PA*SE: Parallel A* for Slow Expansions

Mike Phillips
mlphilli@andrew.cmu.edu
Carnegie Mellon University

Maxim Likhachev
maxim@cs.cmu.edu
Carnegie Mellon University

Sven Koenig
skoenig@usc.edu
University of Southern California

Abstract

Planners need to become faster as we seek to tackle increasingly complicated problems. Much of the recent improvements in computer speed is due to multi-core processors. For planners to take advantage of these types of architectures, we must adapt algorithms for parallel processing. There are a number of planning domains where state expansions are slow. One example is robot motion planning, where most of the time is devoted to collision checking. In this work, we present PA*SE, a novel, parallel version of A* (and weighted A*) which parallelizes state expansions by taking advantage of this property. While getting close to a linear speedup in the number of cores, we still preserve completeness and optimality of A* (bounded sub-optimality of weighted A*). PA*SE applies to any planning problem in which significant time is spent on generating successor states and computing transition costs. We present experimental results on a robot navigation domain (x,y,heading) which requires expensive 3D collision checking for the PR2 robot. We also provide an in-depth analysis of the algorithm's performance on a 2D navigation problem as we vary the number of cores (up to 32) as well as the time it takes to collision check successors during state expansions.

Introduction

Over the past few years, improvements to CPUs have mostly been in the form of additional cores instead of faster speed. In order to harness the power of these advancements in hardware, we must adapt algorithms for an ever more parallel computer. Heuristic search is one of the core tools in the field of artificial intelligence. In this work, we focus on the commonly used best-first search, A* (Hart, Nilsson, and Raphael 1968). The original A* algorithm is serial, as are most of its variants and extensions. Parallelizing A* is difficult because of the complex dependencies between pairs of states in the graph being searched.

A* guarantees completeness and when a solution is found, it is cost-minimal. A* also has a low worst case complexity, ensuring that any state will only be expanded at most once (provided the heuristics are consistent). However, at any given point in the search, A* only allows a state from the set of states with the minimal priority (f-value) to be expanded from the $OPEN$ list (this contains the states the search has discovered but hasn't expanded yet). In this work, we derive a new expansion rule which generalizes the original one and allows many more states to be expanded without harming guarantees on completeness or optimality. Consequently, these states can be expanded in parallel. Like A*, each state only needs to be expanded at most once.

The new expansion rule is defined in terms of pairwise state independence relationships. If a state s is independent of another state s', then the expansion of s' cannot lead to a shorter path to s and, therefore, it is safe to expand s before s'. If the independence relationship holds in both directions (s' is also independent of s), then we know that s and s' can be expanded in parallel. When choosing a state for expansion, there may be several independence checks that need to be run on a candidate state in order to confirm that it is safe to expand. Performing these checks before each expansion takes some time and, therefore, the proposed method, PA*SE (Parallel A* for Slow Expansions), is best suited for domains where state expansions are slow. That way, the overhead is dominated by the expansion time and a speedup can be achieved by parallelizing state expansions.

A good example of such a domain is robot motion planning with full-body collision checking. Expansions in this domain are slow because, when generating a successor, the planner must check that the body of the robot does not collide with any obstacles. This requires expensive collision checking of the mesh representing the body of the robot with the obstacles in the environment (typically represented as a 3D point cloud).

In this paper, we present PA*SE, a parallel version of A*, followed by a version with a relaxed independence check that allows for higher parallelization of expansions at the cost of optimality, and finally a parallel version of weighted A*. We provide theoretical analyses of the algorithms. Our experimental results on up to 32 processors show a near-linear speed up over A* and weighted A*. Finally, we show the benefits of our algorithm on (x,y,heading) motion planning with full-body collision checking.

Related Work

The first work toward parallel A* was likely a parallel version of Dijkstra's algorithm (Quinn 1986). This algorithm

always lets threads expand the cheapest states in $OPEN$ in parallel and may expand states before they have a minimal cost from the source state. This is resolved by allowing states to be re-expanded. As usual, the search terminates when there is nothing left to expand.

Around the same time, a similar approach was applied to A* (Irani and Shih 1986) (Leifker and Kanal 1985). The approach works like the parallel Dijkstra algorithm above but, as usual, the states are sorted differently in the $OPEN$ list and the termination condition is a bit trickier since expanding all states in the graph would defeat the efficiency of A*. This approach is the most similar to ours. We compare against this method in our experiments.

A similar approach starts with few threads and if the gaol is not found within a certain number of expansions, the search is restarted with more threads (Vidal, Bordeaux, and Hamadi 2010). While the approach worked well in practice, the algorithm provides no guarantees on solution quality.

PRA* (Parallel Retracting A*) provides a dramatically different approach by giving each processor its own $OPEN$ list (Evett et al. 1995). A hashing function is used to map every state in the graph to a processor. Upon generation, a state is hashed and then sent to its appropriate core. Every core expands states in its $OPEN$ list but locking of these lists must be used in order to pass a generated successor state to another processor.

Parallel Structured Duplicate Detection (PSDD) uses a state abstraction function to group states into "nblocks" (Zhou and Hansen 2007). Processors take entire nblocks (in parallel) and are allowed to expand all states in them without locking because the algorithm ensures that neighboring nblocks are not chosen to be expanded in parallel.

Parallel Best-NBlock-First (PBNF) combines the last two approaches by running PRA* but with a hashing function based on state abstraction from PSDD (Burns et al. 2010). This avoids much of the locking time experienced by PRA*. This approach has also been extended to weighted (bounded sub-optimal) and anytime search. We compare against this method in our experiments.

