of costing at most $g(s)$.

In hopes of avoiding duplicate effort, the A* variants we consider are designed to expand each state at most once. Thus, before expanding s, it's important to verify that we already have a path from s_{start} to s satisfying the desired suboptimality bound. Formally, we say a state s is **safe for expansion** once we have deduced that $g(s) \leq \epsilon g^*(s)$.

wA* sorts the frontier by the numeric keys $f(s) = g(s) + wh(s)$ where $w = \epsilon$. Let $bound(s)$ be defined by $g(s) + f(s') - f(s)$ where s' is a state with minimum f-value among all frontier states. Then

$$\begin{aligned} bound(s) &= g(s) + \min_{s'}\{f(s')\} - f(s) \\ &= \min_{s'}\{g(s) + f(s') - f(s)\} \\ &= \min_{s'}\{g(s') + \epsilon\,(h(s', s_{goal}) - h(s, s_{goal}))\} \\ &\leq \min_{s'}\{g(s') + \epsilon h(s', s)\} \end{aligned}$$

where each $\min$ ranges over the frontier.

Provided s is in the frontier, it can be shown the latter expression is at most $\epsilon g^*(s)$. Therefore, s is considered safe for expansion if $g(s) \leq bound(s)$. Substituting in the definition of $bound(s)$, the latter inequality reduces to $0 \leq \min_{s'}\{f(s')\} - f(s)$. That is, s must have the minimum f-value of the frontier. This can be stated as a principle:

Expansion Rule 1 (wA* rule)**.** *A state $s \in OPEN$ is safe for expansion if its f-value is minimal among states in the frontier.*

Already, this grants a trivial degree of parallelism: if multiple states have the same minimal f-value, they can be expanded simultaneously. The main contribution of wPA*SE is to generalize this principle, allowing more states to be simultaneously safe for expansion.

Review of wPA*SE

Before describing how wPA*SE generalizes the wA* rule for safe expansion, let's outline the entire search algorithm in more detail. Our presentation of wPA*SE differs slightly from the original version in (Phillips, Likhachev, and Koenig 2014), but the algorithm we describe is essentially equivalent. Algorithm 1 is a skeleton for wPA*SE. It begins by clearing the data structures and expanding out all edges coming from the start state.

Intuitively, $OPEN$ represents the frontier of states which are candidates for expansion, initially consisting of the direct successors of s_{start}. Once a safe state is identified and selected for expansion, it's removed from $OPEN$ and inserted into the $CLOSED$ and BE (Being Expanded) lists. BE represents the freshly $CLOSED$ states: they are still in the process of being expanded, but are about to leave the frontier. Its cardinality $|BE|$ will never exceed the number of threads.

Algorithm 1 $main()$

```
OPEN := BE := CLOSED := FROZEN := ∅
g(s_start) := 0
expand(s_start)
search() on multiple threads in parallel
```

Every thread of wPA*SE runs Algorithm 2 in parallel. The search frontier is the union of $OPEN$ and BE, which are represented by balanced binary trees sorted by the key values $f(s) = g(s) + wh(s, s_{goal})$ for some parameter $w \geq 0$. Usually we recommend setting $w = \epsilon$, but possible motivations to select alternatives are discussed later.

Each thread begins by attempting to identify and extract an element $s \in OPEN$ which is safe for expansion (i.e. $g(s) \leq bound(s)$). Each time a thread finds a safe s, it performs an expansion as described in Algorithm 3. The search terminates once the goal is safe for expansion.

The assignable variables $v(s)$ and the $FROZEN$ list are never used, and exist in the pseudocode 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. $FROZEN$ consists of

$CLOSED$ states for which $g(s) < v(s)$, and hence would be candidates for expansion if not for the fact that s was already expanded. Thus, $OPEN \cup BE \cup FROZEN$ is precisely the set of states s for which $g(s) < v(s)$. All other states have $g(s) = v(s)$.

There are many ways to implement the auxiliary function $bound(s)$; as discussed in the previous section, we obtain a trivially parallelized version of wA* by using $bound(s) = g(s) + f(s') - f(s)$ for the minimizing $s' \in OPEN \cup BE$. wPA*SE uses the implementation in Algorithm 4. It satisfies $bound(s) \leq \epsilon g^*(s)$, rendering it usable as a rule.

