Proceedings of the Ninth International
Symposium on Combinatorial Search (SoCS 2016)

Repair Policies for Not Reopening Nodes in Different Search Settings

Vitali Sepetnitsky
ISE Department
Ben-Gurion University
Be'er-Sheva, Israel
(sepetnit@bgu.ac.il)

Ariel Felner
ISE Department
Ben-Gurion University
Be'er-Sheva, Israel
(felner@bgu.ac.il)

Roni Stern
ISE Department
Ben-Gurion University
Be'er-Sheva, Israel
(roni.stern@gmail.com)

Abstract

We examine two policies for reopening of nodes: *never reopen* (NR) and *always reopen* (AR). While there are circumstances where each policy is beneficial, we observed empirically that NR is usually faster. However, NR may fail to return a solution of the desired quality in two scenarios: (1) in a *bounded suboptimal search* when inconsistent heuristics are used and (2) in a *bounded cost* setting. To remedy this we provide two *repair* policies for NR when the desired quality was not obtained. The first policy is to restart AR. The second policy is to repeatedly place the nodes that were not reopened back in OPEN and continue with NR. Experimental results show that both repair polices outperform the AR policy.

1 Introduction and Overview

It is well-known that A* with a *consistent* heuristic[1] ($\equiv f$-cost is *monotonically increasing* along paths) expands a state only after the lowest cost path to it was found. This is not the case for *non-monotonic* f-functions (where f might decrease along paths) which may occur if h is inadmissible or even if it is admissible but is inconsistent. In such cases, a state n may be generated with a smaller g-value, after it has already been expanded. At this point, n can be removed from CLOSED and put back in OPEN – an action called *reopening*. Reopening is optional – we might choose not to reopen. In this case the newly seen state remains in CLOSED and not placed back in OPEN (it is a common practice to update its g-value and its parent pointer when applicable). Two baseline reopening polices are *always reopen* (AR) and *never reopen* (NR).

If the optimal solution is required and the heuristic is admissible then the A* algorithm (Hart, Nilsson, and Raphael 1968) is guaranteed to return the optimal solution. However, in many cases obtaining the optimal solution is not feasible or is not needed. Some problems, especially those who grow exponentially, require a large amount of computing resources (i.e., both time and memory). In addition, some applications (e.g., video games, embedded systems or mobile apps) significantly restrict the amount of time (sometimes below 1ms (Bulitko et al. 2011)) or memory allowed for problem solving. In such cases, one must settle for a suboptimal solution by trading time/memory for solution quality.

In this paper we focus on two suboptimal search settings where non-monotonic cost functions may occur:

- **Bounded-suboptimal search** (Thayer and Ruml 2011) (denoted here as BSS). In BSS(B) we are given a bound B and the task is to find a solution with cost $\leq B \times C^*$ where C^* is the cost of the optimal solution (P_{opt}).
- **Bounded-cost search** (Stern et al. 2014) (denoted here as BCS). In BCS(C) we are given a cost C and the task is to find a solution with cost $\leq C$.

In both cases the aim is to quickly find a solution that is within a given bound (either constant or relative to P_{opt}).

At a first glance it seems that performing more reopening will improve the quality of the solutions (as better paths to the node in question are being considered) but increase the number of node expansions (as the subtree below the node might be expanded again). However, we classify 9 different possible relations between AR and NR and present an example graph where most of these cases exist by only varying the weight for *Weighted A** (WA*). We then show that these cases are evident on a variety of common testbed domains.

Nevertheless, our experiments show that on average NR expands fewer nodes than AR. Therefore, NR should be preferred if it is guaranteed to return a solution within the bound. But, there are scenarios where NR may fail to return a solution within the bound. We identify two such scenarios: **(1:)** in BSS when h is inconsistent. **(2:)** in BCS even if h is consistent.

A simple solution for such cases would be to activate AR and pay its overhead. However, in this paper we suggest an alternative: *NR with repair*. First, NR is activated. If it turns out that the desired quality was not obtained then the solution returned should be *repaired*. We discuss and study two *repair* policies for NR: **(1:)** Fallback to AR. **(2:)** Move the nodes that were not reopened back to OPEN and continue the NR search repeatedly. These methods are general; they are applicable across both search settings (BSS and BCS) and for many search algorithms. Experimental results show that both repair polices outperform AR on a variety of domains on both search settings and with various search algorithms.

