Depth-First Memory-Limited AND/OR Search and Unsolvability in Cyclic Search Spaces

Akihiro Kishimoto , **Adi Botea** and **Radu Marinescu**
IBM Research, Ireland
{akihirok, adibotea, radu.marinescu}@ie.ibm.com

Abstract

Computing cycle-free solutions in cyclic AND/OR search spaces is an important AI problem. Previous work on optimal depth-first search strongly assumes the use of consistent heuristics, the need to keep *all* examined states in a transposition table, and the existence of solutions. We give a new theoretical analysis under relaxed assumptions where previous results no longer hold. We then present a generic approach to proving unsolvability, and apply it to RBFAOO and BLDFS, two state-of-the-art algorithms. We demonstrate the performance in domain-independent nondeterministic planning.

1 Introduction

When cost-optimal heuristic search cannot memorize the examined search space effectively, depth-first search (DFS) is often preferred to best-first search (BFS), e.g., [Korf, 1985]. Depth-first memory-limited search (DFMLS) combines DFS with a transposition table (TT). DFMLS thus caches previous search results and reuses them for the nodes revisited during search [Plaat *et al.*, 1996; Reinefeld and Marsland, 1994]. When the TT is full and new results need to be saved, existing TT entries are replaced with new entries [Nagai, 1999].

Bonet and Geffner's Bounded Learning DFS (BLDFS) algorithm (2005a; 2005b) keeps revising lowerbounds of the optimal solution by a series of depth-first searches limited by a threshold that is explicitly set at the root, until finding the optimal solution. During search, the revised lowerbounds are saved in the TT. BLDFS often outperforms BFS algorithms such as AO* [Nilsson, 1980] and CFC$_{REV^*}$ [Jiménez and Torras, 2000]. Although BLDFS can be regarded as DFMLS, its theoretical properties assume monotonic lowerbounds in the TT [Bonet and Geffner, 2005b]: the TT requires unlimited capacity with consistent heuristics in use. This assumption does not hold when the search space is larger than the TT and TT entries need to be replaced. In this case, the effect is equivalent to using an inconsistent heuristic.

RBFAOO [Kishimoto and Marinescu, 2014] is a depth-first memory-limited AND/OR search algorithm based on ideas of Recursive Best-First Search (RBFS) [Korf, 1993] with the TT. It was shown to be complete (with respect to solvable or unsolvable instances) if the search space is a finite DAG.

We focus on optimal DFMLS on finite *cyclic* AND/OR search spaces where a solution must be cycle-free, assuming an additive state model as well as *inconsistent heuristics and/or a limited size TT*. First, we outline new challenging issues that arise under these relaxed assumptions, such as returning suboptimal solutions due to the TT and cycles. These issues are not considered in the previous work of Bonet and Geffner (2005a; 2005b) and Kishimoto and Marinescu (2014). Akagi *et al.* (2010) raise similar issues but analyzed them for OR search. We then develop a generic approach to prove unsolvability with cycles, and apply it to RBFAOO and a BLDFS variant. Eriksson *et al.* (2017; 2018a; 2018b) have studied proving unsolvability, but their approach is specific to OR search.

Furthermore, we present theoretical properties of our approach and show that, if a solution exists, then an optimal solution cost can eventually be found with limited memory. In addition, our approach can also prove the unsolvability of some instances in a cyclic search space but, in some other unsolvable instances, the technique may fail due to an infinite loop. While Akagi *et al.* (2010) consider the same assumptions, their theoretical results are for IDA*+TT, which is OR search. Our new theoretical results hold for two algorithms of different behaviors in more general scenarios than those of Akagi *et al.* (2010) and Bonet and Geffner (2005a; 2005b). Moreover, as a corollary, our analysis additionally includes RBFS+TT, which is a different OR search compared to IDA*+TT.

Finally, we validate our theoretical analysis with experiments in domain-independent nondeterministic planning, showing that there are instances that require our approach to prove unsolvability, and that the ability to find optimal solution costs is preserved for solvable instances.

2 Preliminaries

We consider a state model with fully-observable states and nondeterministic actions. A tuple $\langle S, A, s_0, S_T \rangle$ defines the problem space, where S is a finite set of states, $s_o \in S$ is an initial state, A is a finite set of actions, and $S_T \subseteq S$ is a set of terminal states. Unlike Bonet and Geffner (2005a), we allow the case where S_T is empty, for instances with no solution. $A(s) \subseteq A$ denotes a set of applicable actions to a non-terminal state s. Each action $a \in A(s)$ returns a set of successor states denoted by $F_{s,a} \subseteq S$. If $F_{s,a}$ has more than one state, action a is *nondeterministic*. Otherwise, a is *deterministic*. The cost of a transition from state s with action a, denoted by $c(s,a)$, is assumed $c(s,a) > 0$ and discretized to an integer for any a and s. For simplicity, the costs for a terminal state (aka a *goal* in [Bonet and Geffner, 2005a]) and a deadend state are 0 and ∞, respectively. Our results can easily be generalized for terminal states with non-negative costs.