Expansion Rule 2 (wPA*SE rule)**.** *A state $s \in OPEN$ is safe for expansion if its $g(s) \leq bound(s)$ using the implementation of bound listed in Algorithm 4.*

To sketch the intuition behind this implementation, we argue that in order for $s \in OPEN$ to be unsafe for expansion, there must be an optimal path from s_{start} to s passing through some safe $s' \in OPEN \cup BE$. Thus,

$$g(s) > \epsilon g^*(s) = \epsilon g^*(s') + \epsilon c^*(s', s) \geq g(s') + \epsilon h(s', s).$$

For s to be safe, then, it suffices that $g(s) \leq g(s') + \epsilon h(s', s)$ for all $s' \in OPEN \cup BE$. Assuming $w \leq \epsilon$, this inequality is guaranteed to hold whenever $f(s') \geq f(s)$. Hence, it suffices to check all s' for which $f(s') < f(s)$. These checks are expensive, their cost per state being at least proportional to the parallelism. The principal aim of our extensions is to substantially reduce the number of checks needed while increasing parallelism.

Before continuing, we briefly note that atomic locks are used for concurrency. For conceptual clarity, the mechanism presented here is considerably simpler than our C++ implementation. We will not discuss details here, but it bears mentioning that every use of the main data structures is guarded by the same global lock. As locks are often used to group consecutive operations into a single atomic operation, we may treat certain lines of code as simultaneous in the analysis. For example, a state is removed from $OPEN$ at the same time as it gets inserted into $CLOSED$ and BE.

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
  $expand(s)$
  $v(s) := v_{expand}$
  remove s from BE
end while

Algorithm 3 $expand(s)$

for all $s' \in successors(s)$ **do**
  LOCK s'
  if s' has not been generated yet **then**
    $g(s') := v(s') := \infty$
  end if
  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 $bound(s)$
$g_{front} := g(s)$
$s' :=$ first state in $OPEN \cup BE$
while $f(s') < f(s)$ **and** $g(s) \leq g_{front}$ **do**
  $g_{front} := \min(g_{front},\ g(s') + \epsilon h(s', s))$
  $s' :=$ state following s' in $OPEN \cup BE$
end while
return g_{front}

ePA*SE

Having formulated wPA*SE in a convenient new framework, we are now prepared to present our extensions.

We introduce the assignable variables $g_p(s)$. Their semantics are similar to $bound(s)$ but somewhat more intricate. While $bound(s)/\epsilon$ is a lower bound on the unrestricted distance $g^*(s)$, $g_p(s)/\epsilon$ is a lower bound on the cost from s_{start} to s, restricting ourselves to paths in which s is immediately preceded by an expanded state. That is, $g_p(s) \leq \epsilon(g^*(s') + c(s', s))$ for all $s' \in CLOSED$. To maintain this invariant, we initialize $g_p(s)$ to ∞ just as Algorithm 3 did for $g(s)$ and $v(s)$, and then make the following assignment immediately before the second **if** statement of $expand(s)$:

$$g_p(s') := \min(g_p(s'),\ b + \epsilon c(s, s'))$$

Here, b is the lower bound on $\epsilon g^*(s)$ computed by the $bound(s)$ call when s was extracted, or 0 if $s = s_{start}$. Note that at all times, $g(s) < \infty \Rightarrow g(s) < g_p(s)$.

The performance gains of ePA*SE come from changing the implementation of $bound(s)$ to the version shown in Algorithm 5. It now makes use of a constant $c_l \geq 0$, denoting the best known lower bound on the graph's edge costs. c_l can be 0 if we are agnostic about the possible costs, but ePA*SE can make use of larger bounds if available.

Expansion Rule 3 (ePA*SE rule)**.** *A state $s \in OPEN$ is safe for expansion if its $g(s) \leq bound(s)$ using the implementation of bound listed in Algorithm 5.*

Algorithm 5 $bound(s)$ enhanced for ePA*SE/PARA*

> **FUNCTION** $g_{back}(s', s)$
> **if** $s' = NULL$ **then**
>   **return** ∞
> **else if** $w \leq \epsilon$ **then**
>   **return** $g(s) + f(s') - f(s) + (2\epsilon - w - 1)c_l$
> **else**
>   **return** $\frac{\epsilon}{w}\left(g(s) + f(s') - f(s)\right) + (\epsilon - 1)c_l$
> **end if**
> **FUNCTION** $bound(s)$
> $g_{front} := g_p(s)$
> $s' :=$ first state in $OPEN \cup BE$
> **while** $g_{back}(s', s) < g(s) \leq g_{front}$ **do**
> $g_{front} := \min(g_{front},\ g_p(s') + \epsilon h(s', s))$
> $s' :=$ state following s' in $OPEN \cup BE$
> **end while**
> **return** $\min(g_{front},\ g_{back}(s', s))$

