

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 parallelized. 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) (Phillips, Likhachev, and Koenig 2014) 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 a known parallel algorithm for 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 ePA*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*) (Evetts 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”. Each processor takes an entire nblock and can expand all its states without locking, 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* hashing 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. By taking the quickest result, their solution qual-

ity is bounded by the worst quality bound among their set of search algorithms. 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 identification of states eligible for simultaneous expansion, while also increasing the number of eligible states.

Since runtime and available time for these algorithms tend to be hard to predict, anytime algorithms such as Anytime Repairing A* (ARA*) (Likhachev, Gordon, and Thrun 2003) and Anytime Window A* (Aine, Chakrabarti, and Kumar 2007) are an active area of research. By analogy with ARA*, we also contribute an anytime extension of ePA*SE.

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 whose cost is 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 $g(s) \leq \epsilon g^*(s)$.

Since we do not know $g^*(s)$ in practice, our general framework takes some easily computable function $bound(s)$ to act as a lower bound on $\epsilon g^*(s)$. wA* sorts the frontier by the numeric keys $f(s) = g(s) + \epsilon h(s, s_{goal})$, which can be used to compute $bound(s) = g(s) + f(s') - f(s)$ where s' is a state with minimum f -value among all frontier states.

Now,

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

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 was to generalize this principle, deducing more states as 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.

Each thread of wPA*SE runs an instance of Algorithm 2, which successively extracts a state from the frontier and then expands it. The search frontier is the union of *OPEN* and *BE*, which are represented by balanced binary search trees sorted by the key values $f(s) = g(s) + wh(s, s_{goal})$ for some weight parameter $w \geq 0$. Usually we recommend setting $w = \epsilon$, but possible motivations for alternatives are discussed later.

A state $s \in OPEN$ can only be extracted if it 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 valid functions that can take the place of $bound(s)$; correctness (i.e. soundness) only requires $bound(s) \leq \epsilon g^*(s)$, while termination (i.e. completeness) takes a little more care (see Theorem 2). As discussed in the previous section, we obtain a trivially parallelized version of wPA* by using $bound(s) = g(s) + f(s') - f(s)$ for the minimizing $s' \in OPEN \cup BE$. wPA*SE uses the substantially tighter $bound$ computed by Algorithm 4.

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

Algorithm 1 *main()*

```

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

```

Algorithm 2 *search()*

```

while g(sgoal) > bound(sgoal) do
  among s ∈ OPEN such that g(s) ≤ bound(s), re-
  move 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)
  vexpand := g(s)
  UNLOCK s
  expand(s)
  v(s) := vexpand
  remove s from BE
end while

```

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 it for all s' for which $f(s') < f(s)$. These checks are expensive, their cost per state being proportional to the number $|BE|$ of states undergoing simultaneous expansion. The principal aim of our extensions is to substantially reduce the number of checks needed while increasing parallelism.

Algorithm 3 *expand(s)*

```

for all s' ∈ successors(s) do
  LOCK s'
  if s' has not been generated yet then
    g(s') := v(s') := ∞
  end if
  if g(s') > g(s) + c(s, s') then
    g(s') = g(s) + c(s, s')
    bp(s') = s
    if s' ∈ CLOSED then
      insert s' in FROZEN
    else
      insert/update s' in OPEN with key f(s')
    end if
  end if
  UNLOCK s'
end for

```

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. Where locks are used to group consecutive operations into a single atomic operation, we may treat the corresponding 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 4 Auxiliary Functions

```

FUNCTION successors(s)
  return {s' | c(s, s') < ∞}
FUNCTION f(s)
  return g(s) + wh(s, sgoal)
FUNCTION bound(s)
  gfront := g(s)
  s' := first state in OPEN ∪ BE
  while f(s') < f(s) and g(s) ≤ gfront do
    gfront := min(gfront, g(s') + eh(s', s))
    s' := state following s' in OPEN ∪ BE
  end while
  return gfront

```

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 related to $g(s)$ and $bound(s)$ with subtle alterations. While $bound(s)/\epsilon$ is a lower bound on unrestricted distance $g^*(s)$ from s_{start} to s , $g_p(s)/\epsilon$ is a lower bound on the cost of s_{start} -to- s paths in which s is immediately preceded by a *CLOSED* state. That is, $g_p(s) \leq \epsilon(g^*(s') + c(s', s))$ for all $s' \in CLOSED$. To maintain this invariant, $expand(s)$ should initialize g_p to ∞ just as it does for g and v , and then make the following assignment immediately before the sec-

and **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 selected for expansion, or 0 if $s = s_{start}$. Note that at all times, $g(s) \leq g_p(s)$.

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 g_p as well as 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.