Similarly to Bonet and Geffner (2005a), we map this state model into an AND/OR graph G. An OR node is labeled by a state s. An AND node is labeled by the set of states reached by an action a, namely $F_{s,a}$. The root of G is an OR node labeled by the initial state s_0. The children of an OR node s are AND nodes corresponding to the applicable actions $A(s)$ in state s, and the children of an AND node $F_{s,a}$ are the OR nodes corresponding to the states in $F_{s,a}$. The edge cost from OR node s to AND node $F_{s,a}$ is $c(a,s)$, while the edge cost from an AND node to an OR node is 0.

Definition 1 (solution tree)**.** *Given an AND/OR graph G, a* solution tree *ST is a subtree of G such that: (1) the root node of ST is the initial state s_0, (2) for each internal OR node n in ST, one of n's children is in ST, (3) for each internal AND node n in ST, all of n's children are in ST, and (4) all tip nodes in ST are terminal nodes (goals).*

While Bonet and Geffner (2005a) introduce BLDFS for the *max state model* (MAX), we consider here the *additive state model* (ADD) of Bonet and Geffner (2005a) where the cost of a solution tree ST is defined as the sum of the edge costs in ST. An optimal solution has a minimal cost.

We can support the MAX with small changes. Yet, focusing on the ADD in our setting is essential, because RBFAOO and BLDFS face new challenges discussed in Section 3.

Each node n in ST is associated with a value $V(n)$ capturing the optimal solution cost of the subtree rooted at n. $V(n)$ can be computed recursively based on the values of n's successors: OR nodes by minimization, AND nodes by summation. The ADD model has applications such as the (dis)assembly domain [Jiménez and Torras, 2000] and web service composition [Liang and Su, 2005]. Chemical synthesis planning [Heifets and Jurisica, 2012] is a remarkable example, when the objective is minimizing the reaction cost.

Path $p = s_0, a_0 \rightarrow \cdots \rightarrow s_{k-1}, a_{k-1} \rightarrow s_k$ indicates a path from the initial state s_0 to an OR node s_k by a sequence of actions $a_0, \cdots, a_{k-1}$. Path $p = s_0, a_0 \rightarrow \cdots \rightarrow s_k, a_k$ leads to an AND node F_{s_k,a_k} from s_0 by a sequence of actions $a_0, \cdots, a_k$. Path $p + a$ indicates a path that extends p with action a leading to an AND node $F_{n,a}$, where n is an OR node reached by path p. Analogously, $p + s$ extends path p by adding an OR child of an AND node reached by path p.

A *cyclic* path has $s_i = s_j$ for some $i \neq j$. The search space can contain cyclic paths, but a solution tree has no cyclic path. If no cycle-free solution tree exists, the solution cost is ∞.

Since we solve a minimization task, the algorithms we present attempt to improve a lowerbound L of the optimal solution cost C^*.

When $L = C^*$, the solution is proven optimal. To prove unsolvability, it is sufficient to ensure that $L > U$, where U is an upperbound of the optimal solution. While computing good U values is an important topic of research, in this paper, we assume that such U cannot be calculated, and the algorithms need to explicitly prove $L = \infty$.

We say that an algorithm has an *infinite loop* if it runs forever without finding C^* or proving $L = \infty$. We assume that $\infty - k < \infty$ holds, where k is a small number. In the actual implementation, a large finite number is often used to represent ∞. We assume that $V(n) \ll \infty - k$ holds if an optimal solution exists, and that no algorithm can gradually increase L to $\infty - k$ in finite time if no solution exists.

RBFAOO and BLDFS

These algorithms maintain at each node n a lowerbound $q(n)$ (called a *q-value*) on $V(n)$. During search, they improve and cache in the TT $q(n)$ which is calculated by backing up the q-values of n's children, until proving $q(r) = V(r)$ at the root r or $q(r) = \infty$ (i.e., no solution). For an admissible heuristic h, $q(n)$ is computed as:

1. $q(n) = 0$, if n is a terminal OR node.
2. $q(n) = h(n)$, if n is a non-terminal tip node.
3. $q(n) = \min_{a \in A(n)}(c(n,a) + q(F_{n,a}))$, if n is an internal OR node and $F_{n,a}$ is an AND child of n.
4. $q(F_{n,a}) = \sum_{s \in F_{n,a}} q(s)$, if $F_{n,a}$ is an internal AND node and s is an OR child of $F_{n,a}$.