[1] h is consistent if $\forall(u,v) : |h(u) - h(v)| \leq cost(u,v)$

		Path returned			Nodes Expansions					
Range	W	AR	NR	P	Subrange	Trend	W	X_i	AR	NR
1	$1 \leq W < 3.75$	P_{opt}	P_{opt}	$P^{=}$	1a	$P^{=}N^{=}$	$1 \leq W < 3$	both	11	11
					1b	$P^{=}N^{NR}$	$3 \leq W < 3.75$	both	13	11
2	$3.75 \leq W < 4.25$	P_2	P_{opt}	P^{NR}	2a	$P^{NR}N^{NR}$	$3.75 \leq W < 4$	both	12	11
					2b	$P^{NR}N^{AR}$	$4 \leq W < 4.25$	NR	8	11
3	$4.25 \leq W < 5$	P_2	P_3	P^{AR}	3a	$P^{AR}N^{AR}$	$4.25 \leq W < 4.5$	NR	8	10
					3b	$P^{AR}N^{NR}$	$4.5 \leq W < 5$	None	8	6
4	$5 \leq W$	P_3	P_3	$P^{=}$	4	$P^{=}N^{=}$	$5 \leq W$	None	5	5

Table 1: The path and expansions for different values of W

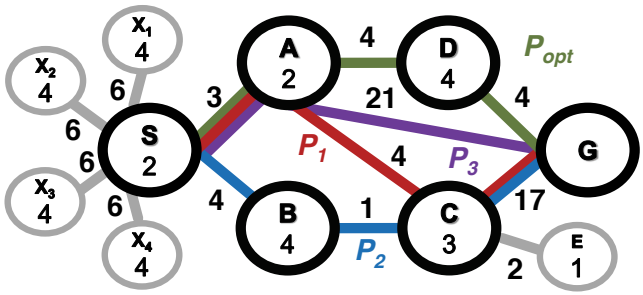


Figure 1: An Example Graph that shows the trends

2 A Case Study: Weighted A* for BSS

In this section we perform a case study of NR and AR in the BSS setting when using *Weighted A** (WA*) (Pohl 1970). WA* is perhaps the most famous and simple BSS(B) algorithm. WA* prioritizes nodes in OPEN according to $f(n) = g(n) + W \cdot h(n)$, where $g(n)$ is the cost of the lowest known path from the start state to n, $h(n)$ is an admissible heuristic of reaching the goal from n and $W \geq 1$ is a parameter which may cause the f-function to be non-monotonic.

The notion of reopening (within the context of WA*) has been first introduced by (Pohl 1970) which proved that setting $W = B$ will satisfy the constraints of BSS(B) if AR is used. Likhachev et al. (2004) proved that WA* with NR (where $W = B$), is also guaranteed to find a solution within the desired bound for BSS(B), given that h is consistent. A major question to ask in this context is as follows:

What is the tradeoff between NR and AR in terms of number of nodes expanded and in the quality of the solution returned?

2.1 A graph with different trends

Different possible relations between AR and NR can occur regarding the path returned and the number of expanded nodes. In Figure 1 we show an example graph which demonstrates this by only modifying the weight W but without changing the structure of the graph.[2]

Let $P(AR)$ and $P(NR)$ denote the length of the path returned by the AR and NR policies, respectively. Let $P^{AR}, P^{=}, P^{NR}$ denote a win for AR ($P(AR) < P(NR)$), a tie ($P(AR) = P(NR)$) and a win for NR ($P(NR) < P(AR)$), respectively. Similarly, let $N(AR)$ and $N(NR)$ denote the number of nodes expanded by the AR and NR policies, respectively. Finally, let $N^{AR}, N^{=}, N^{NR}$ denote a win for AR ($N(AR) < N(NR)$), a tie ($N(AR) = N(NR)$) and a win for NR ($N(NR) < N(AR)$), respectively. There are $3 \times 3 = 9$ combinations and we use a four character notation P^*N^* to denote these cases. For example, $P^{AR}N^{NR}$ denotes the case where in the path aspect (P), AR is the winner, while in the nodes-expanded aspect (N), NR is the winner.

In the graph given in Figure 1(Left) the task is to go from node S to node G (the number in each node is its heuristic value). For now, assume that states E, X_1, X_2, X_3, X_4 and their connecting edges (all colored in gray) do not exist in the graph. The graph has four paths from S to G: $P_1 = \{S, A, C, G\}$ of cost 24, $P_2 = \{S, B, C, G\}$ of cost 22, $P_3 = \{S, A, G\}$ of cost 24 and the optimal path, $P_{opt} = \{S, A, D, G\}$, of cost 11. The exact step by step executions of AR and NR for different values of W are omitted (they are provided in a technical report (Sepetnitsky, Felner, and Stern 2014)) but we summarize them now.

Node C is first generated by path P_1 but C is seen again when B is chosen for expansion. For different ranges of W different paths are returned as shown in the left part of the table in Table 1. For very large values (*Range 4*), reopening of C will never happen as B will never be chosen for expansion, hence both policies tie. For very small values (*Range 1*), reopening will make no difference as P_{opt} will be returned anyway – again a tie. *Range 3* is the intuitive case where AR is expected to result in a better path that NR. With AR, C is reopened and P_2 is returned, while with NR, C is not reopened and P_3 is returned. A somewhat counter-intuitive case is *Range 2*, where AR finds a better path (P_2) and halts, but NR continues to search and eventually finds P_{opt}.