HDA* (Hash Distributed A*) is also based on the hashing idea from PRA* (Kishimoto, Fukunaga, and Botea 2009). However, an asynchronous message passing system allows the hashed states to be delivered to their receiving processor without causing the transmitting thread to block while waiting for a lock.

Yet another direction is to run multiple serial planning algorithms in parallel (Valenzano et al. 2010). The planners should either be diverse or at least use different parameters. The solution quality of this approach is bounded by the worst solution quality bound from the set of algorithms run in parallel. Clearly, in this approach states are expanded multiple times since all planners run completely independently of one another.

All of these approaches have to allow for states to be re-expanded (possibly many times) in order to guarantee optimality (or bounded sub-optimality in the case of weighted search). All of them could potentially expand an exponential number of states, especially in the case of weighted A* (it is possible that the work in (Valenzano et al. 2010) could have a bounded number of expansions, depending on the planners used). Our approach to parallel A* and weighted A* ensures that a state never has to be expanded more than once while maintaining the same guarantees on solution quality as their serial counterparts. This can significantly reduce the number of expansions and in our domains where expansions are time-consuming, this can greatly reduce planning times.

Algorithm

We start by presenting a parallel version of A* (PA*SE) and then present two simple variants that can improve planning speed. The variants are no longer optimal but still guarantee that the worst case solution cost is bounded. A key attribute of all of these algorithms is that no state will be expanded more than once. We show experimentally, how this improves planning speed.

Definitions and Assumptions

First, we list some definitions and notation that will help explain PA*SE.

- $G(V, E)$ is a finite graph where V is the set of vertices (or states) in the graph and E is the set of directed edges connecting pairs of vertices in V.
- s_{start} is the start state.
- s_{goal} is the goal state.
- $c(u, v)$ is the cost of the edge from vertex u to vertex v.
- The g-value, $g(s)$, is the cost of the cheapest path from s_{start} to s found by the algorithm so far.
- $g^*(s)$ is the minimum cost from s_{start} to s.
- $h(s)$ is a consistent heuristic. It is guaranteed not to overestimate the distance to the goal and satisfies the triangle inequality.
- $OPEN$ is an ordered list of states the search has generated but has not expanded yet.
- $CLOSED$ is the set of states the search has expanded (it is used to prevent re-expansion).

The objective of A* is to find a path (sequence of edges) that connects s_{start} to s_{goal}. A* does this by repeatedly expanding the state in $OPEN$ with the smallest f-value, defined as $f(s) = g(s) + h(s)$.

In addition to the above definitions, we assume that there also exists a heuristic $h(s, s')$ that can quickly provide an estimate of the cost between any pair of states. We require $h(s, s')$ to be forward-backward consistent, that is, $h(s, s'') \leq h(s, s') + h(s', s'') \forall s, s', s''$ and $h(s, s') \leq c^*(s, s') \forall s, s'$, where $c^*(s, s')$ is the cost of a shortest path from s to s' (Koenig and Likhachev 2005). This property typically holds if heuristics are consistent. For example, the heuristic $h(s, s') = h(s) - h(s')$ is guaranteed to be forward-backward consistent. As described later, our algorithm also makes use of a set BE, which contains the states currently being expanded in parallel.

Prior work: Parallel A* with Re-expansions

Parallel A* (PA*) (Irani and Shih 1986) operates as follows:

- Threads take turns removing the state with the smallest f-value from the $OPEN$ list by locking the $OPEN$ list.
- Once a thread has a state it expands it in parallel with the other threads.
- After a thread generates the successors of its state (and g-values), it puts them in the $OPEN$ list after locking it.

It's well known that A* (with consistent heuristics) never expands any state more than once because upon expansion, a state has an optimal g-value ($g(s) = g^*(s)$). Therefore, it never happens that, later in the search, a shorter path is found to a state that has already been expanded. However, parallel A* does not maintain this guarantee. States may not have optimal g-values upon expansion is because they may be expanded out of order. For example, suppose the $OPEN$ list currently looks like this: $(s_1, s_2, \dots)$. Now assume that two threads expand s_1 and s_2 in parallel but s_2 finishes first and generates s_3. Since s_2 was not at the front of the $OPEN$ list its g-value may not have been optimal. The expansion of s_1 might have directly lowered the g-value of s_2 (if s_1 has an edge to s_2) or it could generate successors that would have been placed in the $OPEN$ list ahead of s_2 and would have eventually led to a reduction of $g(s_2)$. Therefore, parallel A* will have to expand s_2 several times to guarantee that it finds a shortest path to it. Even worse, any time s_2 gets re-expanded because its g-value decreases, any of its successors, such as s_3, that were expanded previously, might have to be re-expanded again. This can lead to lots of states being expanded many times.

PA*SE

Our version of parallel A* obeys the invariant that, when a state is expanded, its g-value is optimal. Therefore, every state is expanded at most once, which provides a speedup over the previous approach (PA*) in our experiments. It does this by only choosing states for expansion that can not have their g-value lowered subsequently during the expansion of other states. This holds for any state in BE or the $OPEN$ list, as well as any other state that might later be inserted into $OPEN$. In this sense, PA*SE is a generalization of the A* expansion rule which only guarantees that the state with the minimum f-value cannot have its g-value lowered in the future. We show that there are often many states that have this property and therefore can be expanded in parallel.