Recall that wPA*SE upper-bounds $\epsilon g^*(s)$ by the minimum of $g(s') + \epsilon h(s', s)$ for $s' \in OPEN \cup BE$. Hoever, not every element of $OPEN \cup BE$ needs to be considered separately. Indeed, suppose we choose an arbitrary subset $V \subseteq OPEN \cup BE$ over which we explicitly minimize $g(s') + \epsilon h(s', s)$. As we saw when discussing wA*, $g(s) + f(s') - f(s) \leq g(s') + \epsilon h(s', s)$. Thus, if we let $\bar{V} = (OPEN \cup BE) \setminus V$, a valid implementation of $bound(s)$ produces

$$\min\left(\min_{s' \in V}\{g(s') + \epsilon h(s', s)\},\ g(s) + \min_{s' \in \bar{V}}\{f(s')\} - f(s)\right).$$

wA* can be seen as the instance of this general definition with $V = \emptyset$. Since $OPEN$ and BE are sorted, a minimizing element of $\bar{V}$ is easily found. In PA*SE, V consists of the states s' such that $f(s') < f(s)$. The term corresponding to $\bar{V}$ is trivially minimized by s, yielding the value $g(s)$. Nonetheless, the term corresponding to V can take substantial effort to compute. Much of this effort can be removed by decreasing the size of V.

Before doing so, we note a few optimizations which will be justified in the formal analysis. Firstly, we can replace g by g_p in the term $g(s') + \epsilon h(s', s)$. Since $g_p(s) > g(s)$, this can only make the condition $g(s) \leq bound(s)$ more likely to hold, thus increasing parallelism. Likewise, if $c_l > 0$, the $\bar{V}$ term can be improved using Lemma 2.

It remains only to define V. Larger V tightens (i.e. increases) the value of $bound(s)$, but increases the computational expense. In ePA*SE, we begin with $V = \{s\}$ and iteratively add elements from $OPEN \cup BE$ in increasing order starting at the minimum f-value. We continue this until we have determined with certainty whether or not $\min_{s' \in V}\left(g_p(s') + \epsilon h(s', s)\right)$ (i.e. the value we would obtain if we let $V = OPEN \cup BE$) is greater than $g(s)$. That is, we do the minimum possible work while ensuring the result of the check $g(s) \leq bound(s)$ matches what would be obtained in the case where V is the whole frontier.

g_{front} is the bound computed from V, while g_{back} is computed from $\bar{V}$. The former decreases and the latter increases monotonically as states s' are added to V. The first termination condition $g_{back} \geq g(s)$ corresponds to a point after which the result of the comparison $g(s) \leq bound(s)$ cannot change by growing V, because $g_{back} \geq g(s)$ would continue to hold thereafter and Lemma 2 forbids any future elements moving from $\bar{V}$ to V from changing the result. This is guaranteed to hold for $s' = s$, ensuring that V never takes elements which would not have been considered by wPA*SE, aside from s. Similarly, the second termination condition $g(s) > g_{front}$ corresponds to a guarantee that $g(s) \leq bound(s)$ can never hold no matter how V is defined.

Note that the nested **if** statements in the pseudocode of g_{back} can be optimized away since we usually know in advance of the search whether $w \leq \epsilon$, and the $s' = NULL$ case can be handled by placing a null element with $f = \infty$ at the end of $OPEN$ and BE. Such an element will always trigger the $g_{back} \geq g(s)$ termination condition, ensuring that we never iterate past the end of $OPEN \cup BE$.

In summary, the ePA*SE implementation yeilds a sharper comparison than wPA*SE while doing explicit checks against no more, and often many fewer, states of the frontier. For instance, all states whose f-values are within $(2\epsilon - w - 1)c_l$ of the minimum can be expanded in parallel, as can any set of states which wPA*SE considers safe for expansion.

PARA*: Parallel Anytime Reparing A*

Finally, we note that by analogy with ARA*, ePA*SE can be made into an anytime algorithm, iteratively computing solutions with progressively smaller suboptimality bounds. We can modify Algorithm 1 so that, instead of creating threads to run $search()$ only once, the $search()$ threads are called in a loop which terminates only when the agent decides it's no longer worthwhile to spend additional planning time to improve the solution. Between iterations, the

$thaw()$ procedure in Algorithm 6 must be called to place the $FROZEN$ states back into the $OPEN$ list, since any state with $g(s) < v(s)$ can potentially be expanded to improve other g-values. Thus, the $FROZEN$ list now serves a concrete purpose.