Let $th(n)$ be a threshold at node n. Both algorithms examine a subtree rooted at node n in a depth-first manner until $th(n) < q(n)$ holds or n is solved optimally (*termination condition*). RBFAOO employs a *local threshold* controlling mechanism that initially sets $th(r) = \infty - 1$ at the root r. It updates $q(n)$ at node n using q-values of n's children. If a termination condition holds, it backtracks to n's parent. Otherwise, it selects a child to examine further. For $q'(F_{n,a}) = c(n,a) + q(F_{n,a})$, let $q'(F_{n,a_1})$ and $q'(F_{n,a_2})$ be the smallest and second smallest values among the list of values of n's children. If n has only one child, $q'(F_{n,a_2}) = \infty$. RBFAOO selects a child as follows:

- At an OR node n, select F_{n,a_1} with a new threshold $th(F_{n,a_1}) = \min(th(n), q'(F_{n,a_2})) - c(n,a_1)$.
- At an AND node $F_{n,a}$, select any unsolved child s with a new threshold $th(s) = th(F_{n,a}) - q(F_{n,a}) + q(s)$.

BLDFS employs a *global threshold* controlling scheme that gradually increases the threshold at the root. In each iteration, it sets $th(r) = q(r)$ at the root r. At an OR node n, if a child $F_{n,a}$ holds $q'(F_{n,a}) \leq th(n)$, it examines $F_{n,a}$ with $th(F_{n,a}) = th(n) - c(n,a)$. At an AND node, it updates the threshold in the same way as RBFAOO, which is also described in [Bonet and Geffner, 2005a].

3 Challenges under Relaxed Assumptions

In the ADD model with a consistent heuristic and unlimited TT capacity, BLDFS can always return an optimal solution in finite time if one exists in the finite cyclic search space. This property is based on the fact that BLDFS does not need to generate an action leading to a repeated state, since it never enters into a cycle [Bonet and Geffner, 2005b]. However, when using an inconsistent heuristic, even with an unlimited

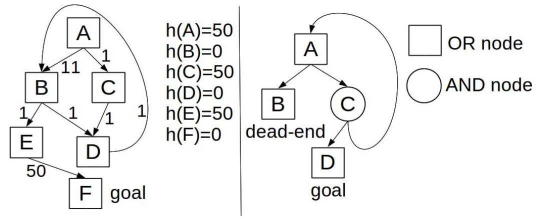

Figure 1: Left: example where BLDFS and RBFAOO enter into a cycle and return a suboptimal solution; Right: example that cannot be proven unsolvable if cycles are unfolded.

TT, BLDFS may enter into a cycle. If the TT is updated without carefully considering that repeated state, BLDFS may lead to a suboptimal solution.

Example with Inconsistent Heuristic

In the example shown in Figure 1(left), both BLDFS and RBFAOO enter into a cycle when using an admissible, inconsistent heuristic whose values are shown in the figure. The optimal solution opt is $A \rightarrow C \rightarrow D \rightarrow B \rightarrow E \rightarrow F$ with solution cost 54. BLDFS sets $th(A) = h(A) = 50$. If it does not detect a repeated state, $th(A)$ is large enough to reach B again via path $p_1 = A \rightarrow B \rightarrow D \rightarrow B$. In case of RBFAOO, $th(B) = q'(C) - c(A, A \rightarrow B) = 40$. As in the case of BLDFS, $th(B)$ is large enough to revisit B via p_1.

Now assume that BLDFS generates *no* action forming a cycle. Then, it incorrectly returns a suboptimal solution $sopt = A \rightarrow B \rightarrow E \rightarrow F$ with cost 62, since it searches in the following order: (1) Reach D via $p_2 = A \rightarrow B \rightarrow D$; (2) No action is available at D, since $D \rightarrow B$ is *not* generated since a cycle is formed. The value of ∞ (i.e., deadend) is mistakenly saved in the TT for D; (3) Reach C via $A \rightarrow C$, and backup the value of ∞ of D to C; (4) The only reachable path to a goal is now $sopt$. RBFAOO suffers from the same issue. It reaches D via p_2 first and does not generate $D \rightarrow B$.