A state s is "safe to expand" for A* if we can guarantee that its g-value is already optimal. This is equivalent to saying that there is no state currently being expanded (in BE), nor in the $OPEN$ list that can lead to a reduction of the g-value of s. We define s to be *independent* of state s' iff $g(s) - g(s') \leq h(s', s)$. We then define state s to be safe to expand if

$$g(s) - g(s') \leq h(s', s) \forall s' \in OPEN \cup BE. \quad (1)$$

The left-hand side of this inequality is an upper bound on the cost of a path from s' to s that could reduce the g-value of s. On the right-hand side of the inequality we have our forward-backward consistent heuristic. We know that a path from s' to s must cost at least $h(s', s)$. So if this lower bound on the cost of the path is at least as large as the maximum cost path that can lower the g-value of s, then we know that there is no path of sufficiently small cost from s' to s. Therefore, s is independent of s' and can safely be expanded before s'. If the opposite is also true (s' is independent of s), s and s' can be expanded in parallel.

With this rule, we can choose states to expand in parallel and guarantee that every state we expand has an optimal g-value and therefore no state will need to be expanded more than once. However, the check to see if s is safe to expand is quite expensive. To determine if we can expand s safely, we have to check whether s is independent of every state in $OPEN \cup BE$, which could be many states. It turns out that we only need to check that s is independent of states with f-values less that $f(s)$. This is similar to how A* can expand a state with the smallest f-value without worrying that it may be affected by states in $OPEN$ with larger f-values. As we prove in Theorem 2, if $f(s') \geq f(s)$, then s is independent of s' and can be expanded before s'. We can therefore update Equation 1 with a more efficient version.

$$g(s) - g(s') \leq h(s', s) \quad (2)$$

$$\forall s' \in \{a \in OPEN | f(a) < f(s)\} \cup BE.$$

This dramatically reduces the number of independence checks that need to be performed.

Algorithm 1 PA*SE Thread

```
1: LOCK
2: while s_goal does not satisfy Equation 2 do
3:     remove an s from OPEN that has the smallest f(s) among
       all states in OPEN that satisfy Equation 2
4:     if such an s does not exist then
5:         UNLOCK
6:         wait until OPEN or BE change
7:         LOCK
8:         continue
9:     end if
10:    insert s into BE
11:    insert s into CLOSED
12:    UNLOCK
13:    S := getSuccessors(s)
14:    LOCK
15:    for all s' ∈ S do
16:        if s' has not been been generated yet then
17:            f(s') := g(s') := ∞
18:        end if
19:        if s' ∉ CLOSED and g(s') > g(s) + c(s, s') then
20:            g(s') := g(s) + c(s, s')
21:            f(s') := g(s') + h(s')
22:            insert/update s' in(to) OPEN with f(s)
23:        end if
24:    end for
25:    remove s from BE
26: end while
27: UNLOCK
```

Algorithm 1 shows how we apply our independence rule to A* in order to make it parallel. Most of the algorithm

is identical to A*. On line 3, we remove a state from the $OPEN$ list for expansion that has the lowest possible f-value that is also safe to expand. Then after a state is selected, it is marked as "being expanded" by adding it to BE on line 10. Line 13 shows the expensive expansion step where the successors of s are generated and the costs of the edges to them are computed. Lines 15-24 show the typical A* relaxation step where, if a shorter path to a successor is found, its g-value and f-value are reduced accordingly and it is placed/updated in $OPEN$. Line 25 shows that, when an expansion is finished, we remove the state from the BE list. The search terminates when the goal can be safely expanded. In the case of no solution, the algorithm is also allowed to terminate if the $OPEN$ list is empty and all threads are idle (BE is empty).

Most of the lines in the algorithm occur under lock because they manipulate the $OPEN$ list. However, due to the expensive expansions on line 13, most of the time, the algorithm is lock-free, allowing for parallelism. The only other time it is lock-free is, if on line 3, an independent state is not found. This can only be caused by conflicts from other states that are simultaneously being expanded. In this case, the thread unlocks until one of the other threads puts its successors into the $OPEN$ list and releases the state it has expanded (lines 4-9).

In the algorithm, all threads take turns locking the $OPEN$ list and running independence checks. Since a thread must check a candidate for expansion against all states currently being expanded (the BE list), the more threads are currently expanding states, the longer it takes to run the independence checks for an additional thread. For an additional thread to provide a speed-up, it needs to be able to run the independence checks and remove a state from the $OPEN$ list before any of the threads currently expanding states finish. Otherwise, the thread that finished expanding could just as easily grab a new state as the additional thread. Therefore, the number of threads PA*SE can support is ultimately limited by how long an expansion takes. The more time-consuming expansions are, the more threads the algorithm can support. Robot motion planning is an ideal domain for PA*SE because expanding a state frequently requires expensive collision checking to generate successors.

It is important to note that, in the presented approach, a thread searching the $OPEN$ list for a state to expand may have a candidate fail an independence check. The algorithm then just moves on to the next state in the $OPEN$ list since it may still be possible for a state with a larger f-value to pass all of its independence checks. However, the farther down the list we look, the more checks need to be run (this is why the algorithm starts with the state with the smallest f-value). Verifying a state farther down the $OPEN$ list is more expensive (since more checks need to be run) and may also be less likely to succeed (because, with more checks, there are more chances to fail since it only has to fail once to not have independence). Therefore, the algorithm starts with the state with the smallest f-value. The next section discusses how to increase the likelihood of states early in the $OPEN$ list passing their checks.