The introduction of g_p and c_l increases $bound(s)$, potentially increasing the number of nodes which are simultaneously safe for expansion. As an additional benefit, safe nodes are detected more quickly when the loop terminates early: for instance, if $w \leq \epsilon$ and $f(s) - f(s') \leq (2\epsilon - w - 1)c_l$, the loop will not consider s' . Provided s is safe for expansion, the loop in Algorithm 5 never takes more iterations than its wPA*SE equivalent. In a later section, our analysis of this early termination property leads to Theorem 3.

Expansion Rule 3 (ePA*SE rule). *A state $s \in OPEN$ is safe for expansion if $g(s) \leq bound(s)$ 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  $\infty$ 
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  $bound(s)$ 
 $g_{front} := g_p(s)$ 
 $s' := \text{first state in } OPEN \cup BE$ 
while  $g_{back}(s', s) < g(s) \leq g_{front}$  do
   $g_{front} := \min(g_{front}, g_p(s') + \epsilon h(s', s))$ 
   $s' := \text{state following } s' \text{ in } OPEN \cup BE$ 
end while
return  $\min(g_{front}, g_{back}(s', s))$ 

```

Recall that wPA*SE lower-bounds $\epsilon g^*(s)$ by the minimum of $g(s') + \epsilon h(s', s)$ among $s' \in OPEN \cup BE$. However, not every element of $OPEN \cup BE$ requires individual consideration. Indeed, suppose we choose an arbitrary subset $V \subseteq OPEN \cup BE$ over which we explicitly minimize $g(s') + \epsilon h(s', s)$. For the purposes of this sketch, assume $w \leq \epsilon$. As we saw when discussing wA*, $g(s) + f(s') - f(s) \leq g(s') + \epsilon h(s', s)$. Thus, if we let $V = (OPEN \cup BE) \setminus V$, 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 one instance of this general definition with $V = \emptyset$. Since $OPEN$ and BE are sorted, a minimizing element of \bar{V} is easily found. In wPA*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 $g_p(s) \geq g(s)$, 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 in accordance with 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 $g(s) \leq \min_{s' \in OPEN \cup BE} (g_p(s') + \epsilon h(s', s))$. 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 if V were 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 yields 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 Repairing 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 re-expanded to improve other g -values. Thus, the *FROZEN* list now serves a concrete algorithmic 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 proof 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.

To avoid excessive computation of $bound(s_{goal})$, the termination condition can be changed to $s_{goal} \in CLOSED$, provided that *thaw()* also adds s_{goal} to *OPEN*.

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 $g(s) + (\epsilon - 1)c_l \leq g_p(s)$, $g_p(s) \leq \min_{s' \in CLOSED} \{v(s') + \epsilon c(s', s)\}$, 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 $g(s) < v(s)$ some-time since the most recent call to *thaw()*

Proof. Induction on time. \square

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* beginning 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

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

\square

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

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

\square

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 $\pi = \langle s_0, s_1, \dots, s_N \rangle$ 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,

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

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

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

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

On the other hand, suppose $s_{i-1} \notin CLOSED$, 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 $j \leq i - 2$ so $c^*(s_j, s_i) \geq 2c_l$ and, by the induction hypothesis, $v(s_j) \leq \epsilon g^*(s_j)$.

Let $g_{old}(s_i)$ denote the value of $g(s_i)$ at the time of the most recent *thaw()* (or ∞ if 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 $g_{old}(s_i) \leq v(s_j) + c^*(s_j, s_i)$. Now by the initialization of g_p ,

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

In all three cases, we see 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 *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 $g(s_{goal}) \leq bound(s_{goal})$. By construction, the path given by following back-pointers costs at most $g(s_{goal})$. The claim now follows from Theorem 1. □

