Proceedings of the Seventeenth International Symposium on Combinatorial Search (SoCS 2024)

Avoiding Node Re-Expansions Can Break Symmetry Breaking

Mark Carlson, Daniel D. Harabor, Peter J. Stuckey
Monash University
mark.carlson@monash.edu, daniel.harabor@monash.edu, peter.stuckey@monash.edu

Abstract

Symmetry breaking and weighted-suboptimal search are two popular speed-up techniques used in pathfinding search. It is a commonly held assumption that they are orthogonal and easily combined. In this paper we illustrate that this is not necessarily the case when combining a number of symmetry breaking methods, based on Jump Point Search, with Weighted A*, a bounded suboptimal search approach which does not require node re-expansions. Surprisingly, the combination of these two methods can cause search to fail, finding no path to a target node when clearly such paths exist. We demonstrate this phenomenon and show how we can modify the combination to always succeed with low overhead.

Introduction

Finding shortest paths in grids is a widely used application in games and robotics, where perhaps thousands of potential paths need to be generated quickly. Because of its importance there is a great deal of research into how to speed it up. Speed-up techniques can be broadly separated into several categories: *abstraction methods*, which reason over a compact representation of the state space, e.g., (Botea, Müller, and Schaeffer 2004; Geisberger et al. 2008); *heuristic improvements*, which strengthen cost estimates used to drive A* search, e.g., (Goldberg and Harrelson 2005; Cohen et al. 2018); *goal bounding and dead-end avoidance*, which prune nodes that cannot belong to any admissible path, e.g., (Björnsson et al. 2005; Hu et al. 2021); *symmetry breaking*, which avoids exploring equivalent paths during the search, e.g., (Harabor and Grastien 2014; Carlson et al. 2023); and *relaxation of optimality*, which trade solution quality for speed, e.g., (Pohl 1970; Pearl and Kim 1982). In this paper we explore the connection between two popular and in principle orthogoal speed-up techniques: symmetry breaking via Jump Point Search (JPS) and optimality relaxation via Weighted A* (WA*).

JPS (2014) (Harabor and Grastien 2014) is a widely used algorithm for breaking path symmetries on uniform-cost grids. JPS speeds up search by scanning the grid for "interesting points", where the search direction is forced to change, and only places these nodes on the queue ("jumping" to them). In doing so it avoids considering many symmetric paths. A number of variant algorithms have been developed since its introduction: Bounded JPS (BJPS) (Sturtevant and Rabin 2016), which limits the distance a jump can go to a bound b; Constrained JPS (CJPS) (Zhao, Harabor, and Stuckey 2023) which further reduces redundant work; and JPS for Weighted maps (JPSW) (Carlson et al. 2023) which extends the JPS ideas to weighted grid maps.

w	JPS	BJPS	CJPS	JPScc	JPSW
1.2	0	4	5	2	12
2	0	78	30	85	180
4	0	423	160	597	631
8	0	765	424	939	1024

Table 1: Number of failures of Weighted A* without re-expansion for JPS-style algorithms across 1,037,980 (JPS, BJPS, CJPS, JPScc) or 1,028,440 (JPSW) problems.

Suboptimal searches trade off some lack of optimality of the path found with an expected shorter runtime. Weighted A* (WA*) (Pohl 1970) is among the simplest and most popular; it works by inflating heuristic estimates using a multiplier $w \geq 1$. Biased by the weighted heuristic, the resulting search is one which trades solution quality for speed. WA* is complete, bounded-suboptimal and fast. Importantly, it achieves these gains without *re-expanding* any nodes (Likhachev, Gordon, and Thrun 2003).

We would expect that symmetry breaking and relaxation methods do not interact, allowing them to be safely used together. However, this turns out not to be the case. We ran JPS, BJPS with bound 8, CJPS, and JPS with corner cutting (JPScc) on 1,037,980 uniform cost problem instances and JPSW on 1,028,440 weighted problem instances, using WA* with various heuristic weights w and without allowing re-expansions. The number of failures (due to either exceeding the bound or failing to find a path) for each of these methods is shown in Table 1. Failures occurred for all methods except the canonical JPS, but were rare; even the worst configuration failed on less than 0.1% of problems.

In the remainder of this paper we explore how and why the combination of WA* and JPS-style algorithms leads to missing paths or paths that exceed the bound promised by WA*. We then develop a revised version of Weighted A*

Copyright © 2024, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

which avoids these problems when used in conjunction with JPS-style algorithms. Finally, we present experimental performance results for our method.

Preliminaries

Given a weighted directed graph $G = (N, E)$ with nodes N and edges $e = (n, n') \in E \subseteq N \times N$ each with associated (positive) weight $d(n, n')$. A path π from node $s \in N$ to $t \in N$ is a sequence $\pi = \langle s = n_0, n_1, \ldots n_m = t \rangle$ where $(n_i, n_{i+1}) \in E, 0 \leq i < m$ with cost $c(\pi) = \sum_{i=0}^{m-1} d(n_i, n_{i+1})$. Define $sp(s, t)$ as the cost of a shortest path from node s to t in G.