Relaxing the Independence Rule

In this section, we introduce a modification that makes it much easier for states to pass the independence checks. The PA*SE algorithm remains the same. We simply modify the rule (used on Lines 2 and 3) for checking independence as follows.

$$g(s) - g(s') \leq \varepsilon_p h(s', s) \tag{3}$$

$$\forall s' \in \{a \in OPEN | f(a) < f(s)\} \cup BE.$$

All that has changed is that the lower bound on the cost from s' to s has been inflated by a chosen parameter $\varepsilon_p \geq 1.0$. Clearly, the larger the parameter, the easier it is to satisfy the independence rule (unless the heuristic is the zero heuristic). However, when the weight is larger than 1, states are no longer guaranteed to have an optimal g-value when they are expanded. While it may be higher than optimal, we can show that the g-value is bounded (Theorem 3). Specifically, upon expansion of a state s, $g(s) \leq \varepsilon_p g^*(s)$. It's important to note that this bound holds even though we only allow each state to be expanded at most once.

wPA*SE

Additionally, PA*SE works with the weighted A* algorithm (Pohl 1970). Weighted A* expands states according to the priority $f(s) = g(s) + wh(s)$. The heuristic is inflated by a weight $w \geq 1.0$ which causes the search to be more goal directed. In practice, weighted A* tends to find solutions significantly faster than A* even for weights close to 1. While the optimal solution is not guaranteed anymore, it has been shown that $g(s) \leq wg^*(s)$ even when only allowing each state to be expanded once (Likhachev, Gordon, and Thrun 2003). Weighted A* allows for an increase in speed at the expense of solution quality. This is especially useful for domains where finding optimal solutions requires too much time or memory, such as high dimensional motion planning problems.

It turns out that PA*SE and weighted A* can be combined trivially. We simply use our algorithm (with ε_p) but use the weighted A* computation of the f-value. The only constraint is that $\varepsilon_p \geq w$.[1] The bound on the solution quality remains ε_p as we prove in the next section.

Theoretical Analysis

We now analyze PA*SE and wPA*SE for arbitrary values of $\varepsilon_p \geq 1$ and $w \geq 1$.

Theorem 1 *When wPA*SE that performs independence checks against BE and the **entire** OPEN list chooses state s for expansion, then $g(s) \leq \lambda g^*(s)$, where $\lambda = max(\varepsilon_p, w)$.*

Proof sketch: Assume, for sake of contradiction, that $g(s) > \lambda g^*(s)$ directly before state s is expanded and, without loss of generality, that $g(s') \leq \lambda g^*(s')$ for all states s' selected for expansion before s. Consider any cost-minimal path $\tau(s_{start}, s)$ from s_{start} to s and let s_m be the state in

[1] Actually we can have $w > \varepsilon_p$ but then we have to run independence checks against all states in $OPEN$

$OPEN \cup BE$ closest to s_{start} on $\tau(s_{start}, s)$. s_m is no farther away from s_{start} on $\tau(s_{start}, s)$ than s since s is in the OPEN list. Thus, let $\tau(s_{start}, s_m)$ and $\tau(s_m, s)$ be the subpaths of $\tau(s_{start}, s)$ from s_{start} to s_m and from s_m to s, respectively.

If $s_m = s_{start}$ then it holds that $g(s_m) \leq \lambda g^*(s_m)$ since $g(s_{start}) = g^*(s_{start}) = 0$. Otherwise, let s_p be the predecessor of s_m on $\tau(s_{start}, s)$. s_p has been expanded since every state closer to s_{start} on $\tau(s_{start}, s)$ than s_m has been expanded (since every unexpanded state on $\tau(s_{start}, s)$ different from s_{start} is either in $OPEN \cup BE$ or has a state closer to s_{start} on $\tau(s_{start}, s)$ that is in $OPEN \cup BE$). Thus, $g(s_p) \leq \lambda g^*(s_p)$ according to our assumption about all states selected for expansion before s. Then,

$$\begin{aligned} g(s_m) &\leq g(s_p) + c(s_p, s_m) \\ &\leq \lambda g^*(s_p) + c(s_p, s_m) \\ &\leq \lambda g^*(s_m), \end{aligned}$$

where the first step holds due to the g-value update of s_m when s_p was expanded and the last step holds due to s_p being the predecessor of s_m on the cost-minimal path $\tau(s_{start}, s)$, which implies that $g^*(s_m) = g^*(s_p) + c(s_p, s_m)$. In either case, $g(s_m) \leq \lambda g^*(s_m)$.

We use $cost(\tau)$ to denote the cost of any path τ. Then,

$$\lambda cost(\tau(s_{start}, s_m)) = \lambda g^*(s_m) \geq g(s_m) \qquad (4)$$

and

$$\lambda cost(\tau(s_m, s)) \geq \varepsilon_p h(s_m, s), \qquad (5)$$

where the last inequality holds since $h(s_m, s) \leq cost(\tau(s_m, s))$ (because $h(s_m, s)$ satisfies forward-backward consistency and thus is admissible) and $\varepsilon_p \leq \lambda$ (because $\lambda = max(\varepsilon_p, w)$). Adding Inequalities 4 and 5 yields

$$\begin{aligned} \lambda cost(\tau(s_{start}, s)) &= \lambda cost(\tau(s_{start}, s_m)) + cost(\tau(s_m, s)) \\ &\geq g(s_m) + \varepsilon_p h(s_m, s). \end{aligned}$$