The pathmax heuristic does not resolve the issue in our example, since it is still an inconsistent heuristic [Holte, 2010].

In the ADD model, Bonet and Geffner (2005b) ensure an algorithmic equivalence between LDFS with no explicit threshold (i.e., use $q(n)$ as a threshold for node n) and BLDFS with an explicit threshold. However, the equivalence no longer holds in our relaxed setting. For example, assume that an OR node n has one AND node m that has two successors s_1 and s_2, where $h(n) > c(n, m) + h(s_1) + h(s_2)$ due to an inconsistent heuristic. Then, LDFS has an issue of setting up a threshold that is large enough to strictly improve $q(n)$ or solve n after LDFS finishes examining s_1 or s_2. Hence, the explicit threshold of BLDFS is necessary even in the ADD.

Example with Consistent Heuristic and Limited-Size TTs

If existing TT entries are overwritten with the lowerbounds of new nodes, the heuristic estimates in the TT become *inconsistent*. This is equivalent to using an inconsistent heuristic.

We show an example only for RBFAOO, since an example for BLDFS can be constructed analogously. The heuristic shown in Figure 1(left) becomes consistent by replacing $h(A) = h(C) = 50$ with $h(A) = h(C) = 0$. In this case, RBFAOO first reaches D via $p_3 = A \rightarrow C \rightarrow D \rightarrow B$, and only generates action $B \rightarrow E$ (i.e., $B \rightarrow D$ forms a cycle). Then, RBFAOO backtracks to A and stores improved q-values, i.e., $q(B) = 51$, $q(D) = 52$, and $q(C) = 53$, based on $h(E) = 50$. Next, it examines B. Assume now that $q(B)$ and $q(D)$ are removed from the TT before B is examined. Then, it reaches D via p_2, since $th(B) = q'(C) - c(A, A \rightarrow B) = 43$ is large enough. It stores the value of ∞ in the TT entry of D since $D \rightarrow B$ forms a cycle. If the TT never removes D, RBFAOO cannot reach a goal via opt.

One straightforward way to be able to return an optimal solution with Figure 1 (left) is to perform so-called unfolding cycles (keep examining a node), even if it encounters a repeated state. While this approach returns an optimal solution if one exists, it cannot prove unsolvability of a problem instance that involves cycles such as Figure 1(right).

4 Boosting Ability to Prove Unsolvability

We introduce a generic approach to proving unsolvability, even for limited TTs and inconsistent heuristics, and combine it with RBFAOO and BLDFS with an ADD model. The heuristic h must be admissible, but can be inconsistent.

Generic Principles

The suboptimal solution returned in the example from Section 3 is caused by saving in the TT a value of ∞ calculated by a cycle which holds only for that path. We bypass this incorrectness by extending TTs to separately save two types of values: q-values and *r-values*.

Since the q-value of a node n, $q(n)$, described in Section 2, is a path-independent lowerbound on the node's cost, it is retrieved at n via any path. An r-value for a node n reached via a path p, $r(n, p)$, is a path-specific lowerbound. The value $r(n, p)$ is retrieved only when n is reached by path p. Since r-values take into account that a repeated state is a deadend, $r(n, p)$ is computed as follows:

1. $r(n, p) = 0$, if n is a terminal OR node and p does not form a cycle.
2. $r(n, p) = \infty$, if n is an OR node and p forms a cycle.
3. $r(n, p) = h(n)$, if n is a non-terminal tip node.
4. $r(n, p) = \min_{a \in A(n)} (c(n, a) + r(F_{n,a}, p + a))$, if n is an internal OR node and $F_{n,a}$ is an AND child of n.
5. $r(F_{n,a}, p) = \sum_{s \in F_{n,a}} r(s, p + s)$, if $F_{n,a}$ is an internal AND node and s is an OR child of $F_{n,a}$.

When a search reaches a node n via a path p, and further explores the space rooted at n, it uses a threshold to control the search underneath. Both $q(n)$ and $r(n, p)$ get updated based on the search results. Because of the definition of the state model, a repeated node is always an OR node. If the search expands n and generates a child ch that belongs to path p, then ch is a repeated node. It is a deadend along this path, and not expanded further along this path. When updating $q(n)$ and $r(n, p)$, we set $r(ch, p + a + ch) = \infty$. However, computing $q(n)$ uses $q(ch)$, as shown in Section 2, as q is a path-independent estimation.