Algorithm 6 $thaw()$

choose new $\epsilon \in [1, \infty]$ and $w \in [0, \infty]$
$OPEN := OPEN \cup FROZEN$ 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

For technical reasons which will become apparent in the proofs of correctness, the g_p values must be reinitialized when ϵ decreases. $thaw()$ already does this for the $OPEN$ list. When $expand(s)$ sees a state s' for the first time since the most recent call to $thaw()$, it performs the reset operation

$$g_p(s') := g(s') + 2(\epsilon - 1)c_l.$$

This generalizes and replaces the initialization step $g_p(s') := \infty$ from ePA*SE. Indeed, since all g-values are initialized to ∞, the g_p values are also initialized the first time around to $\infty + 2(\epsilon - 1)c_l = \infty$. Since g, v and g_p are always initialized before use, we will consider the unseen states as implicitly initialized for the purposes of analysis.

The following lemma lists some easily checked invariants of ePA*SE and PARA*.

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

- $OPEN \cap CLOSED = \emptyset$
- $BE \cup FROZEN \subseteq CLOSED$
- *If* $g(s) < \infty$*,* $bp(\cdot)$ *can be followed from* s *back to* s_{start} *to yield a path from* s_{start} *to* s *costing at most* $g(s)$
- *If* $s \neq s_{start}$*, then* $g(s) + (\epsilon - 1)c_l \leq g_p(s)$*,* $g_p \leq min_{s' \in CLOSED}\{v(s') + \epsilon c(s', s)\}$*, and* $g(bp(s)) + c(bp(s), s) \leq g(s) \leq min_{s'}\{v(s') + c(s', s)\}$
- $s \in OPEN \cup BE \cup FROZEN \Leftrightarrow g(s) < v(s)$
- $s \notin OPEN \cup BE \cup FROZEN \Leftrightarrow g(s) = v(s)$
- $s \in OPEN \cup CLOSED$ *iff we had* $g(s) < v(s)$ *sometime since the most recent call to* $thaw()$

Proof. Induction on time. □

The last line of the fourth point is a relaxation of $g(s) = \min_{s'}\{v(s') + c(s', s)\}$, an invariant often seen in sequential A* variants such as ARA*. Our relaxation allows more parallel variable assignments, such as modifying the g-value of a state which is undergoing expansion.

Theoretical Analysis

Correctness

We investigate ePA*SE/PARA* starting at the core: the $bound$ function of Algorithm 5.

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

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

where the last two inequalities follow by WLOG having $h(s, s') \geq c_l$ and Lemma 1.

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 every state* s*,*

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

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, g_{front} is bounded above by $g_p(s') + \epsilon h(s', s)$ for all $s' \in V$, where V consists of s and the states considered by the loop in $bound(s)$. Meanwhile, Lemma 2 ensures that g_{back} is bounded above by $g_p(s') + \epsilon h(s', s)$ for all $s' \in \bar{V}$, where $\bar{V} = (OPEN \cup BE) \setminus V$. Therefore,

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

To prove the second claim, note that the loop in $bound(s)$ terminates under one of two conditions.

If the loop terminates because $g(s) \leq g_{back}$, then by Lemma 2, $g(s) \leq g_{back} \leq \min_{s' \in \bar{V}}\{g_p(s') + \epsilon h(s', s)\}$. Since $g_{front} = \min_{s' \in V}\{g_p(s') + \epsilon h(s', s)\}$, it follows that $g(s) \leq bound(s)$ iff $g(s) \leq \min_{s' \in OPEN \cup BE}\{g_p(s') + \epsilon h(s', s)\}$.

On the other hand, if the loop terminates because $g(s) > g_{front}$, then the final assignment to g_{front} must correspond to a state s' for which

$$g(s) > g_p(s') + \epsilon h(s', s) = g_{front} \geq bound(s).$$

□

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 since $s_0 = s_{start}$ was already expanded,

$$g_p(s_1) \le \epsilon c(s_0, s_1) = \epsilon g^*(s_1).$$

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

If so, the induction hypothesis implies $v(s_{i-1}) \le \epsilon g^*(s_{i-1})$. Hence by Lemma 1,

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

On the other hand, suppose $s_{i-1} \notin CLOSED$, as might occur after a $thaw()$. Choose the maximum $j < i$ such that $s_j \in CLOSED$, or $j = 0$ if there is no such j. Then $j \le i - 2$ so $c^*(s_j, s_i) \ge 2c_l$ and, by the induction hypothesis, $v(s_j) \le \epsilon g^*(s_j)$.