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 $c_l > 0$.

Proof. Let $\pi = \langle s_0, s_1, \dots, s_N \rangle$ be a minimum-cost path from $s_0 = s_{start}$ to $s_N = s_{goal}$. At any fixed time in the algorithm's operation, let i be maximal such that $g(s_i) < \infty$ and $g(s_i) \leq \epsilon g^*(s_i)$. We will prove that s_i is eventually expanded. Since expanding s_i ensures $g(s_{i+1}) \leq \epsilon g^*(s_i) + c(s_i, s_{i+1}) \leq \epsilon g^*(s_{i+1})$, it then follows by mathematical induction that $g(s_{goal})$ eventually attains a finite value not exceeding $\epsilon g^*(s_{goal})$.

First, suppose for contradiction that $v(s_i) \leq \epsilon g^*(s_i)$. Then by Lemma 1, $g(s_{i+1}) \leq v(s_i) + c(s_i, s_{i+1}) \leq \epsilon g^*(s_i) + c(s_i, s_{i+1}) \leq \epsilon g^*(s_{i+1})$, in contradiction to our construction of i . Thus, we must have $g(s_i) \leq \epsilon g^*(s_i) < v(s_i)$, which by Lemma 1 and Theorem 1 implies $s_i \in OPEN$. Let $\alpha = \min(w, \epsilon)$ and

$$C = \frac{w}{\alpha} (v(s_i) + c(s_i, s_{i+1})) + wh(s_{i+1}, s_N).$$

Once s_i is expanded, C is hereafter a constant. If we define $f_\alpha(s) = g(s) + \alpha h(s)$ and fix an $s \in OPEN \cup BE$ with minimum f_α -value, then s is safe for expansion. Now, $f(s) \leq \frac{w}{\alpha} f_\alpha(s) \leq \frac{w}{\alpha} f_\alpha(s_i) \leq C$. Consider the set of states with f -value at most C . Since s is safe for expansion, and states in $OPEN \cup BE$ are considered in order of increasing f , it must be the case that either some state in this set is currently being expanded, or the next state to be selected for expansion will be in this set. To show that s_i is eventually expanded, it now suffices to show that this set is finite.

Each edge costs at least c_l , so any state s which is separated from s_{start} by more than C/c_l edges must have $f(s) \geq g(s) > C$. Having bounded the depth at which the set of states s satisfying $f(s) \leq C$ may appear, finite out-degrees imply this set must have finite size.

To summarize, in finite time we obtain $f(s_{goal}) = g(s_{goal}) \leq \epsilon g^*(s_{goal})$. By a similar argument to the above, it follows that only finite time can take place before s_{goal} becomes safe for expansion, and the latter criterion is precisely the search termination condition. □

Asymptotic Time Complexity

To get a sense for the increased parallelism of *ePA*SE*, we analyze its worst-case time complexity in the limit where an unbounded supply of processors are available, the goal state is far from the start, and $w \leq \epsilon$. To simplify matters, we consider a “blind” version of *ePA*SE* which considers only the frontmost state $s \in OPEN$ and in which, if the while loop condition of Algorithm 5 succeeds, s is immediately deemed unsafe for expansion. This effectively removes the while loop.

If $|OPEN|$ is the frontier size, $|BE|$ is the number of states deemed simultaneously safe for expansion, and D is the maximum out-degree of a state, then having each of $|BE|$ threads simultaneously expand one node from the frontier now takes $O((|BE| + D) \log |OPEN|)$ time. Since any state deemed safe by Blind ePA*SE can be extracted just as quickly by ePA*SE, any performance guarantees we prove here will also apply to ePA*SE and PARA*.

Let one **parallel expansion time unit** be the time needed for all threads to simultaneously extract and expand one state, unless the blind safety check fails in which case they expand nothing. Algorithm 5 (and its blind variant) declares all states with f -value deviating by no more than $(2\epsilon - w - 1)c_l$ safe for expansion, so they are all expanded in a single time unit. In such a setting, the following bound applies:

Theorem 3. *If $w \leq 1$, ePA*SE completes in at most*

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

parallel expansion time units.

Proof. We prove the two bounds separately. For the first, note that if the minimum f -value is f_{min} , every state with f -value up to $f_{min} + (2\epsilon - w - 1)c_l$ can be expanded simultaneously. Since h is consistent, the successors' f -values are at least $f_{min} + (1 - w)c_l$. Hence, one time unit increases f_{min} by at least $(1 - w)c_l$. When s_{goal} is expanded, $f(s_{goal}) = g(s_{goal}) \leq \epsilon g^*(s_{goal})$. Since f_{min} is initially $f(s_{start}) = h(s_{start}) \geq 0$, it suffices to take

$$\frac{\epsilon g^*(s_{goal})}{(1-w)c_l} \text{ parallel expansion time units.}$$