When n is reached via a new path p', $q(n)$ is used to reduce duplicate search effort. In addition, since $q(n)$ is more conservative than $r(n, p')$, we set $r(n, p') = q(n)$ if $r(n, p') <$

Algorithm 1 RBFAOO$_r$: RBFAOO with our approach

```
Require: Root node $root$
 1: $root.th = \infty - 1$
 2: $\boldsymbol{r}$ = RBFAOOSearch($root$)
 3: return $\boldsymbol{r}$
```

$q(n)$. This update is not uncommon, since $r(n, p') = h(n)$ when n is reached via p' for the first time.

Threshold for r-values. Typical DFMLS updates thresholds based on the q-value. In contrast, we adjust a threshold based on the r-value. The node to examine is selected by the smallest r-value at each OR node, and the threshold is updated to observe an improvement of the r-value as an indication of search progress. While Akagi *et al.* (2010) use this idea, they apply it only to IDA*+TT, and do not store r-values in the TT.

Reuse of proof trees. If a node n is optimally solved, then typical DFMLS marks n as solved and saves it in the TT. When n is reached via any path, n is regarded as solved without re-examining the search space rooted at n. However, care must be taken in this approach to ensure that no *implicit cycles* are created [Campbell, 1985; Kishimoto and Müller, 2004]. For example, assume a node n is saved in the TT as solved, and n's solution tree has a node c. If c is removed from the TT, n is reached by a path p, and p contains c, then regarding n via p as solved is not correct due to a cyclic path $c \cdots \to n \cdots \to c$. We avoid implicit cycles by checking both q-values and r-values when marking a node as solved. Assume that a node n via path p is solved or proven a deadend. If $q(n) = r(n, p)$, the TT entry for $q(n)$ marks the flag as solved. This is used as a path-independent solution. If $q(n) \neq r(n, p)$, the TT entry for $r(n, p)$ marks the flag as solved, which is reused only for n via path p. Theorem 4 in Section 5 ensures the correctness.

Space saving in the TT. The TT is typically implemented as a hash table, and has a fixed number of entries to store information on nodes. Compared to the TTs which take only q-values, our approach manages q- and r-values of each node. To save the TT entries, if $r(n, p) = q(n)$, only $q(n)$ is saved. Otherwise, in addition to the TT entry for $q(n)$, another TT entry is used to save $r(n, p)$. When $r(n, p)$ is retrieved but no information is available except $q(n)$, $q(n)$ is used for $r(n, p)$.

Combination with RBFAOO

Algorithms 1-4 show a combination of our approach with RBFAOO, denoted by RBFAOO$_r$. We omit the overestimation technique [Kishimoto and Marinescu, 2014], as automatically setting an efficient overestimation parameter in cyclic search spaces remains open. The essential differences from original RBFAOO, using the r-values instead of q-values, are highlighted in bold. Algorithm 2 focuses on searching for a solution. Method FromTT (Algorithms 3 and 4) retrieves from TT: $r(n, p)$ where p is a path to reach n, $q(n)$, and a flag whether n is solved. If p forms a cycle due to n, then set $r(n, p) = \infty$. If the TT has no entry for $q(n)$ or $r(n, p)$, use $h(n)$. Also, FromTT returns $q(n)$ for $r(n, p)$ if $r(n, p) < q(n)$.

Node n is a 3-field tuple: a threshold th, a flag *slvd* and a path $path$ to reach n. The flag is true if n is solved or proven

Algorithm 2 RBFAOOSearch with our approach

```
Require: Node $n$
 1: if (HasNoChildren($n$)) then
 2:     $q$ = Evaluate($n$) //Terminal node/deadend
 3:     SaveInTT($n$, $\boldsymbol{q}$, $q$) //Store search result
 4:     return $\boldsymbol{q}$
 5: GenerateChildren(n)
 6: if ($n$ is an OR node) then
 7:     loop
 8:         $(F_{n,a}, \boldsymbol{r}, \boldsymbol{r_2}, q)$ = BestChild($n$)
 9:         SaveInTT$(n, \boldsymbol{r}, q)$
10:         if $(\boldsymbol{n.th} < \boldsymbol{r} \vee n.slvd)$ then
11:             break
12:         $\boldsymbol{F_{n,a}.th = \min(n.th, r_2) - c(n,a)}$;
            $F_{n,a}.path = n.path + a$
13:         RBFAOOSearch($F_{n,a}$)
14: else
15:     loop
16:         $(\boldsymbol{r}, q)$ = Sum($n$) //$n$ is an AND node
17:         SaveInTT$(n, \boldsymbol{r}, q)$
18:         if $(\boldsymbol{n.th} < \boldsymbol{r} \vee n.slvd)$ then
19:             break
20:         $(s, \boldsymbol{r_s})$ = UnsolvedChild($n$)
21:         $\boldsymbol{s.th = n.th - (r - r_s)}$; $s.path = n.path + s$
22:         RBFAOOSearch($s$)
23: return $\boldsymbol{r}$
```