A* search is a well-understood search method that makes use of an (*admissible*) heuristic function $h(s, t)$ which gives a lower bound on the distance from node s to t, i.e, $h(s, t) \leq sp(s, t)$. An important additional property of heuristic functions we require is that they are *consistent*, that is, for any nodes n and m, $|h(n, t) - h(m, t)| \leq sp(n, m)$. A* search from s to t starts with an OPEN list containing the start location s. It keeps track of two costs per node: $g[n]$ is the shortest distance from s to n currently found — initially $g[s] = 0$ and $g[n] = +\infty, n \neq s$; $f[n]$ is an under-estimate of the shortest path from s to t via n (at the time we expand n) — initially $f[s] = h(s, t)$ and $f[n] = 0, n \neq s$. We repeatedly take the node n from the OPEN list with minimal value $f[n]$. We then *expand* the node, computing for each edge $(n, n') \in E$ whether $g[n] + d(n, n') < g[n']$ (there is a path to n' via n which is shorter than any previous path) and in this case adding n' to OPEN if it is not there already and setting $g[n'] = g[n] + d(n, n')$ and $f[n'] = g[n] + h(n', t)$. This continues until we remove t from the OPEN list which guarantees that $g[t] = sp(s, t)$. We can reconstruct the actual path using back-pointers in a well-understood manner.

Weighted A*

The A* algorithm may do a large amount of work proving optimality of the solution path when the heuristic is inaccurate. Weighted A* (WA*) (Pohl 1970) is an approach to reducing this problem at the cost of some suboptimality. Given a multiplicative factor w, WA* replaces the heuristic $h(s, t)$ by $h_w(s, t) = w \times h(s, t)$, making it inadmissible and inconsistent. The paths found by WA* are guaranteed to be no longer than $w \times sp(s, t)$, but the search can expand dramatically fewer nodes.

In order to avoid terrible worst case behaviour, weighted A* is usually run without re-expansion. That is once a node n is expanded, we never re-add it to the OPEN list, even if we find a shorter path to it. The reason for this in that without re-expansion WA* performs $O(|N|)$ node expansions, but with re-expansion this grows to $O(2^{|N|})$ on general graphs (Martelli 1977). However, the grid maps we study in this paper do not satisfy the conditions required for this; a better bound on the number of expansions is $O(|N|^2)$ in the worst case (Felner et al. 2011; Chen and Sturtevant 2019). WA* without reopening is still guaranteed to find a path within the bound (Likhachev, Gordon, and Thrun 2003).

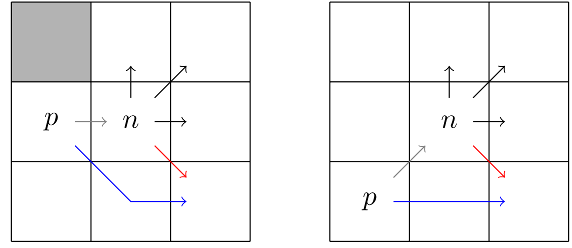

Figure 1: Neighbourhood pruning examples for nodes n with parents p. The blue paths have lower tie-broken cost than the paths using the red moves, so the red moves are pruned. The neighbourhood successors are shown as black arrows.

Grid Maps

This paper studies path planning in grid maps. A grid map is defined by $H \times W$ *cells*, the name we use for vertices of a grid map. For each cell of a weighted grid map we have an associated terrain cost $t \in \mathbb{R}^+ \cup \{+\infty\}$. A *move* is a transition from one grid cell to another adjacent grid cell. Each move is represented as a vector $\vec{m}$ and has a corresponding direction and a magnitude. We distinguish between orthogonal moves $\vec{o}$ of length 1 and diagonal moves $\vec{d}$ of length $\sqrt{2}$. We assume the weight $d(n, n')$ of edge (n, n') is the given by its length by the average terrain cost of the 2 (for othogonal moves) or 4 cells (for diagonal moves) that touch the edge. Some cells are blocked (effectively have a terrain cost $+\infty$). Note that usually movement in grid maps disallows *corner cutting*, that is we cannot move diagonally through the corner point of a blocked cell (note that the weight of such a move is $+\infty$ using the definitions above). In unweighted grid maps when we allow corner cutting the weight of this move in $\sqrt{2}$.

Jump Point Search for Weighted Terrains

JPSW is an optimal and online pathfinding algorithm for such weighted grid maps. It consists of two key ingredients: pruning rules, which eliminate unnecessary moves from consideration; and jumping rules, which skip over cells where the search has a well-understood structure. We describe these in terms of Jump Point Search for Weighted grid maps (JPSW) (Carlson et al. 2023) as its theoretical framework easily accommodates other JPS-style algorithms.