Adding the grayed states and edges to the graph enriches the ranges of W to also have different tendencies in node expansions. These additions split the four ranges into sub-

[2] A number of authors discussed the influence of AR and of NR on the path returned and on the number of expanded nodes (Thayer and Ruml 2010; Malima and Sabanovic 2007; Valenzano, Sturtevant, and Schaeffer 2014). For example, (Hansen and Zhou 2007) discuss some reasons for why AR may require fewer node expansions than NR in some cases, but did not discuss the case where AR finds worse solutions than NR as we now show.

ranges (based on which algorithm expands the X_i nodes) depicted in the right part of the table in Table 1. Thus, in this example graph we have 6 out of the 9 possible cases of P^*N^* (see the *Trend* column) by only changing the values of W.

2.2 Experimental results

We experimented with AR and NR on a number of domains. For each domain we generated 100 random instances and used a range of values for W. We experimented with WA* and with *Explicit Estimation Search* (EES) (Thayer and Ruml 2011). EES is a special case of *focal search*. EES uses a FOCAL list of nodes which includes all nodes $n \in$OPEN with $f(n) \leq B \times f_{min}$, where f_{min} is the minimal f-value in OPEN. EES also uses two other inadmissible estimates $\hat{h}$ and $\hat{d}$ which are learned throughout the search execution. EES has specific rules based on $\hat{h}$ and $\hat{d}$ as to which node from FOCAL to expand next. EES was shown to perform very well on many domains for the BSS setting (Thayer and Ruml 2011).

Figure 2 shows results on the 15-puzzle (Manhattan distance heuristic), on the map `den400d` from (Sturtevant 2012)[3] and on a dock robot planning domain (Nau, Ghallab, and Traverso 2004) (with distances at most 2 and at most 30% of boxes out of places) for WA* (top) and EES (bottom). The figure shows the percentage of instances, partitioned into five disjoint groups according to the cost of the found path. The bottom three slices (below the black line) highlight the "anomalies" which contain the *counter intuitive* cases – when NR finds a *strictly* better path than AR (P^{NR}) and when AR expands *strictly* less nodes than NR (N^{AR}). The two bottom slices (below the dotted line) are the (N^{AR}) cases where AR expands fewer nodes.

2.3 Which policy to choose?

The above discussion revealed that there are particular instances where each of the policies is better in one or in two of the measurements. However, when focusing only on the number of nodes expanded, Figure 2 shows strong tendency that in nearly 90% of the cases, the number of nodes expanded by NR is *less than or equal to* the number of nodes that AR expands (both guarantee a solution within the bound). In fact, even for the case of $W = 2$ for the 15-puzzle (the best situation for AR across all our experiments) in nearly 70% of the cases NR won in the nodes expanded count.

In problems like BSS and BCS, which are the focus of this paper, solutions are acceptable as long as they are guaranteed to be within the bound (their exact length is of less importance). Based our results, if NR is guaranteed to return a solution within the bound then it should be the policy of choice as it is usually faster than AR and is easier to implement. The key question is what are the circumstances where NR will fail to return a solution within the bound.

[3]Similar results (not shown) were obtained on other maps namely on `brc202d` and `ost003d`

3 When NR Fails

There are two possible settings (BSS with an inconsistent heuristic and BCS in general) where NR may fail to return a solution within the desired quality while AR is guaranteed to do so. We deal with each of these in turn.

3.1 BSS(B) with inconsistent h

The proof by Likhachev et al. (2004), that WA* with NR will also meet the desired bound, explicitly assumes that h is consistent. (See their sections A.2 and A.3.2) But, it does not hold for inconsistent h.

The example in Figure 3 shows that WA* fails to return a solution within the bound when h is inconsistent.[4] The optimal path (denoted P_{opt}) from S to G costs 4. Numbers inside the nodes are their admissible h-values. h is inconsistent because $h(A) - h(B) = 2 > cost(A, B) = 1$. Assume that the desired suboptimality bound is set to 1.1 and that we activate WA* with $W = 1.1$ and with NR. First, S is expanded and three nodes are generated: A with $f(A) = 4.3$, B (via P_1) with $f(B) = 3.7$ and G (via P_2) with $f(G) = 4.6$. B is now expanded and G is generated again with $f(G) = 4.6$ (no need to update the parent pointer). Next, A is expanded ($f(A) = 4.3$) and B is generated again, now with $f(B) = 3.1$. Since NR is applied B is not reopened (we do, however, update the parent pointer of B to A). Finally, G is expanded and path P_2 is returned with cost 4.6. But, since $W = 1.1$ the desired solution quality is at most 4.4. By contrast, when applying AR, B is reopened with $f(B) = 3.1$. B will now be expanded and G will be generated via P_{opt} with $f(G) = 4$. Finally, G is expanded and P_{opt} is returned.

Providing a similar example in which EES with an inconsistent heuristic and NR also fails to return a bounded solution is more involved, as it required defining all the auxiliary heuristics used by EES. Nevertheless, experimental evidence that this occurs for EES too is provided later in the paper.