We assume that wPA*SE performs independence checks against BE and the **entire** $OPEN$ list and chose s to be expanded. Thus,

$$\begin{aligned} \varepsilon_p h(s_m, s) &\geq g(s) - g(s_m) \\ g(s_m) + \varepsilon_p h(s_m, s) &\geq g(s), \end{aligned}$$

according to our independence rule. Then,

$$\begin{aligned} \lambda g^*(s) &= \lambda cost(\tau(s_{start}, s)) \\ &\geq g(s_m) + \varepsilon_p h(s_m, s) \\ &\geq g(s). \end{aligned}$$

Contradiction. ■

Theorem 2 *Assume that $w \leq \varepsilon_p$ and consider any two states s and s' in the OPEN list. s is independent of s' if $f(s) \leq f(s')$.*

Proof sketch:

$$\begin{aligned} f(s) &\leq f(s') \\ g(s) + wh(s) &\leq g(s') + wh(s') \\ g(s) &\leq g(s') + w(h(s') - h(s)) \\ &\leq g(s') + wh(s', s,) \\ &\leq g(s') + \varepsilon_p h(s', s), \end{aligned}$$

where the second-to-last step holds due to the forward-backward consistency of the heuristic. Thus, state s is independent of state s' per definition. ■

Theorem 2 shows that wPA*SE does not need to perform independence checks against BE and the **entire** OPEN list provided that w and ε_p are chosen so that $w \leq \varepsilon_p$. Also, Theorem 2 implies the expansion rule of serial A* since it states that any state in the OPEN list with the smallest f-value is independent of all other states in the OPEN list.

Theorem 3 *Assume that $w \leq \varepsilon_p$. If wPA*SE chooses s_{goal} for expansion, then $g(s_{goal}) \leq \varepsilon_p g^*(s_{goal})$.*

Proof sketch: The theorem directly follows from Theorem 1 for a version of wPA*SE that performs independence checks against BE and he **entire** OPEN list since $\lambda = max(\varepsilon_p, w) = \varepsilon_p$. Theorem 2 shows that the behavior of wPA*SE does not change if it performs independence checks against BE and only those states in the OPEN list whose f-values are smaller than the f-value of the state considered for expansion, which reduces the computation time substantially. ■

We still need to prove that wPA*SE finds a path from s_{start} to s_{goal} whose cost is no greater than $\varepsilon_p g^*(s_{goal})$ if a path from s_{start} to s_{goal} exists and otherwise terminates with failure. This proof makes use of Theorem 3 and is otherwise similar to the equivalent proof of serial weighted A*.

Experimental Results

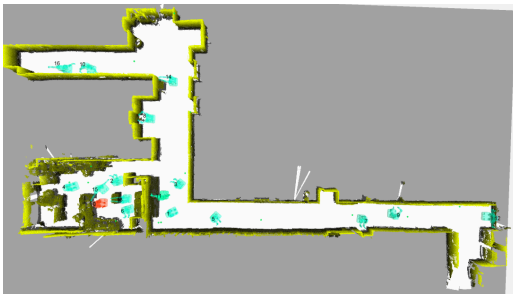
For our experiments, we first show how PA*SE successfully accelerates planning in a robot motion planning domain that requires expensive collision checking. Then, on a grid search domain, we provide a thorough analysis of the algorithm and how it is affected by various parameters.

An important implementation detail is that, while A* typically implements the $OPEN$ list as a heap, in PA*SE we must be able to access all states in order by f-value, not just the minimum one. Therefore, maintaining OPEN as a data structure that is fully sorted (e.g. bucket list or balanced binary tree) may be advantageous. In our experiments, we used a red-black tree. While this may not be as efficient as a heap in practice, the runtime of operations performed on the $OPEN$ list are negligible compared to how expensive the expansions are.

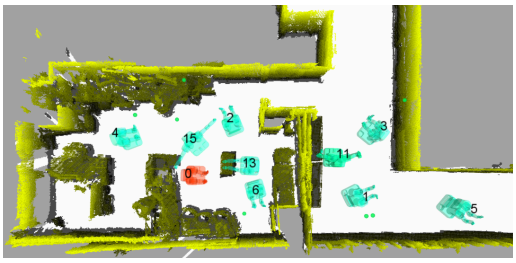
Robot Navigation

Our domain is robot navigation (x,y,heading) with 3D collision checking. The heading dimension is important when moving a robot that is not shaped like a cylinder (which could just be represented as a point robot in (x,y). The PR2 robot has a square base and two large arms. When carrying objects, the arms may need to pass over a counter or stool. This kind of motion would be illegal using 2D collision checking which projects the robot down to a footprint and the obstacles to the ground plane. Therefore, in cluttered environments, it is crucial to reason about the true 3D model of the robot's body.

In general, robot motion planning is an ideal domain for PA*SE. These types of problems have time-consuming expansions because, in order to generate a successor for a state,



(a) The 3D environment and the states used for starts and goals



(b) A close-up of some of the start and goal states

Figure 1: The domain used for our robot experiments. There are 17 numbered poses shown. Our 16 trials came from planning between consecutively numbered poses.

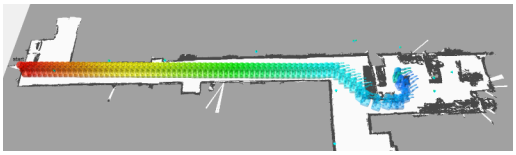


Figure 2: A visualization of a path found by our planner

the edge connecting them must be collision checked. Collision checks usually have to be performed at a high resolution making the expansions even more expensive.