Pruning rules: JPSW assigns to every path π a tie-breaking value $|\vec{m}|$ based on the type of the final move $\vec{m}$ of π. The tie-broken cost of π is then given by the tuple $(c(\pi), |\vec{m}|)$. (Carlson et al. 2023) show that every reachable node has a minimum tie-broken cost path for which every prefix is a minimum tie-broken cost path. It is therefore safe to prune a move $\vec{m}$ from n to n' which is not on a minimum tie-broken cost path to n'. We determine this by attempting to find an alternative path to n' with strictly lower tie-broken cost. If we do not find such a path, then we do not prune $\vec{m}$.

The most important method for deciding whether a move should be pruned when expanding a node n is the neighbourhood pruning strategy. This strategy compares alternative paths starting at the parent of n and contained within the

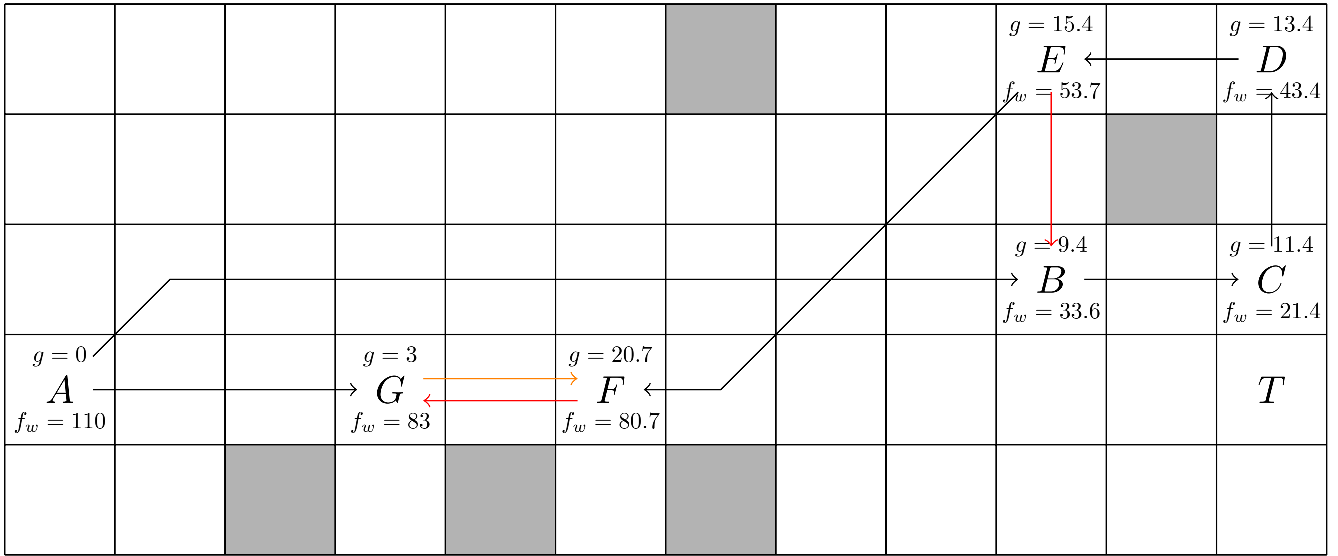

Figure 2: Simplified diagram of the search behaviour of many JPS-style algorithms with heuristic weight 10 and without re-expansions, starting at A with target T. The labelled nodes are expanded in alphabetical order.

3×3 neighbourhood of cells surrounding n to prune moves from n. Note that this depends only on the layout of the neighbourhood and the direction of the parent to n. We call the set of successors not pruned by this rule the *neighbourhood successors* of n. Examples of this strategy are shown in Figure 1. Additionally, the neighbourhood successors of a hypothetical node with parent direction $\vec{d}$ whose neighbourhood has a single unique finite terrain cost is called the *natural successors* of $\vec{d}$.

JPSW utilises two additional strategies for pruning moves under this framework. We omit their descriptions as they are not necessary to understand the content of this paper.

Jumping rules: When a cell is expanded, instead of adding its direct successors to OPEN, JPSW uses a jumping procedure to find alternative successors in the same direction that are farther away. When considering an orthogonal direction $\vec{o}$, the jumping procedure scans in the direction of the move until one of the three stopping conditions is met:

1. The target cell is reached.
2. A cell whose neighborhood is not uniform is reached.
3. The move is no longer possible due to an obstacle.

In cases 1 and 2 the cell that was reached is added to OPEN. In case 3 no successor is generated.

Jumping in a diagonal direction $\vec{d}$ is similar but requires some additional work. Before each diagonal step JPSW requires two orthogonal jumps, in directions $\vec{o_1}$ and $\vec{o_2}$ s.t. $= \vec{o_1} + \vec{o_2} = \vec{d}$. Each of these orthogonal jumps may add a successor to OPEN.

Note that these jumping procedures scan only in the direction of the natural successors of the jump direction. The second stopping condition ensures that the neighbourhood pruning strategy will prune all other potential moves during jumps, although it does not ensure that there are neighbourhood successors which are not natural successors.