3.2 BCS

Potential Search (PTS) (Stern et al. 2014) is an algorithm specifically designed for BCS. PTS is a best-first search algorithm which chooses to expand the node, n, from OPEN with the largest "potential", $u(n)$ which is defined as $u(n) = \frac{C-g(n)}{h(n)}$. They showed that nodes with larger potential are expected to lead to a goal with a solution cost within the bound. In addition, for an admissible h, the algorithm prunes any node n for which $f(n) = g(n) + h(n) > C$. $u(\cdot)$ is not necessarily monotonic ($u(\cdot)$ may either increase or decrease when moving from a parent to a child). While Stern

[4]The example was inspired by the example graphs given by (Valenzano, Sturtevant, and Schaeffer 2014). In addition, for the case of inconsistent heuristic (Valenzano, Sturtevant, and Schaeffer 2014) showed that when a best-first search is employed, the solution returned by NR cannot be larger than the solution returned by AR by more than the inconsistency along the optimal path defined as $\sum_{j=1}^{k-1} INC_h(n_j, n_{j+1})$. That analysis does not refer to the bound B; NR with an inconsistent h may return a solution larger than B.

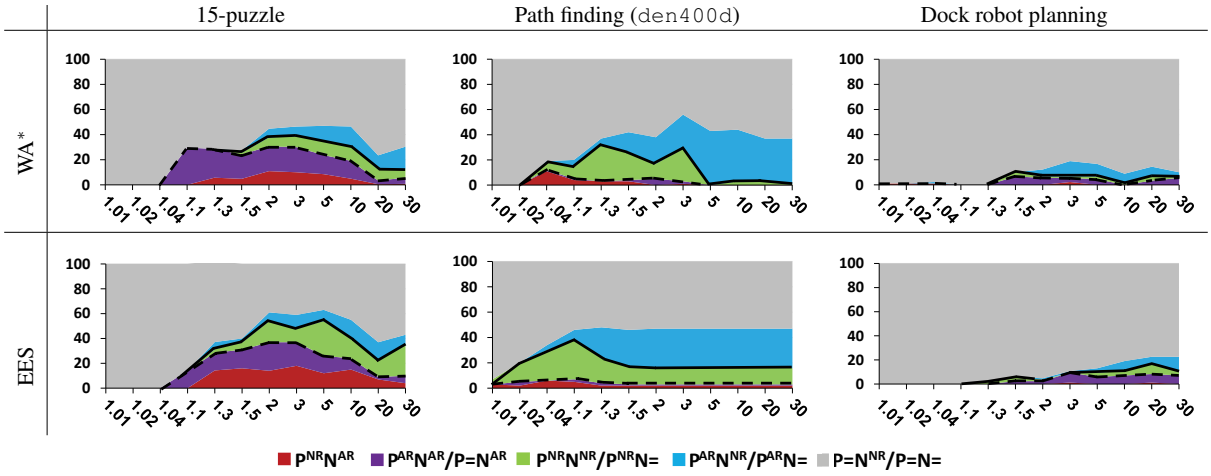


Figure 2: Percentage of instances per class (y-axis) as a function of different values of the bound B (x-axis) for WA* and EES

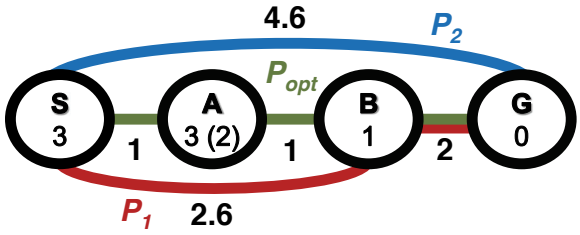


Figure 3: Examples of NR failures on WA* and PTS

et al. (2014) do not specifically state this, the pseudo code of PTS provided in that paper (page 6, Lines 6-7) clearly reopens nodes. Furthermore, Stern et al. do not specifically prove that PTS will find a desired solution if such exists. But, this is a straightforward derivation of their pseudo code as follows. PTS is a best-first search. In a finite state-space, at some point it will generate the entire graph (and via all paths if AR is activated). In particular, P_{opt} will be found at some point.

However, PTS with NR may fail to return a solution within the bound C. For example, consider Figure 3 again. In order to show that the phenomenon also occurs with a consistent h we alter $h(A)$ to be 2 (was $h(A) = 3$). Assume that we set $C = 4.5$. After expanding S, G is revealed via P_2 with $f(G) = 4.6$. But since $f(G) > C$, G is discarded. In addition, A (via P_{opt}) and B (via P_1) are generated with $u(A) = 1.75$ and $u(B) = 1.9$. B is now expanded and, again, G (with $f(G) = 4.6$) is generated and discarded. Next, A is expanded ($u(A) = 1.75$) and B is generated again, now with $u(B) = 2.5$ (larger than the previous value of $u(B) = 1.75$). Since NR is applied, B is not reopened. Finally, OPEN is empty and the algorithm returns no solution $\leq C = 4.5$. By contrast, PTS with AR will reopen B with $u(B) = 2.5$. B is now expanded and G via P_{opt} is revealed with a path of cost 4 which is within the required bound $C = 4.5$.

Bounded-cost Explicit Estimation Search (BEES) (Thayer et al. 2012) modifies EES to the BCS setting by modifying FOCAL to be all nodes n in OPEN with $f(n) \leq C$. BEES too suffers from the problem that with NR it may fail to return a solution within the bound (example is omitted, results below support this).