We ran 16 planning scenarios on a 3D map built with the PR2's depth sensors. Figure 1 shows the 3D map and the poses used for the start and goal states. Figure 2 shows a path found by our planner. For our heuristic we ran a 2D Dijkstra search from the goal position. In order to be admissible, it checks a point robot against obstacle cells in a grid. It only marks a cell in the 2D map as an obstacle if the entire z-column is blocked in the 3D map (such as by a wall). It also assumes that the robot incurs no cost for rotation (since it is not a dimension in this search). This heuristic is particularly informative because it takes some obstacle information into account. However, it cannot provide pairwise heuristic information which we need for the heuristic $h(s, s')$. For this we use the maximum of the Euclidean distance between the positions and the angular distance between the headings. Since the units of these are different, we convert both of them to the time it takes the robot to execute that motion

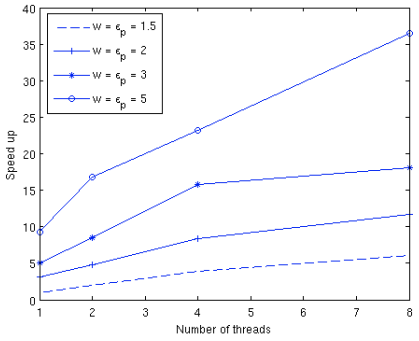


Figure 3: Speedup of wPA*SE over weighted A* (with $w = 1.5$, the point in the lower left) using various weights and threads

(which is the same as what our cost function is). We ran our experiments on an Intel Core i7 with 8 cores.

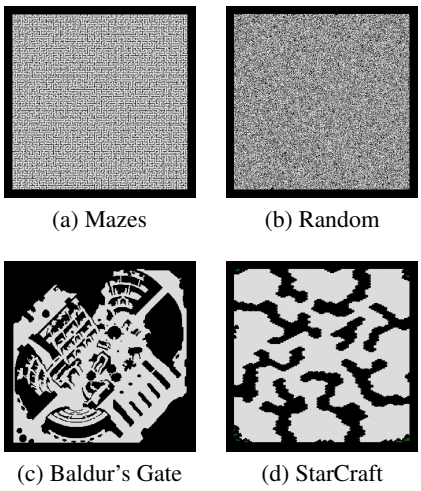
Figure 3 shows how performance of PA*SE improves over serial weighted A*. The points in this plot are computed by taking the time weighted A* with $w = 1.5$ took to find a solution over the time wPA*SE took to find a solution. Values larger than 1.0 represent a speedup. The effect of increasing the number of threads on the speedup is captured by the slope of each individual curve, while the effect of increasing the weights is captured by the upward shift of the entire curve. Increasing the number of threads or the weights improves the runtime and the highest speedup is achieved by increasing both.

Grid Search Domain

In these experiments, we ran our planner on a much larger set of trials while varying important parameters such as ε_p, w, the number of threads, and the time it takes to expand a state. We investigate how each of these parameters affects the performance of our planner. We ran these trials on a simpler 2D grid domain with 8-connected cells. We selected 20 maps from a commonly used pathfinding benchmark database (Sturtevant 2012). Four kinds of maps each represent a quarter of the set (Figure 4). We ran one trial for each map (we chose one of the most difficult ones from the benchmark set). Unlike the robot motion planning experiment, here we artificially set the amount of time it takes to expand a state by having the thread do some unimportant computations for the specified amount of time.

We chose Euclidean distance as both our heuristic $h(s)$ and heuristic $h(s, s')$. This heuristic is fast to compute and never returns a zero estimate for any pair of distinct states. We ran our experiments on an Amazon EC2 computer with an Intel Xeon E5-2680v2 (32 cores).

Figure 5 shows the result of our first experiment. Here we set $\varepsilon_p = w = 1.0$ and vary the number of threads to see how performance improves over serial A*. The points in this plot are computed by taking the time A* took to find a solution over the time PA*SE took to find a solution. So, values larger than 1.0 represent a speedup. The plot shows a near linear speedup in the number of threads. 32 threads (the



(a) Mazes (b) Random

(c) Baldur's Gate (d) StarCraft

Figure 4: The four types of maps used in our experiments

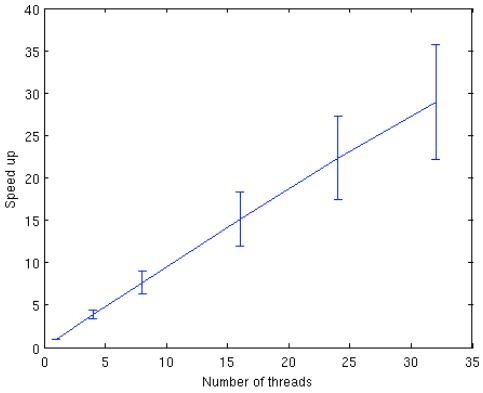


Figure 5: Speedup of PA*SE over serial A* for different numbers of threads

largest we tried) produce on average a 29 times speedup. In this experiment, we had each expansion take 0.0005s (which is common in robotics).