Other JPS-style Algorithms

Previous work describes canonical JPS in terms of canonical orderings. We equivalently describe canonical JPS as a variant of JPSW which is specialised to uniform-cost grid maps. The only pruning strategy used is the neighbourhood pruning strategy and the second jump stopping condition is altered to be when the neighbourhood successors is not a subset of the natural successors.

JPS with corner cutting is simply canonical JPS with corner cutting allowed. BJPS modifies canonical JPS to include an additional jump stopping condition, when the jump has travelled a distance of b cells, which produces a successor. CJPS modifies canonical JPS to perform additional pruning using g values observed during search. In particular, during the jumping procedure, g values are updated for cells which did not stop the jump but had the jump direction been different would have. These g values allow CJPS to avoid expanding suboptimal nodes, a pathological behaviour of canonical JPS.

Symmetry Breaking and Relaxation Are Not Independent

Figure 2 shows an example where BJPS, CJPS, JPScc, and JPSW fail to find any path from A to T using the octile distance heuristic with a weight of 10 and no re-expansions. Each method differs slighly in the exact details; we will describe how this occurs broadly and discuss the differences afterwards. We now give a simplified trace of the search procedure as we try to find a path from A to T:

1. A is expanded, finding jump point nodes B and G The obstacle in the top row prevents E from being generated.
2. B is expanded due to its low-weighted heuristic cost, followed by C and D, going around the obstacle.

3. E is expanded and jumps in directions west, south west, and south, finding B and F. This path to B is dominated by the path from A.
4. F is expanded west and south, finding G. This path to G is dominated by the path from A.
5. G is expanded, finding an improved path to F, however F is not re-opened because re-expansions are disabled.
6. The search terminates without finding a path to T.

It is perhaps most insightful to start by discussing why canonical JPS is *successful* in finding T in this case where the other algorithms are not. In step 1, canonical JPS does not find B when expanding A; rather, it finds C directly and does not record a g value on B. Consequently, the search is unaware that the path to B via E is dominated in step 3, and adds B to the open list. B is then expanded and T is found with the peculiar path $\langle A, C, D, E, B, T \rangle$, which goes through B twice.

CJPS behaves similarly to canonical JPS in step 1, with A finding C instead of B. However, during this jump, CJPS places a g value on B so that it can perform additional pruning later (in step 3). Step 3 then occurs as described, finding that the path to B via E is dominated, and the search ultimately fails to find T.

The choice of 8 for the BJPS bound causes the jump from A in step 1 to stop and place a node at B. Step 3 then occurs as described, and the search fails to find T. We should also note that BJPS with a bound of 2 succeeds in a unique way; this choice of bound additionally causes a node to be created at the intersection of the A to B and E to F steps. This allows the search to find that the path via E is dominated at that cell, so F is not found. G is then expanded, and the search soon finds T.

JPScc differs from the example primarily in where the nodes are. Compared to the example, B moves 1 cell right, C moves 1 cell up, D moves 1 cell left and E moves 1 cell down; each of G and F move one cell right. In this case, placing a node at B during the jump from A is unavoidable due to the non-natural successor C.

For JPSW, the example fails using a terrain cost of 3 or more for the gray tiles. The broader jump stopping rules of JPSW mean that there are additional nodes surrounding each gray tile, however the structure of the search remains the same.

Why Failure Occurs

Recall from the description of JPSW that the pruning rules cannot prune any minimum tie-broken cost path. Consider then how the search traverses the minimum tie-broken cost path π. The search will expand each node n_i along π in sequence, finding the successor n_{i+1}. However, other nodes may be expanded between each of these steps, meaning n_{i+1} may already have been discovered. During a suboptimal search, this may result in n_{i+1} being expanded with a suboptimal g value.

In this case, if n_{i+1} has previously been discovered and expanded, it is possible that the required successor n_{i+2} had been pruned. Under our pruning framework, this is clearly possible as no extension to the suboptimal path to n_{i+1} can be minimum cost; that same extension to the prefix of π ending at n_{i+1} has lower cost. The pruning rules are therefore allowed to prune any successor of n_{i+1} in the suboptimal expansion, including the next node on π, n_{i+2}.

In optimal search, this case simply does not occur. In Weighted A* with re-expansions, after n is expanded n_{i+1} is re-added to the open list, and will later be expanded with a minimum tie-broken cost path. However, when re-expansions are disabled, n_{i+1} is never expanded with a minimum tie-broken cost path. The effect is that π is pruned, which may result in all paths to the target being pruned.

We can see this occur in the example in Figure 2. The minimum tie-broken cost path is $\langle A, G, F, T \rangle$. F is expanded with a suboptimal g value, and happens to prune its rightwards successor. However, when G is expanded and finds the minimum tie-broken cost path to F, F is not re-added to the open list and is not re-expanded. Therefore, the path $\langle A, G, F, T \rangle$, which is the only path the pruning rules are guaranteed not to prune, is removed from consideration and no path to T is found.