Algorithm 3 BestChild

```
Require: Node $n$
 1: $n.slvd = \perp$ ($\perp$ stands for false)
 2: $\boldsymbol{r = r_2} = q = \infty$; $ch_{best}$ = undefined
 3: for (each child $F_{n,a_i}$ of $n$) do
 4:     $(\boldsymbol{r_{ch_i}}, q_{ch_i}, f_{ch_i})$ = FromTT($F_{n,a_i}$)
 5:     $q_{ch_i} = c(n,a_i) + q_{ch_i}$; $\boldsymbol{r_{ch_i} = c(n,a_i) + r_{ch_i}}$; $q =$
        $\min(q, q_{ch_i})$
 6:     if $(\boldsymbol{r_{ch_i} < r} \vee (\boldsymbol{r = r_{ch_i}} \wedge \neg n.slvd))$ then
 7:         $\boldsymbol{r_2 = r}$; $\boldsymbol{r = r_{ch_i}}$; $ch_{best} = F_{n,a_i}$; $n.slvd = f_{ch_i}$
 8:     else if $(\boldsymbol{r_{ch_i} < r_2})$ then
 9:         $r_2 = r_{ch_i}$
10: return $(ch_{best}, \boldsymbol{r}, \boldsymbol{r_2}, q)$
```

unsolvable. A node n also has state information, to enable to retrieve $q(n)$ from the TT. SaveInCache saves in the TT an $r(n, p)$, $q(n)$, and whether n via p is solved.

Examining a node n includes a check whether n has children or not. If not (lines 1–4 in Algorithm 2), the Evaluate method sets $n.slvd = \top$ and checks if n is a terminal or a path-independent deadend (i.e., deadend unrelated to cycles). The q-value is set to 0 for a terminal node and to ∞ for a deadend, and so is the r-value. These values are saved in the TT. Lines 7–13 and 15–22 show respectively the cases when n is an OR node or an AND node, which have similar steps:

- Update the q-value and r-value for n, based on the values of the children (see lines 8 and 16, and Algorithms 3–4).
- Perform the backtracking test (lines 10–11 and 18–19). RBFAOO$_r$ backtracks to n's parent if either $n.th < r(n, p)$ or $n.slvd = \top$ holds.
- Select a child s (lines 8 and 20) otherwise. At OR nodes, $s = F_{n,a}$ is the child with the smallest r-value among unsolved children (method BestChild). At AND nodes,

Algorithm 4 Sum

```
Require: Node $n$
1: $n.slvd = \top$ ($\top$ stands for true)
2: $\boldsymbol{r} = q = 0$
3: for (each child $ch_i$ of $n$) do
4:     $(\boldsymbol{r_{ch_i}}, q_{ch_i}, f_{ch_i}) = \text{FromTT}(ch_i)$
5:     $q = q + q_{ch_i}$; $\boldsymbol{r = r + r_{ch_i}}$; $n.slvd = n.slvd \wedge f_{ch_i}$
6: $n.slvd = n.slvd \vee \boldsymbol{r = \infty}$
7: return $(\boldsymbol{r}, q)$
```

Algorithm 5 BLDFS$_r$: BLDFS variant with our approach

```
Require: Root node $root$
1: $root.slvd = \bot$; $\boldsymbol{root.r} = h(n)$
2: while $\neg root.slvd$ do
3:     $\boldsymbol{root.th = root.r}$
4:     BLDFSDriver($root$)
5: return $\boldsymbol{root.r}$
```

choose any unsolved child s. Then, update $th(s)$ (lines 12 and 21), and recursively examine s (lines 13 and 22).

A small yet efficient change to the original RBFAOO is to store $q(n)$ in the TT, immediately after it is updated (lines 9 and 17). This is necessary to quickly back-propagate a larger q-value involving repeated states. Bonet and Geffner (2006) use a similar idea for an MDP version of LDFS.