Figure 6 shows how reducing the expansion time affects how many threads PA*SE can support. This experiment also has $\varepsilon_p = w = 1.0$. The speedups are shown with respect to serial A*. When the expansion time drops to 0.00005s, it is faster to run 16 threads than 32. When the expansion time drops to 0.00001s, 4 threads are almost as fast as 16. It seems that the speedup provided by PA*SE is close to linear as long as the time per expansion is large enough to support that number of threads. Having more threads, can actually slow down the algorithm

Figure 7 shows the results from an experiment showing the effect of the two parameters w and ϵ_p. Each curve's speedup is computed with respect to running with the same number of threads on the same map, but with both weights set to 1.0. We can see that, in general, w produces speedups. Although the extent depends on the map type (on the maze

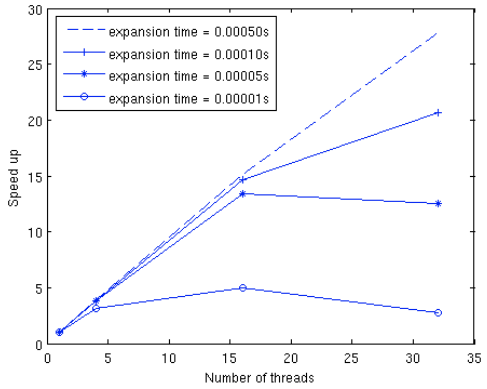


Figure 6: Speedup over serial A* for different expansion times

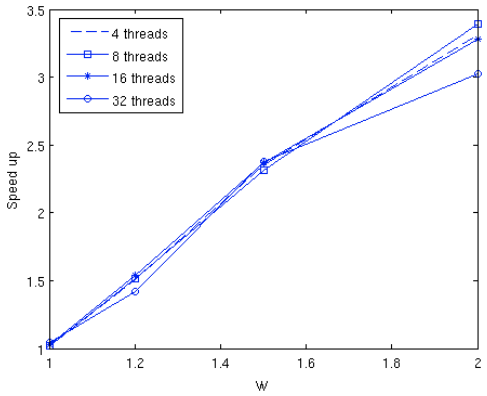


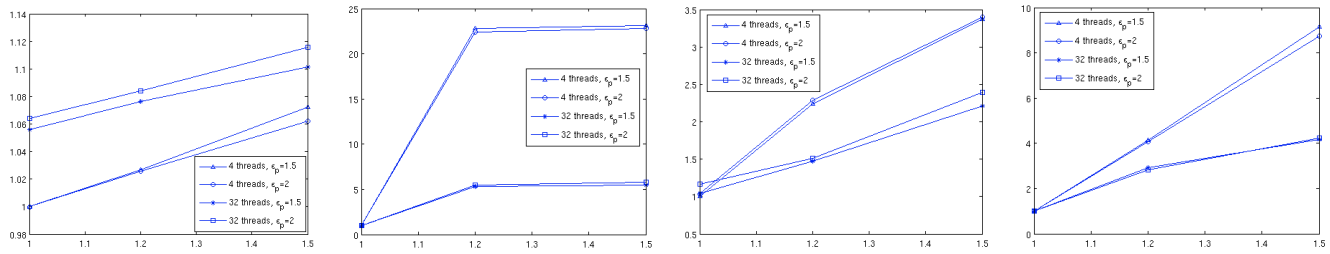
Figure 8: Speedup of wPA*SE over weighted PA* (w is the weight used for both approaches)

maps we have very little improvement due to the poor heuristic). On the other hand, we see that ϵ_p has relatively little impact, probably because the threads are already busy so the relaxed independence checks do not help much.

Comparisons

We compare PA*SE to parallel A* that allows re-expansions (Irani and Shih 1986). This approach was not extended to weighted A* in their paper, but the modifications are trivial (just change the priority function to inflate the heuristic term in the f-value). In this experiment, we set expansion time to 0.0005s We set $\varepsilon_p = w$ to maintain the same theoretical bound that the other approach has.

Figure 8 shows our speedups against weighted PA*, which allows re-expansions. Once again, values larger than 1.0 indicate a speedup. The data seems to suggest that wPA*SE doesn't perform better than this approach as the number of threads increase However, we get a larger speedup as w increases. Presumably, the reason is that as the sub-optimality bound gets larger, states are more likely to be expanded multiple times in the other approach. Once again, being able to increase w is crucial in high-dimensional problems such as robot motion planning in order to achieve faster



(a) Mazes (b) Random (c) From Baldur's Gate (d) From StarCraft

Figure 7: Speedup produced for different weights. The y-axes are speedups and x-axes are weight w. Each curve's speedup is computed with respect to running with the same number of threads and on the same map, but with both weights set to 1.0

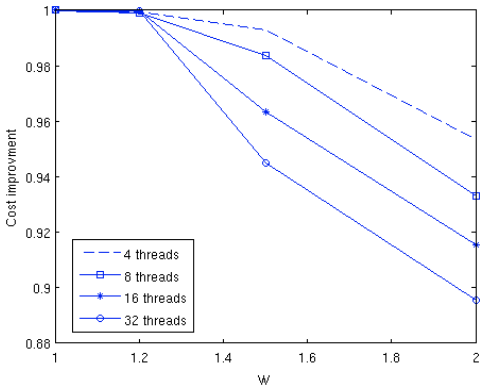


Figure 9: Cost improvement over weighted PA* (w is the weight used for both approaches)

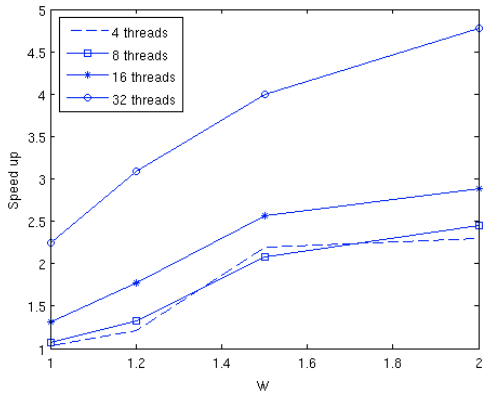


Figure 10: Speedup of PA*SE over PBNF (w is the weight used for both approaches)

planning times. Increasing this weight seems to help us more than the other approach. It seems that this relationship is relatively invariant to the number of threads. The expansion "speedups" have almost the exact same ratios to the time speed-ups. In fact, all expansion "speedup" ratios differ by more than 0.1 of their corresponding speed-up ratio.