Algorithm 1 FLS: Search using our focal list strategy.

Require: expanded[n] = false for all $n \in N$
Require: $\{s, t\} \subseteq N$ % start + target
```
1: OPEN ← {s}
2: FOCAL ← {s}
3: while OPEN ≠ {} do
4:    if FOCAL = {} or
      f_w[top(FOCAL)] > wf[top(OPEN)] then
5:       n ← pop(OPEN)
6:    else
7:       n ← pop(FOCAL)
8:       remove(OPEN, n)
9:    end if
10:   if n = t then
11:      return g[t]
12:   end if
13:   expanded[n] ← true
14:   for (n, n') ∈ E do
15:      if g[n] + d(n, n') < g[n'] then
16:         update g, f, f_w for n'
17:         push_or_update(OPEN, n')
18:         if ¬expanded[n'] then
19:            push_or_update(FOCAL, n')
20:         end if
21:      end if
22:   end for
23: end while
```

Fixing with Focal Search

We propose to fix the problem of the interaction of WA* with symmetry pruning approaches by using a strategy based on focal search (Pearl and Kim 1982). Our approach, FLS, maintains two queues, OPEN ordered by $f = g + h$ and FOCAL ordered by $f_w = g + wh$. When a node is relaxed, we add it to OPEN but only add it to FOCAL if it has not been expanded yet. To select a node for expansion, we take from

FOCAL when f_w[top(FOCAL)] $\leq wf$[top(OPEN)] and otherwise take from OPEN. When we take a node from FOCAL, we remove it from OPEN. Note that if we choose to expand a node n from OPEN, the selected node cannot be on FOCAL as this would require $f_w[n] > wf[n] \Rightarrow g[n] > wg[n]$, which is not true. Pseudocode is given in Algorithm 1.

Lemma 1. *FLS expands each node at most twice.*

Proof. Once a node n is expanded the first time then $expanded[n]$ is set. Hence it cannot be re-added to FOCAL. Hence if n is expanded again it must be expanded from the OPEN list. Since the top of the OPEN list has the least f value, no expansion can reduce it g value, as h is consistent. Thus the node n will not be re-added to OPEN. □

Using this strategy, we ensure that no node is expanded more than twice, once sub-optimally from FOCAL and once optimally from OPEN, which prevents the exponential worst-case of WA* with re-expansions. The condition for expanding from FOCAL explicitly maintains the required suboptimality bound. Additionally, it delays the re-expansion of nodes as long as possible, until the cost-to-go suggested by FOCAL exceeds the lower bound maintained by OPEN.

Theorem 1. *FLS always finds a w bounded-suboptimal path from s to t if one exists.*

Proof. (Sketch) The w bounded suboptimality simply arises since no node expanded from the FOCAL list has an f_w value above $w \times sp(s,t)$ since it is not greater than $w \times f_{top}$ where f_{top} is the f value on the top of open and $f_{top} \leq sp(s,t)$ since h is admissible.

Now we argue how we always find a path from s to t if one exists. Consider the optimal path p from s to t that is visited by the underlying symmetry breaking approach (JPS, BJPS, CJPS, JPScc, JPSW). We show that each node appearing in this path must appear in OPEN with the correct g value at some time, if we find no other path to t. s is on OPEN initially, and expanding every node n on p places the next node n' on the path p in OPEN with the correct g value. Either FLS finds a path to t, or it must empty the OPEN list. Hence either the entire path p will be expanded eventually reaching t, or another path to t will be found. □

When we use this strategy to search the example in Figure 2, the search proceeds in the same manner until F is reached via G after F had already been expanded. F is added back to OPEN, but not re-added to FOCAL. FOCAL is now empty and so F is selected from OPEN to expand, resulting in T being added to OPEN and FOCAL. T is then selected from FOCAL and the goal is found.

Experimental Setup

We implemented our focal list in C++ using the warthog pathfinding library. Full code is available at `https://bitbucket.org/mcar0024/pathfinding/src/jps-wastar-fixing/`. The experiments were run on an AMD Ryzen 9 5950X with 16 GB 3200MHz DDR4 memory.

For the uniform cost algorithms, we tested with the Iron Harvest benchmarks (Harabor, Hechenberger, and Jahn 2022) and many of the standard Moving AI grid map benchmarks (Sturtevant 2012). Specifically, we used the *Baldur's Gate II* unscaled and scaled to 512×512, *Dragon Age: Origins*, *Starcraft*, *Random*, *Room*, and *Street* benchmark sets from Moving AI. In total there were 661 maps and 1,037,980 problem instances.