For the other bound, let $t = 2\epsilon - w - 1$. Since f -values never decrease along paths, once the minimum f -value in $OPEN$ surpasses f_{min} , from then on all states with f -value up to $f_{min} + tc_l$ are always safe for expansion.

With each subsequent time step, states with f -value up to $f_{min} + tc_l$ have their g -values increased 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 state in $OPEN$ has f -value $\geq f_{min} + tc_l$. Continuing this process until f_{min} exceeds $\epsilon g^*(s_{goal})$, the total number of parallel expansion time units required is at most

$$\begin{aligned} & t + 2t + 3t + \dots + \lfloor u + 1 \rfloor t \\ & \leq \frac{t}{2}(u + 1)(u + 2) \\ & = \frac{t}{2}(u^2 + O(u)) \\ \text{where } u & = \frac{\epsilon g^*(s_{goal})}{c_l t}. \end{aligned}$$

□

Thus, if we ignore factors hidden in the time unit, the worst-case time complexity of ePA*SE is roughly linear in the goal distance when w is considerably smaller than 1, and

becomes quadratic as $w \rightarrow 1$. Compare this against the exponential complexity of sequential search. This result suggests that using $w < \epsilon$ might become favorable as upcoming technological developments raise the supply of processor cores. The case $w = 0$ completely ignores the heuristic-to-goal, yielding a parallel anytime algorithm for the classic single-source shortest paths problem.

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 do not recommend it in practice as it demands extensive checking during state extraction, it may be worthwhile if expansion times are especially long. Furthermore, we gain usable information from the additional checks, as the tightened result of $bound(s)$ propagates via g_p . The $w = \infty$ extreme corresponds to sorting by h -value. Indeed, if we are willing to check against all of $OPEN \cup BE$, then arbitrary orderings on $OPEN$ and BE become permissible, but at the price of the completeness promised by Theorem 2.

Edgewise Supoptimality (must rewrite or delete)

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 , $g(s) \leq \epsilon g^*(s) + \delta k(s)$.

The following is an extension inspired by (Klein and Subramanian 1997): suppose $c_l = 0$ or c_l is known to be very small compared to the mean edge cost c_m along the optimal path. In such a case, the bound in Theorem 3 scales poorly. To remedy the situation, we “grow” the small edges, effectively running ePA*SE with $c'_l = c_l + \delta$ and $c'(s, s') = \max(c(s, s'), c'_l)$.

Theorem 4. *If the mean cost of the edges along a 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 achieve a suboptimality factor of α , we can set $\epsilon = 1$ and $\delta = (\alpha - 1)c_m$.*

Proof. Let $k(s)$ be the number of edges along a minimum-cost path to s . 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.

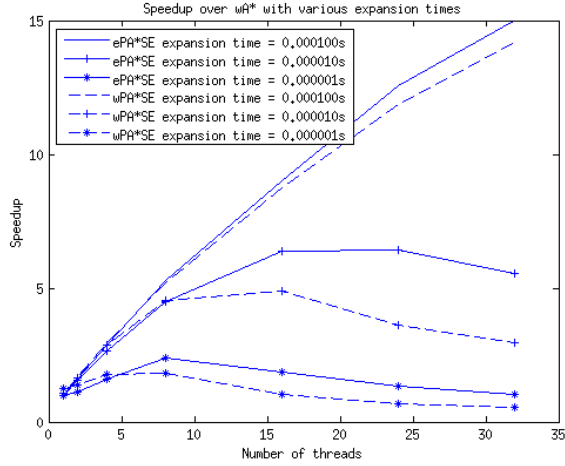


Figure 1: 2D grid speedup as the number of threads and expansion time are varied.

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.

In the first set of experiments, we varied the time per expansion between 10^{-6} , 10^{-5} and 10^{-4} seconds, and the number of threads between 1, 2, 4, 8, 16, 24 and 32. For each of these settings, Figure ?? displays the average speedup factor: the time taken by a sequential wA* search divided by the time taken by wPA*SE or ePA*SE. We observe 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.

In the second set of experiments, we varied the time per expansion between 10^{-5} and 10^{-4} seconds, and the weight parameters $w = \epsilon$ between 1.1, 1.5, 2, 3 and 4. Figure ?? displays the speedup factor for each setting in a similar format. We observe that ePA*SE achieves very high gains when the weight is low, gradually dissipating as the weight increases. In particular, when the weight is close to 1, the speedup over sequential search is approximately linear in the number of processors. ePA*SE is found to perform at least as well as the tested competitors in all cases, and considerably better in some classes of scenarios.

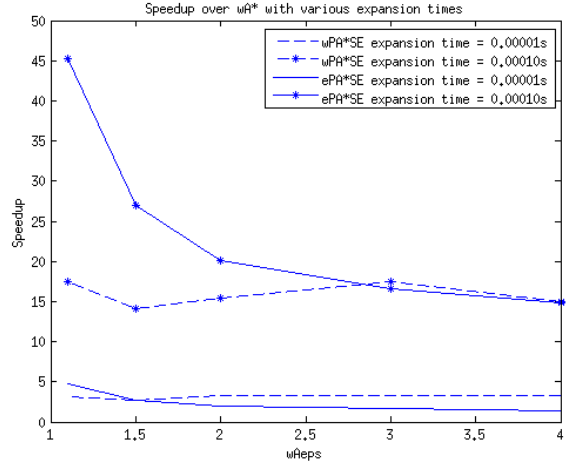


Figure 2: 2D grid speedup as the weight parameter and expansion time are varied.

Conclusion

We have presented a framework which unifies wA* and wPA*SE, differentiating them primarily by expansion rule. Within this framework, we created ePA*SE and proved it maintains the completeness and guaranteed bounded suboptimality of wPA*SE, with additional 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.
- Evetts, 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*.
- 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.