An immediate solution for both these settings will be to use the AR policy. However, as discussed above AR will expand more nodes on most of the cases. In the next sections we provide an alternative that enjoys the complementary benefits of both AR (solution quality guarantee) and of NR (fewer nodes expansions on average). It is called *NR with repair*.

4 NR with Repair

While NR is not guaranteed to always return a solution within the bound for the settings above, it might succeed in doing this for many particular problem instances. Whether the solution is within the bound is checked differently for the two settings:

BCS: a solution P is within the bound if $P \leq C$.

BSS: a solution P is within the bound if $P \leq B \times L$ where L is a lower bound on P_{opt}. In many cases L is obtained by setting $L = f_{min}$, where f_{min} is the minimal f-value in OPEN (we use an extended version of this bound as elaborated below).

For instances where NR was lucky to return a solution within the bound it will, on most cases, be faster than AR. Based on this, the main idea of *NR with repair* (NRR) is to run NR until a solution is returned. Now, if the solution is within the bound, the algorithm halts. Otherwise, we need to "repair" the solution and decrease it to fit the bound. We provide two such repair strategies next. Importantly, both repair strategies are general and may work on both BSS and BCS. They can also be activated on top of different search algorithms that are designed for BSS or BCS.

Algorithm 1: NRR2 main procedure

```
1  NR-Repair(start state S)
2      OPEN ← {S}; CLOSED ← ∅; ICL ← ∅
3      while OPEN ≠ ∅ do
4          ICL ← Solve-with-NR(OPEN, CLOSED)
5          if Provable goal found then
6              Halt and return goal
7          else
8              OPEN ← OPEN ∪ ICL
9              CLOSED ← CLOSED \ ICL
10         end
11     end
12     Halt
13 end
```

4.1 NRR1: NR then AR

A simple repair mechanism is to first run NR. If it fails to return a solution within the desired bound we resort to *restarting* AR from scratch. We label it by NRR1. *Restarting* is often used in anytime algorithms for different purposes (Richter, Thayer, and Ruml 2010; Thayer and Ruml 2010). Let p be the percentage of cases where NR failed and let T_{nr} and T_{ar} denote the average running time of NR and AR, respectively. The average running time of NRR1 will be

$$T_{NRR1} = T_{nr} + p \times T_{ar}$$

NRR1 will thus be faster than AR if $p \leq 1 - \frac{T_{nr}}{T_{ar}}$.

4.2 NRR2: Incremental Repair

A more complex repair strategy is called *incremental repair* (denoted NRR2) given in Algorithm 1. In the main loop we first try to solve the problem with NR (with any relevant search algorithm that uses OPEN and CLOSED). During the execution of NR (Line 4) we keep all nodes which were considered for re-expansion (but not re-expanded due to the NR policy) in an *inconsistency list* (ICL). This list is returned to the main loop when NR terminates (Line 4). If the solution found cannot be proven to be within the bound (Line 5) we add all the nodes from ICL into the existing OPEN and delete them from CLOSED (Lines 8-9). Then, another iteration of NR is executed. This is repeatedly done until we find a solution within the desired bound. The main advantage of NRR2 is that it reuses previous information of the ICL and thus may refrain for restarting the search from scratch as NRR1.

The general idea of maintaining ICL and adding it to OPEN at a later stage was used before in the context of anytime search algorithms. It was first proposed by Likhachev et al. (2004) as part of their Anytime Repairing A* (ARA*) algorithm, where a series of WA* executions are performed with NR. After every iteration, the W parameter is decreased and the ICL is added to OPEN. Thayer and Ruml (2010) studies using this idea as one of three frameworks that allow to convert BSS algorithms into anytime search algorithms.

4.2.1 Correctness of NRR2

The correctness of NRR2 is a direct consequence of proving that the found solution is within the desired bound (Line 5). Proving this is algorithm dependent. As described above this is trivial for BCS. We now elaborate on proving this for BSS. Setting $L = f_{min}$ (as usually done for AR) is not enough when using NR and we use the following lower bound:

Let $LB = \min_{n \in \{\text{OPEN} \cup ICL\}}(g(n)+h(n))$. LB is similar to f_{min} but is defined on the union of OPEN and ICL.

Lemma 1 *For any admissible $h(n)$, at all times, LB is a lower bound on P_{opt}.*

As a result, a solution P is within the bound if $P \leq B \times LB$ and this is used in Line 5 of Algorithm 1 for BSS(B).

To prove Lemma 1 we need to first prove the following:

Lemma 2 *There always exists a node $z \in \{\text{OPEN} \cup ICL\}$ such that $g(z) = g^*(z)$ and z is on P_{opt}.*

Proof: By induction. It is true initially when z is the start state. We now prove that this property is kept when a node n is expanded. Let z be the node that holds the property before the expansion. If $n \neq z$, z still keeps the property after the expansion. If $n = z$ then one of its children z' must be on the optimal path with $g(z') = g^*(z')$. z' is either added to OPEN or, if $z' \in$ CLOSED, it is added to ICL.□

To complete the proof of Lemma 1 we note that $LB \leq f(z)$ by definition.