For JPSW, we used the *Baldur's Gate II* unscaled and scaled to 512×512, *Dragon Age: Origins*, *Starcraft*, *Random*, *Room*, *Street*, *Warcraft III*, and *Terrain* benchmark sets from Moving AI. In total there were 682 maps and 1,028,440 problem instances. A limitation of these benchmarks is that most of them were designed for evaluating uniform cost grid pathfinding algorithms, although Sturtevant chose to include several terrain types on some maps. We chose to assign ground ('.') weight 1, trees ('T') weight 1.5, shallow water ('S') weight 2, water ('W') weight 4, and out-of-bounds ('@') weight 10. The *Terrain* maps are different from the others in that they contain significantly more terrain types which are not intended to represent anything. For these maps, we assigned each terrain type a weight chosen uniformly at random, then shifted and scaled such that the least expensive terrain had weight 1 and the most expensive had weight 20.

Experimental Results

The first experiment evaluates the failure rate of Jump Point Search variants when combined with Weighted A* *without re-expansions*. We used heuristic weights 1.2, 2, 4, and 8 with each JPS variant: canonical JPS (JPS), bounded JPS (BJPS), constrained JPS (CJPS), JPS with corner cutting (JPScc) and JPS for Weighted Maps (JPSW). Table 1 shows the number of failures we saw with each combination: failures can be of two kinds, in some cases we return a shortest path outside the suboptimality bound; in others we fail to find any path at all. Note all the instances have a feasible path. Clearly failures occur for every variation except JPS; but they are rare (0.1% of problems in the worst case). This shows that the problem we explore is not just restricted to carefully constructed examples. Failures increase as we increase w; this is unsurprising since we have more chance to first expand nodes on an optimal path in the wrong direction in this case.

From the results of the first experiment it is clear that we cannot safely use WA* without re-expansions. For the remainder of the experiments we compare Weighted A* with re-expansions, versus our focal search approach.

The next experiment explores how often applying the "speed-up" technique of WA* (with re-expansions) to CJPS lead to slowdowns; and how using FLS rather than WA* affects this. Table 2 compares the two methods across all maps and the hardest 50% (defined as those where CJPS took more time than the median solve time). We can see that both methods improve on CJPS on almost all instances. The smaller the sub-optimality bound the more work the focal approach has to do, but it is comparable with re-expansion for larger w values. While across all maps it appears that WA* is still preferable to FLS, when we consider the hardest

	All		Hardest 50%	
w	WA*	FLS	WA*	FLS
1.2	**33**	201	**5**	58
2	**38**	87	11	**3**
4	**49**	95	13	**2**
8	**56**	103	13	**2**

Table 2: Number of maps (of 661) solved slower than optimal CJPS, both overall and for CJPS's most difficult 50%.

	All		Hardest 50%	
w	WA*	FLS	WA*	FLS
1.2	**11**	19	**4**	12
2	42	**26**	16	**0**
4	53	**36**	17	**0**
8	62	**40**	19	**0**

Table 3: Number of maps (of 661) solved with fewer node expansions than optimal CJPS, both overall and for CJPS's most difficult 50%.

maps we see that FLS is much more robust here. Table 3 similarly compares the two in terms of node expansions. FLS performs comparatively better here, which is unsurprising as FLS has more costly open list operations.

A more detailed comparison of the methods for $w = 2$ is shown in Figure 3, which plots the runtime of optimal CJPS versus the "speed-up" techniques using WA* and FLS. Note that the comparison is a log-log plot. Clearly both WA* and FLS generally lead to substantial speed-ups, and some minor slowdowns particularly for the easier examples. What is clear is that the worst case behaviour of WA* is concentrated on the harder examples.

We repeated the previous experiment comparing WA* as a "speed-up" technique for JPSW; and how using FLS affects this. The results are shown in Table 4. As in the previous experiment, over all maps WA* is causing slowdown less often than FLS, but in the hardest maps (for JPSW) FLS is more robust. Similarly, Table 5 compare in terms of node expansions. Like with CJPS, this comparison favours FLS due to the higher cost of open list operations.

Figure 4 and 5 show the comparison in more detail for $w = 2$ and $w = 4$. Once again on the quickest examples both WA* and FLS can lead to reasonable slowdowns, but for the hardest examples FLS completely avoids the worst case behaviour of WA*. Indeed for $w = 4$ it remains far faster than optimal JPSW for all the larger instances.

Part of the reason for relatively poor performance of FLS compared to WA* on smaller maps is due to the overhead from performing double the queue operations. We measured this cost by running FLS without a heuristic weight, which behaves identically to A* except with extra queue operations. We found that for CJPS the overhead was 10%, and for JPSW the overhead was 25%. However, this is small compared to the potentially very large worst case behaviour of WA* that FLS avoids.