RBFAOO$_r$ correctly handles the example shown in Figure 1 left. RBFAOO$_r$ reaches B, via $A \to B$, updates $q(B) = \min(1 + h(E), 1 + h(D)) = 1$, then reaches D via $p_1 = A \to B \to D$. It generates a cycle $p_1 + B$, and stores $r(D, p_1) = \infty$ due to the cycle. On the other hand, it sets $q(D) = 1 + q(B) = 2$. It updates $r(B, A \to B) = \min(1 + r(D, p_1), 1 + r(E, A \to B \to E)) = 51$, and $q(B) = \min(1 + q(D), 1 + q(E)) = 3$. It then selects C and reaches D via $p_2 = A \to C \to D$. Since $r(D, p_2) = 4$, which is different from $r(D, p_1) = \infty$, it does not regard D via this path as unsolved. Therefore, it reaches B via $p_2 + B$, eventually leading to optimal solution $p_2 + B + E + F$.

Combination with Bounded LDFS Variant

BLDFS with our approach, denoted BLDFS$_r$, controls its threshold by r-values, based on the approach of Bonet and Geffner (2005a) for the ADD model. While RBFAOO$_r$ and BLDFS$_r$ share the same principles, BLDFS$_r$ examines the space with a global threshold controlling scheme.

Let $th(n, p)$ be a threshold for a node n reached via path p, and rt be the root. In each iteration, BLDFS$_r$ starts examining the search space with $th(rt, rt) = r(rt, rt)$, and examines n via p as long as $r(n, p) \leq th(n, p)$. At the start of search, $th(rt, rt) = h(rt)$, since $r(rt, rt) = h(rt)$.

At each internal node, BLDFS$_r$ selects a child in the same way as described in Section 2 except that it uses the r-values:

- At an OR node n via path p, for a child $F_{n,a}$ via path $p_F = p + a$, if $c(n, a) + r(F_{n,a}, p_F) \leq th(n, p)$ holds, examine $F_{n,a}$ with $th(F_{n,a}, p_F) = th(n, p) - c(n, a)$.
- At an AND node $F_{n,a}$ via path p_F, select any unsolved child s with a new threshold $th(s, p_s) = th(F_{n,a}, p_F) - r(F_{n,a}, p_F) + r(s, p_s)$, where $p_s = p_F + s$.

Algorithm 6 BLDFS Driver with our approach

```
Require: Node $n$
1: if (HasNoChildren($n$)) then
2:     $q$ = Evaluate($n$) //Terminal node/deadend
3:     SaveInTT($n$, $\boldsymbol{q}$, $q$) //Store search results
4:     $n.\boldsymbol{r} = n.q = q$; $n.slvd = \top$
5:     return
6: GenerateChildren(n)
7: if ($n$ is an OR node) then
8:     $slvd = \bot$; $\boldsymbol{r} = q = \infty$
9:     for (each child $ch_i = F_{n,a_i}$) do
10:        $(\boldsymbol{r_{ch_i}}, q_{ch_i}, f_{ch_i}) = \text{FromTT}(ch_i)$
11:        $q_{ch_i} = c(n, a_i) + q_{ch_i}$; $\boldsymbol{r_{ch_i} = c(n, a_i) + r_{ch_i}}$
12:        if ($\boldsymbol{n.th < r_{ch_i}}$) then
13:            $q = \min(q, q_{ch_i})$; $\boldsymbol{r = \min(r, r_{ch_i})}$
14:        else if ($f_{ch_i}$) then
15:            $\boldsymbol{slvd = \top}$; //Solution within the threshold
16:            $q = \min(q, q_{ch_i})$; $\boldsymbol{r = r_{ch_i}}$
17:            //Update the threshold with this upperbound
18:            $\boldsymbol{n.th = r_{ch_i} - 1}$
19:        else
20:            $ch_i.th = n.th - c(n, a_i)$
21:            $ch_i.path = n.path + a_i$
22:            BLDFSDriver($ch_i$)
23:            $q = \min(q, c(n, a_i) + ch_i.q)$;
24:            $\boldsymbol{r = \min(r, c(n, a_i) + ch_i.r)}$
25:            if ($\boldsymbol{ch_i.slvd \wedge ch_i.r \leq ch_i.th}$) then
26:                $\boldsymbol{slvd = \top}$; //Solution within the threshold
27:                //Update the threshold with this upperbound
28:                $\boldsymbol{n.th = c(n, a_i) + ch_i.r - 1}$
29:    $\boldsymbol{n.r = r}$; $n.q = q$; $n.slvd = slvd \vee \boldsymbol{n.r = \infty}$
30: else
31:    //$n$ is an AND node
32:    for (each child $ch_i$) do
33:        $(\boldsymbol{r}, q) = \text{Sum}(n, ch_i)$
34:        if ($\boldsymbol{n.th < r}$) then
35:            break
36:        if ($\neg ch_i.slvd$) then
37:            $\boldsymbol{ch_i.th = n.th - (r - ch_i.r)}$;
38:            $ch_i.path = n.path + ch_i$
39:            BLDFSDriver($ch_i$)
40: SaveInTT($n$, $\boldsymbol{r}$, $q$)
```