4.2.2 Completeness of NRR2

Lemma 3 *In a finite domain, if we continue NRR2 long enough (without halting) we will find P_{opt}.*

Proof: This is a direct consequence of Lemma 2. □

Completeness follows because P_{opt} is surely within the bound for BSS(B) for any $B \geq 1$. For BCS we can halt the algorithm at once. If $P_{opt} \leq C$ a valid solution was found. Otherwise, no solution can be found better than P_{opt}.

5 Experimental Results

WA* and PTS are simpler and easier to implement than EES and BEES. Nevertheless, Thayer et al. (2011; 2012) reported that in many cases EES and BEES outperform WA* and PTS, especially in domains with many different edge costs. Which algorithm to use is up to the user and beyond the scope of this paper. We intend to show that our repair strategies outperform AR across both settings and for all four algorithms across a number of domains.

We chose to experiment on the same variety of domains that were reported above. We only excluded the dockyard robot domain because we could not easily generate an inconsistent heuristic for this domain. The domains we chose allowed us to execute the different algorithms on all different settings on 100 instances.[5]

5.1 Repair methods for BSS(B)

Our first experiment is with WA* on the `brc202d`, `ost003d` and `den400d` maps (depicted in the leftmost column of Table 2) from the *movingai* repository (Sturtevant 2012). 100 instances were randomized and solved for each value of W. h was the *deferential heuristic* (Sturtevant et al.

[5]We refer the reader to the paper by (Ruml 2010) that discusses what sizes of domains should be used. We followed his wisdom.

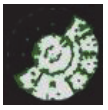
		WA*						EES					
	W	1.01	1.04	1.10	1.30	1.50	2.0	1.01	1.04	1.10	1.30	1.50	2.0
brc202d (IRE = 39.88)													
	NR	7,272	6,707	5,920	4,653	4,068	3,398	11,894	12,155	12,333	12,198	12,294	11,556
	FRt	68	43	17	4	0	0	70	64	58	23	4	0
	FRe	68	47	23	4	0	0	91	94	91	77	61	21
	AR	8,154	7,582	7,238	7,460	6,920	6,619	25,471	27,749	26,898	29,663	34,216	44,204
	NRR1	14,940	13,213	10,405	**6,606**	**4,068**	**3,398**	37,206	39,733	38,973	40,035	40,379	**29,376**
	NRR2	**7,410**	**6,792**	**5,965**	**4,664**	**4,068**	**3,398**	**12,384**	**12,729**	**13,046**	**14,304**	**15,300**	**13,703**
ost003d (IRE = 21.52)													
	NR	1,393	1,284	1,151	907	782	656	1,505	1,641	1,754	1,972	2,076	2,231
	FRt	27	9	4	1	0	0	19	9	4	0	0	0
	FRe	29	19	11	8	13	5	35	50	50	35	28	18
	AR	1,728	1,704	1,642	1,629	1,450	1,136	2,234	3,307	5,238	9,385	10,050	10,205
	NRR1	2,840	2,733	2,538	1,987	**1,420**	**786**	3,630	4,766	6,802	10,781	10,992	**9,730**
	NRR2	**1,411**	**1,298**	**1,166**	**917**	**789**	**658**	**1,722**	**2,241**	**2,718**	**2,946**	**2,673**	**2,797**
den400d (IRE = 2.71)													
	NR	1,018	959	838	716	677	648	983	957	1,013	1,013	1,133	1,378
	FRt	5	4	2	0	0	0	5	3	0	0	0	0
	FRe	7	7	7	3	0	0	9	16	13	7	1	0
	AR	1,063	1,012	977	875	761	747	1,341	1,490	1,665	1,576	1,701	2,508
	NRR1	1,427	1,376	1,245	**840**	**677**	**648**	1,812	1,945	1,948	1,786	**1,422**	**1,378**
	NRR2	**1,033**	**974**	**846**	**717**	**677**	**648**	**1,087**	**1,261**	**1,246**	**1,032**	**1,133**	**1,378**
15-puzzle with 5-5-5 PDB heuristic (IRE = 1.1)													
	NR	228,638	228,638	112,395	12,185	2,494	608	315,509	245,480	231,342	231,168	117,425	34,488
	FRr	1	1	0	0	0	0	3	0	0	0	0	0
	FRe	4	3	1	0	0	0	5	2	1	0	0	0
	AR	228,955	228,955	119,125	12,944	2,558	631	320,467	248,794	237,875	233,837	118,302	36,076
	NRR1	334,388	334,388	141,815	**12,185**	**2,494**	**608**	449,641	466,485	**233,780**	**231,168**	**117,425**	**34,488**
	NRR2	**228,672**	**228,672**	**118,152**	**12,185**	**2,494**	**608**	**318,264**	**301,847**	**231,519**	**231,168**	**117,425**	**34,488**

Table 2: Experimental results on BSS for WA* (Left) and EES (Right)

2009; Felner et al. 2011) over 10 pivots. To achieve inconsistency we used the *randomized pivot mechanism* described in (Felner et al. 2011). While there were 10 pivots, in each node we randomized one pivot and used its heuristic only. This causes an inconsistent heuristic and it was shown to significantly outperform a single fixed pivot. In addition, we implemented BPMX(1) as described by (Felner et al. 2011).