The fourth experiment examines the suboptimality factor arising from using WA* versus FLS. The results in Table 6

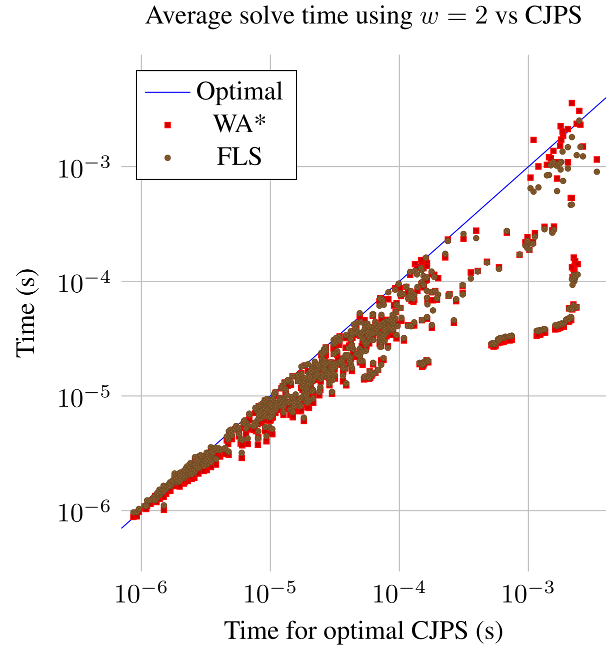

Figure 3: Average run time comparison using Optimal CJPS, Weighted A* CJPS with re-expansion, and our focal list, for $w = 2$

	All		Hardest 50%	
w	WA*	FLS	WA*	FLS
1.2	**49**	161	**37**	51
2	**84**	89	42	**1**
4	**90**	98	14	**0**
8	**72**	97	**0**	**0**

Table 4: Number of maps (of 682) solved slower than optimal JPSW, both overall and for JPSW's most difficult 50%.

show that there is essentially no difference, a tiny increase in suboptimality for FLS. This is unsurprising since both algorithms act almost identically unless WA* is forced to re-expand nodes in which case FLS will also expand nodes (but at most twice). This tiny increase in sub-optimality arises because avoiding re-expansion can cut off a better path to the goal.

Note that we were somewhat surprised by the results for WA* with re-expansion, since we found in limited experiments that applying Weighted A* to standard grid search often led to catastrophic explosion in the search time. This illustrates how Jump Point Search variants, by placing only a limited number of grid cells in the queue, actually work to mitigate the problems of Weighted A* with re-expansion. In particular they appear to avoid the behaviour where re-expanding one node leads to a re-expansion of a large number of reachable nodes from that node.

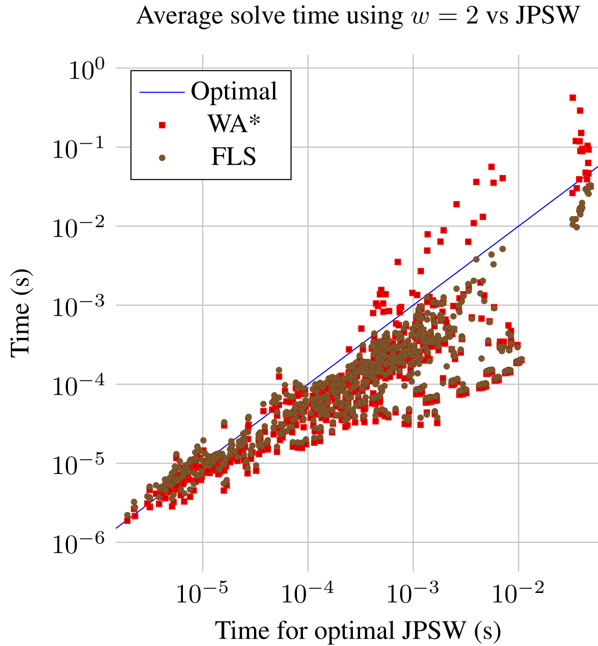

Figure 4: Average run time comparison using Optimal JPSW, Weighted A* JPSW with re-expansion, and our focal list, for $w = 2$

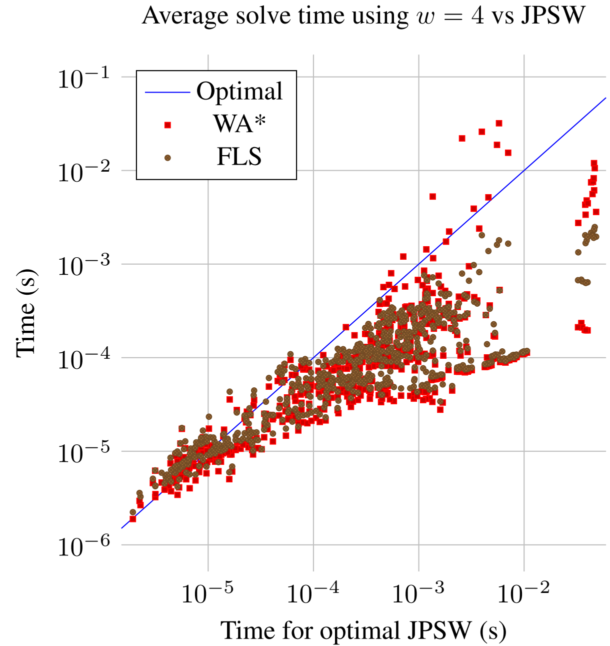

Figure 5: Average run time comparison using Optimal JPSW, Weighted A* JPSW with re-expansion, and our focal list, for $w = 4$

	All		Hardest 50%	
w	WA*	FLS	WA*	FLS
1.2	75 (49)	**38** (161)	50 (37)	**19** (51)
2	98 (84)	**50** (89)	46 (42)	**0** (1)
4	105 (90)	**81** (98)	16 (14)	**0** (0)
8	82 (72)	**76** (97)	**0** (0)	**0** (0)

Table 5: Number of maps (of 682) solved with fewer node expansions than optimal JPSW, both overall and for JPSW's most difficult 50%.