Let $g_{old}(s_i)$ denote the value of $g(s_i)$ at the time of the most recent $thaw()$ (or ∞ is no $thaw()$ has occurred). For all $j < k < i$, s_k can never have been in $OPEN \cup CLOSED$ after the last $thaw()$; for if it had, then it would remain in $OPEN \cup CLOSED$, in contradiction to our construction of i and j. Thus, Lemma 1 implies $g(s_k) = v(s_k)$ held ever since the last $thaw()$, and so $g_{old}(s_i) \le v(s_j) + c^*(s_j, s_i)$. Now by the initialization of g_p,

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

In all three cases, we see that

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

Therefore, by Lemma 3,

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

$\square$

Corollary 1. *At the end of ePA*SE or a search round of PARA*, the path obtained by following the back-pointers $bp(\cdot)$ from s_{goal} to s_{start} is an ϵ-suboptimal solution.*

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

Completeness

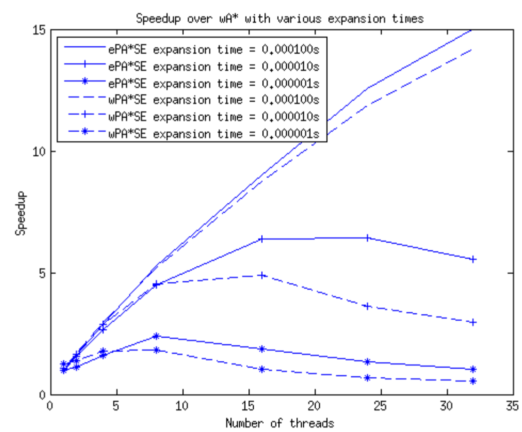
Having shown correctness of the algorithm at termination, it only remains to show that ePA*SE and every search round of PARA* indeed terminates. This is trivial on finite graphs, but we can also say something about a class of infinite graphs. The proof appears in the extended version of this paper.

Theorem 2. *ePA*SE and the search rounds of PARA* terminate in finite time, provided w, $g^*(s_{goal})$ and the out-degrees of states are all finite, and $c_l > 0$.*

Experiments

We tested wPA*SE and ePA*SE with parameters $w = \epsilon = 1.5$ on a 2D grid domain with 8-connected cells. We used the same 20 maps as (Phillips, Likhachev, and Koenig 2014) from a commonly used pathfinding benchmark (Sturtevant 2012). The searches were run on an Amazon EC2 machine with a 32-core Intel Xeon E5-2680v2. We varied the time per expansion between 10^{-6}, 10^{-5} and 10^{-4} seconds, and varied the number of threads between 1, 2, 4, 8, 16, 24 and 32.

As shown below, ePA*SE demonstrates moderate gains over wPA*SE as the number of cores increases, especially when expansion times are fairly fast. However, although ePA*SE demonstrates greater parallelism than wPA*SE, both algorithms falter when the number of threads is too high. This seems to be due to the fact that the data structure locks ensure only one thread can work on state extraction at a time. If the number of threads is high, some expansions finish before a new state is extracted, so the threads never become saturated, resulting in needless overhead. To achieve greater parallelism, it may become necessary to parallelize the extraction process so that multiple threads can access the frontier simultaneously.



Performance Analysis and Blind ePA*SE

Here we state some performance properties of ePA*SE in the massively parallel limit. See the extended version of this paper for proofs, as well as a further extension inspired by (Klein and Subramanian 1997) which offers similar performance bounds when, instead of the lower bound c_l, we have an estimate of the optimal solution's mean edge cost c_m.

By deleting the while loop in $bound(s)$, we arrive at Blind ePA*SE, a simplified version of ePA*SE in which g_p values are no longer used. Blind ePA*SE can only expand states which would complete zero iterations of the $bound(s)$ loop before being declared safe for expansion by ePA*SE. Thus, the performance guarantees shown here for Blind ePA*SE will also hold for regular ePA*SE.