While wPA*SE has the same theoretical bound on the solution quality as weighted PA*, in practice the solution costs need not be the same. Intuitively, since PA* re-expands states when cheaper paths to them are found while wPA*SE does not do that, it should result in a better solution quality than wPA*SE. In Figure 9, we see how our path quality compares. The data points are the average cost improvement of wPA*SE compared to allowing re-expansions (a value of 2 means that our path is 2 times cheaper). We can see that, for $w = 1.0$, the ratio is 1 since both approaches are optimal and, therefore, always result in the same solution cost. However, as the weight increases, their cost gets cheaper compared to ours. We can see that, on average, their costs are no better than 90% of ours. In return wPA*SE can run over 3 times faster by not allowing re-expansions.

We also compared against PBNF (Parallel Best nBlock First) (Burns et al. 2010). This is a state of the art algorithm for parallelizing A* and weighted A*. For the optimal searches, we used Safe PBNF and for weighted search we used wPBNF. The algorithm has a few parameters in addition to the sub-optimality bound. To tune these, we started with the values suggested by the authors for 2D grid search and then did a gradient descent search to minimize their planning times. It is important to note that we tuned the parameters on an 8-core desktop before running it on the 32-core Amazon EC2 computer. It is possible that there is a better set of parameters when using more than 8 threads.

Figure 10 shows our speedups over PBNF. Varying the number of threads does not affect our speedup for 8 or fewer threads. However, we do see a speedup for 16 and 32 threads (though it is possible that PBNF needed a different set of parameters for these). Regardless of the number of threads, our speedup gets larger as w increases. Again, this is likely due to the fact that, as the sub-optimality bound increases, states are more likely to be expanded several times. Also, with the exception of 32 threads, our speedup is not as large as what we saw against parallel A* with re-expansions (the previous experiment). PBNF finds lower cost paths than PA*SE. The graph is almost identical to the one from the previous experiment (about 90% of our cost at best) and is omitted for space reasons.

Discussion

It is important to note that PA*SE targets domains where expansions are time consuming. In robotics domains, this is of-

ten the case due to collision checking, running inverse kinematics, or computing complicated cost functions for motions.

In many domains, one could cache the results of these expensive computations the first time a state is expanded so that re-expansions of states are negligible in comparison. When this is possible, PA*SE will not have a significant speedup over PA*.

However there are many domains where caching is not possible. For instance, when memory is constrained due to a large graph, caching the cost of each edge that the search encounters may not be possible. This is especially true for graphs with high branching factors.

Additionally, there are planners commonly used in robotics where caching is not possible because the g-value or parent of the expanded state affects its successors. This happens because the parent of the state (or the g-value) upon expansion affects what is collision checked. Therefore, when a state is re-expanded the computation will have to be done from scratch. For example, Theta* (Daniel et al. 2010) performs collision checks which depend on the parent of the expanded state. SIPP (Phillips and Likhachev 2011) generates successors which depend on the g-value of the expanded state. In the approach presented by (Barraquand and Latombe 1991), collision checking depends on the parent of the expanded state.

Conclusion

In this work, we presented PA*SE, parallel variants of A* and weighted A*, that do not require states to be re-expanded. Our approach is applicable to any planning domain where state expansions (that is, generating successors and computing their edge costs) account for most of the planning time. Robot motion planning is one such problem. The key to our algorithm is to use state independence checks to ensure that a state can expanded safely because no other remaining state can lower its g-value. We prove that PA*SE returns a minimal cost path and when relaxing the independence checks the sub-optimality can be bounded. We also present a parallel version of weighted A*, which finds a solution with a cost that is bounded by a user chosen parameter. We have shown experimentally that PA*SE can lead to linear speedups in the number of processors (as long as the expansions take sufficient time).

One of the interesting theoretical discoveries is that our independence rule is actually a generalization of the A* expansion rule. A* only allows states with a minimum f-value to be expanded, while our theory shows that this rule can be looser. It is future work to consider dramatically different heuristic search algorithms that take advantage of our more relaxed expansion rule.

Acknowledgements

This research has been supported in part by the ONR ANTIDOTE MURI project grant N00014-09-1-1031. We would also like to thank Ethan Burns for allowing us to use his source code for PBNF.

References

Barraquand, J., and Latombe, J.-C. 1991. Nonholonomic multibody mobile robots: controllability and motion planning in the presence of obstacles. In *IEEE International Conference on Robotics and Automation*, 2328–2335.

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.

Daniel, K.; Nash, A.; Koenig, S.; and Felner, A. 2010. Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research* 39:533–579.

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.

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

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*.

Koenig, S., and Likhachev, M. 2005. Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics* 21(3).

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.; and Thrun, S. 2003. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems 16*.

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

Pohl, I. 1970. First results on the effect of error in heuristic search. *Machine Intelligence* 5:219–236.

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.