	CJPS		JPSW	
w	WA*	FLS	WA*	FLS
1.2	1.006	1.007	1.009	1.010
2	1.032	1.034	1.079	1.081
4	1.074	1.076	1.293	1.294
8	1.102	1.104	1.643	1.644

Table 6: Geometric mean suboptimality factor

Conclusion

In this paper we have demonstrated that two A* speed-up methods that are notionally independent: symmetry breaking, and optimality relaxation, can interact in unexpected ways. This behaviour, while rare, can be catastrophic, where sometimes we not only violate the bound, but fail to find any path at all. We then show how we can build an extension of Weighted A* using a focal list to avoid this behavior. Our new approach does not lose any of the good performance of Weighted A*, but guarantees to find a solution within the suboptimality bound w.

We conjecture that canonical JPS with WA* is complete, and certainly we have no experimental evidence to refute this. The proof eludes us because JPS does not expand the same jump point twice, even if it reaches it from different directions, which means the first expansion can prevent later expansions in different directions, which seems like a potential way to lose paths. But, because of the complex interactions of jump points with the blocked terrain defining the map topology it appears these lost paths are never "fatal" to the overall search. A proof (or counter example) remains as interesting future work.

Acknowledgements

This research is supported by an Australian Government Research Training Program (RTP) Scholarship and Australian Research Council Grant DP200100025.

References

Björnsson, Y.; Enzenberger, M.; Holte, R. C.; and Schaeffer, J. 2005. Fringe Search: Beating A* at Pathfinding on Game Maps. In *IEEE Symposium on Computational Intelligence and Games*, 125–132.

Botea, A.; Müller, M.; and Schaeffer, J. 2004. Near Optimal Hierarchical Path-Finding. *Journal of Game Development*, 1(1): 7–28.

Carlson, M.; Moghadam, S. K.; Harabor, D. D.; Stuckey, P. J.; and Ebrahimi, M. 2023. Optimal pathfinding on weighted grid maps. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 12373–12380.

Chen, J.; and Sturtevant, N. R. 2019. Conditions for avoiding node re-expansions in bounded suboptimal search. *puzzle*, 40: 39–753.

Cohen, L.; Uras, T.; Jahangiri, S.; Arunasalam, A.; Koenig, S.; and Kumar, T. K. S. 2018. The FastMap Algorithm for Shortest Path Computations. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, 1427–1433.

Felner, A.; Zahavi, U.; Holte, R.; Schaeffer, J.; Sturtevant, N.; and Zhang, Z. 2011. Inconsistent heuristics in theory and practice. *Artificial Intelligence*, 175(9-10): 1570–1603.

Geisberger, R.; Sanders, P.; Schultes, D.; and Delling, D. 2008. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In *WEA*, 319–333.

Goldberg, A. V.; and Harrelson, C. 2005. Computing the Shortest Path: A* Search Meets Graph Theory. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 156–165.

Harabor, D.; and Grastien, A. 2014. Improving jump point search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 24, 128–135.

Harabor, D.; Hechenberger, R.; and Jahn, T. 2022. Benchmarks for pathfinding search: Iron harvest. In *Proceedings of the International Symposium on Combinatorial Search*, volume 15, 218–222.

Hu, Y.; Harabor, D.; Qin, L.; and Yin, Q. 2021. Regarding Goal Bounding and Jump Point Search. *J. Artif. Intell. Res.*, 70: 631–681.

Likhachev, M.; Gordon, G. J.; and Thrun, S. 2003. ARA* : Anytime A* with Provable Bounds on Sub-Optimality. In Thrun, S.; Saul, L.; and Schölkopf, B., eds., *Advances in Neural Information Processing Systems*, volume 16. MIT Press.

Martelli, A. 1977. On the complexity of admissible search algorithms. *Artificial Intelligence*, 8(1): 1–13.

Pearl, J.; and Kim, J. H. 1982. Studies in semi-admissible heuristics. *IEEE transactions on pattern analysis and machine intelligence*, (4): 392–399.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial intelligence*, 1(3-4): 193–204.

Sturtevant, N. R. 2012. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2): 144–148.

Sturtevant, N. R.; and Rabin, S. 2016. Canonical Orderings on Grids. In *IJCAI*, 683–689.

Zhao, S.; Harabor, D.; and Stuckey, P. J. 2023. Reducing Redundant Work in Jump Point Search. In *Proceedings of the International Symposium on Combinatorial Search*, volume 16, 128–136.