Our theoretical measure of performance is a certain notion of **parallel depth**. Extracting a state from the frontier takes time logarithmic in the size of the queue; for our purposes, we consider the extraction and subsequent expansion to take one unit of time. Parallel depth is defined as the number of time units required if an unlimited number of threads are run in parallel. All states with f-value deviating by no more than $2\epsilon - w - 1$ from the minimum are safe for expansion, and they are considered to be simultaneously expanded in a single time unit. In such a setting, the following bound applies:

Theorem 3. *Suppose $w \leq 1$. Ignoring lower-order terms that lose significance as $g(s_{goal})/c_l \to \infty$, the parallel depth of Blind ePA*SE is bounded above by*

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

Thus, parallel depth is roughly linear in the goal distance when w is considerably smaller than 1, and becomes quadratic as $w \to 1$ provided $w \leq 1 < \epsilon$. Compare this against the exponential complexity of sequential search. This result suggests that using $w < \epsilon$ might become favorable as upcoming technological developments increase the number of available cores. The case $w = 0$ ignores all information about the goal, yielding a parallel anytime algorithm for the classic problem of single-source shortest paths.

Finally, the case $w > \epsilon$ merits future investigation as it allows a greedier bias in the frontier ordering, without loosening the suboptimality guarantee. While we de not recommend it in practice as it requires extensive checking for state extraction, it may be worthwhile if expansion times are particularly long. Furthermore, we gain usable information from the additional checks, as the result of $bound(s)$ becomes tighter. The $w = \infty$ extreme corresponds to sorting by h-value. Indeed, if we are willing to check against all of $OPEN \cup BE$, arbitrary orderings become permissible, but at the loss of the completeness promised by Theorem 2.

Conclusion

We have presented a framework which unifies wA* and wPA*SE, differentiating them primarily by expansion rule. Within this framework, we extended the algorithm to ePA*SE. We proved that ePA*SE maintains the completeness and guaranteed bounded suboptimality of wPA*SE, and has stronger performance bounds in the limit of massive parallelism. We showed experimental gains in performance, presented an anytime variant, and discussed the role of decoupling the weight parameter from the suboptimality factor.

Future work remains to experimentally investigate the effect of using $w \neq \epsilon$ under various settings. The bottleneck of ePA*SE appears to be the fact that only one thread at a time can access the data structures to extract states. While prior work includes techniques for splitting the frontier for parallel access, none so far have all the theoretical benefits we showed for ePA*SE.

References

Aine, S.; Chakrabarti, P.; and Kumar, R. 2007. Awa-a window constrained anytime heuristic search algorithm. In *International Joint Conference on Artificial Intelligence*, 2250–2255. Morgan Kaufmann Publishers Inc.

Burns, E.; Lemons, S.; Ruml, W.; and Zhou, R. 2010. Best-first heuristic search for multicore machines. *Journal of Artificial Intelligence Research* 39:689–743.

Evett, M.; Hendler, J.; Mahanti, A.; and Nau, D. 1995. PRA*: Massively parallel heuristic search. *Journal of Parallel and Distributed Computing* 25(2):133–143.

Irani, K., and Shih, Y. 1986. Parallel A* and AO* algorithms: An optimality criterion and performance evaluation. In *International Conference on Parallel Processing*.

Kishimoto, A.; Fukunaga, A.; and Botea, A. 2009. Scalable, parallel best-first search for optimal sequential planning. In *International Conference on Automated Planning and Scheduling*.

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

Leifker, D., and Kanal, L. 1985. A hybrid SSS*/alpha-beta algorithm for parallel search of game trees. In *International Joint Conference on Artificial Intelligence*, 1044–1046.

Likhachev, M.; Gordon, G. J.; and Thrun, S. 2003. Ara*: Anytime a* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems*, None.

Phillips, M.; Likhachev, M.; and Koenig, S. 2014. Pa*se: Parallel a* for slow expansions.

Quinn, M. 1986. *Designing efficient algorithms for parallel computers*. McGraw-Hill.

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

Valenzano, R.; Sturtevant, N.; Schaeffer, J.; and Buro, K. 2010. Simultaneously searching with multiple settings: An alternative to parameter tuning for suboptimal single-agent search algorithms. In *International Conference on Automated Planning and Scheduling*.

Vidal, V.; Bordeaux, L.; and Hamadi, Y. 2010. Adaptive k-parallel best-first search: A simple but efficient algorithm for multi-core domain-independent planning. In *Symposium on Combinatorial Search*.

Zhou, R., and Hansen, E. 2007. Parallel structured duplicate detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1217–1224.