Table 2(Left) shows the average number of nodes expanded for the different policies. Columns represent different values of W. As can be seen, NR (Line 1) is always faster than AR (Line 4). The *FRt* row reports the *true failing rate* of NR, i.e., the % of instances where NR failed to return a solution within the desired bound. The *FRe* row represents the *estimated failing rate* of NRR2 which is the % of instances where the algorithm *believes* that the found solution P is not within the bound. *FRe* can be larger than *FRt* because *FRe* may have *false negatives* (as it is based on a lower bound, LB, of the optimal solution), i.e., when $LB \times B < P \leq B \times P_{opt}$. Both *FRt and FRe* can be quite large (up to 68) as can be seen in the table. The last three lines report results on the policies that are guaranteed to never fail (AR, NRR1 and NRR2); **bold** fonts are given when the repair method outperformed AR.

Consider `brc202d` (top part). NRR1 is worse than AR for small values of W where *FRe* is large. The crossing point is for $W = 1.3$ where NRR1 starts to outperform AR. By contrast, NRR2 is only slightly worse than NR across all values of W and is always much better than AR by up to a factor of 2. For $W \geq 1.5$ (as well as for other entries of other domains) *FRe* and *FRt* are 0. As a result, NR succeeds to find a solution within the bound. Nevertheless, these entries reveal the robustness of NRR1 and NRR2. In such cases, they converge to NR but unlike NR they always have the general guarantee of being within the bound.

Similar trends are reported for `den400d` and `ost003d`. Zahavi et al. (2007) defined the *inconsistency rate of* (IRE) of edge (u, v) as $|h(u) - h(v)|$. The average IRE for each domain is shown in Table 2. There is a clear correlation between IRE and FRe/FRt. They are all smaller for these two maps which have more open spaces and less corridors. Thus the negative effect of NR is less severe for these maps. For the 15-puzzle, we used the 5-5-5 PDB heuristic from (Felner, Korf, and Hanan 2004). This heuristic is only slightly inconsistent (IRE=1.1) because the blank was compressed away from the PDB. Still the same trends exit, but are more minor.

We also measured CPU time (not shown), which correlated very well with the number of nodes expanded. The reason is that there is no additional overhead for performing the repair besides the constant time of opening and expanding a node and this is reflected in the nodes count.

Similar trends are reported for EES in Table 2(Right). Here NRR2 outperforms AR by a factor of up to 4 (see $W = 2$).

5.2 Repair methods for BCS(C)

For BCS we varied the desired bound C from the minimum cost of the minimal optimal solution path (over all the 100 instances of the specific map) until the solution cost found remains when increasing C. For the maps (4-connected), h was Manhattan distance which is consistent. The results for PTS and BEES are shown in Figure 4. The x-axis corresponds to the bound C. The top curve shows the average number of expanded nodes using AR (indicated in the left y-

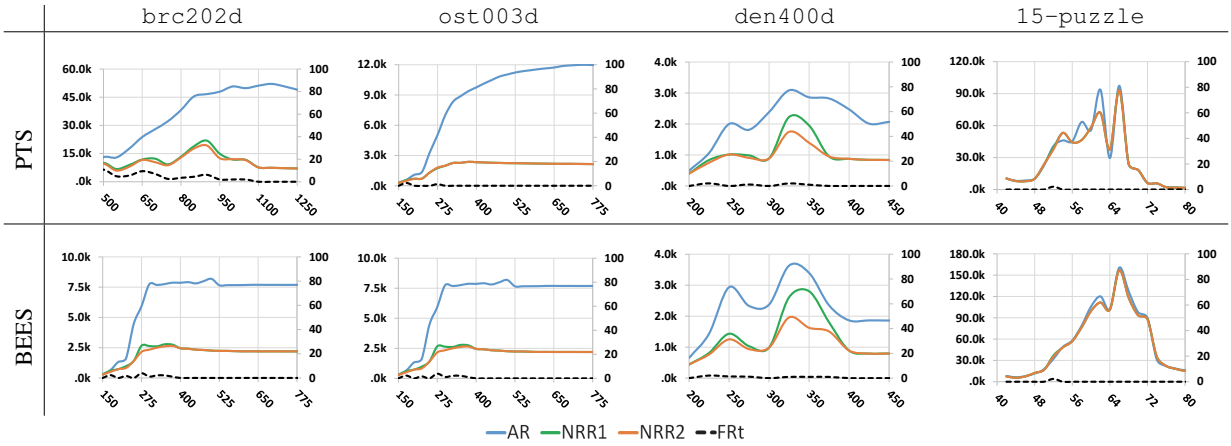


Figure 4: AR compared to NRR1 and NRR2 for PTS and BEES. Number of nodes (y-axis) as a function of C (x-axis).

axis). The dashed curve (at the bottom) shows FRt (indicated in the right y-axis). FRe is not shown because it is identical to FRt for BCS. Finally, the middle curves, show the average number of expanded nodes using NRR1 and NRR2. Unlike the case of BSS above, we did not include results of NR because here, NR may not even return a solution, i.e., OPEN may become empty. It is worth noting that both curves of NRR1 and NRR2 converge to NR exactly when FRt is 0. It can be easily seen that NRR2 outperforms AR by up a factor of up to 4 and 2 for PTS and BEES, respectively (see `ost003d` for $C = 650$).

The results on the 15-puzzle (here, with Manhattan Distance as h) are very interesting as there are a few cases where AR outperforms NRR2. The reason was explained above in Section 2. There are cases where AR is faster than NR for WA* on BSS. The results we see now, suggest that despite the fact that NR is usually faster than AR for BCS, there are some opposite cases for BCS too. In addition, for the 15-puzzle the curves for AR and NRR2 are very close to each other. This is because cycles/transpositions are more rare in this domain (the length of a cycle is at least 12 moves) than in the grid domains (length of a cycle is 4 moves or more).

For BCS the difference between NRR1 and NRR2 is small (the curves almost correlate) while for BSS they are further apart. The reason is that in BSS NR may last for a long time as it always halts with a solution, although not necessarily within the required bound. The NR phase is the main overhead for BSS (exhausting OPEN) and the extra overhead of both repair methods is small compared to that. By contrast, for BCS NR may halt fast without a solution when OPEN is empty.

6 Conclusions and Future Work

We have shown that unlike NR our repair strategies always guarantee to return a solution within the bound. They incur extra overhead over NR only when necessary. NRR2 was shown to be faster than AR on a number of domains across different settings and algorithms by a factor of up to 4. For BSS these trends are stronger when IRE is large.

Future work on repair strategies can include, for example, a light version of incremental repair (NRR2) that only performs the repair iterations up to a given constant number and then resorts to AR. Another repair strategy will start with NR and then run AR on the entire CLOSED list.

7 Acknowledgements

The research was supported by the Israeli Science Foundation (ISF) under grant #417/13 to Ariel Felner.

References

Bulitko, V.; Björnsson, Y.; Sturtevant, N. R.; and Lawrence, R. 2011. Real-time heuristic search for pathfinding in video games. In *Applied Research in Artificial Intelligence for Computer Games*. Springer.

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

Felner, A.; Korf, R. E.; and Hanan, S. 2004. Additive pattern database heuristics. *JAIR* 22:279–318.

Hansen, E. A., and Zhou, R. 2007. Anytime heuristic search. *Journal of Artificial Intelligence Research (JAIR)* 28:267–297.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* SSC-4(2):100–107.

Likhachev, M.; Gordon, G. J.; and Thrun, S. 2004. ARA*: Anytime A* with provable bounds on sub-optimality. In *NIPS*.

Malima, A., and Sabanovic, A. 2007. Motion planning and assembly for microassembly workstation. In *ISCO*, 467–474.

Nau, D.; Ghallab, M.; and Traverso, P. 2004. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc.

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

Richter, S.; Thayer, J. T.; and Ruml, W. 2010. The joy of forgetting: Faster anytime search via restarting. In *ICAPS*, 137–144.

Ruml, W. 2010. The logic of benchmarking: A case against state-of-the-art performance. In *Proceedings of the Third Annual Symposium on Combinatorial Search, SOCS 2010, Stone Mountain, Atlanta, Georgia, USA, July 8-10, 2010*.

Sepetnitsky, V.; Felner, A.; and Stern, R. 2014. Technical report: Different trends for WA*. Technical report. Available at https://www.dropbox.com/s/rako0uf1hzupvht/suplemental.pdf?dl=0.

Stern, R.; Felner, A.; van den Berg, J.; Puzis, R.; Shah, R.; and Goldberg, K. 2014. Potential-based bounded-cost search and anytime non-parametric A*. *Artificial Intelligence* 214:1–25.

Sturtevant, N.; Felner, A.; Barrer, M.; Schaeffer, J.; and Burch, N. 2009. Memory-based heuristics for explicit state spaces. In *IJCAI*.

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

Thayer, J., and Ruml, W. 2010. Anytime heuristic search: Frameworks and algorithms. In *SOCS*.

Thayer, J. T., and Ruml, W. 2011. Bounded suboptimal search: A direct approach using inadmissible estimates. In *IJCAI*.

Thayer, J. T.; Stern, R.; Felner, A.; and Ruml, W. 2012. Faster bounded-cost search using inadmissible estimates. In *ICAPS*.

Valenzano, R. A.; Sturtevant, N. R.; and Schaeffer, J. 2014. Worst-case solution quality analysis when not re-expanding nodes in best-first search. In *AAAI*, 885–892.

Zahavi, U.; Felner, A.; Schaeffer, J.; and Sturtevant, N. 2007. Inconsistent heuristics. In *AAAI*, 1211–1216.