The q- and r- values are back-propagated to the root in the same way as RBFAOO$_r$. For theoretical analysis, we modify slightly the original BLDFS. If a solution is found at node n, our BLDFS variant delays to back up the solved flag of $\top$ to n's parent, until the solution cost at n is proven optimal within the threshold. It is when BLDFS marks the solved flag at the root as solved that the optimal solution cost is proven at the root. Theoretical analysis with the original BLDFS under our relaxed assumptions is left as future work.

Algorithms 5–6 show the pseudocode of BLDFS$_r$, which is a combination of BLDFS with our approach. We use notations that are familiar in the heuristic search community. To explain in detail how r-values and q-values are managed, our pseudocode is represented in a more detail than in the work of Bonet and Geffner (2005a; 2005b). The essential differences from original BLDFS are highlighted in bold. Node n includes the threshold th, called the *bound* in [Bonet and Geffner, 2005a]. In addition, n contains r and q, an r-value and a q-value stored after n is examined, and a path $path$ to n

from the root. The Sum method in Algorithm 4 is extended to receive a child s_i and sets $s_i.slvd$, indicating if s_i is marked as solved in the TT. Sum also updates $n.r$, $n.q$ and $n.slvd$.

5 Theoretical Properties

We define the notion of memory requirement linear in the search depth and the branching factor along the path.

Definition 2. *Given a search problem and an amount of memory available, we say that the* reasonable memory requirement *is satisfied if, for any path that could be explored in the search, the memory is sufficient to store all nodes along the current paths, together with all their siblings.*

If b is the maximum branching factor, and d is the maximum depth of a path explored in the problem at hand, we need at least $O(bd)$ memory, as discussed in [Korf, 1993].

We prove theoretical properties about our approach combined with RBFAOO and our BLDFS variant that require only reasonable memory, denoted by RBFAOO$_r$ and BLDFS$_r$, respectively. Our results ensure that our approach can be enhanced with a TT that has any strategies to replace the TT entries of any nodes previously examined, unless these TT entries are for the nodes/their siblings of the path currently examined. For example, assume that a search path p_1 changes to a new one p_2. Let n be a node on p_1 that is neither on p_2 nor any of the siblings of the nodes on p_2. Then, the q- and r-values for n can be safely removed from the TT.

The reasonable memory requirement is close to the theoretically minimum memory requirement to return an optimal solution *cost*. Note that the size of the solution tree is needed as a memory requirement to return a solution tree.

Let $C(n,p)$ be an optimal solution cost for node n via path p, $\mathrm{T}_q(n)$ be $q(n)$ in the TT, and $\mathrm{T}_r(n,p)$ be $r(n,p)$ in the TT. If no result is saved, $\mathrm{T}_q(n) = h(n)$ where h is an admissible heuristic. So is $\mathrm{T}_r(n,p)$, but if p forms a cycle, then $\mathrm{T}_r(n,p) = \infty$. Since $\mathrm{T}_r(n,p)$ is increased to $\mathrm{T}_q(n)$ as discussed in Section 4, clearly $\mathrm{T}_q(n) \leq \mathrm{T}_r(n,p)$ holds.

Lemma 1. *$T_r(n,p)$ always has admissible values for any path p leading to node n, i.e., $T_r(n,p) \leq C(n,p)$.*

Theorem 2 (Correctness)**.** *The solution cost returned by RBFAOO$_r$/BLDFS$_r$ is always optimal.*

Lemma 3. *Assume that RBFAOO$_r$/BLDFS$_r$ marks a node n as always solved, i.e., n via path p is solved and $T_q(n) = T_r(n,p)$ holds. Then, $T_q(n) = C(n,n)$ holds, where $C(n,n)$ is the optimal solution cost of the subtree rooted at n.*

Theorem 4 guarantees that an optimal solution cost at n can be transposed from one path to another (see Section 4).

Theorem 4. *Assume that node n via path p is solved and $T_q(n) = T_r(n,p)$. Then, marking n as solved via any path does not affect the optimal solution cost.*

Proof. Let s_0 be the root. We prove the theorem by showing that a node n marked as always solved is not a part of the optimal solution cost at s_0, if $\mathrm{T}_q(n)$ causes an *implicit* cycle. In this case, $\mathrm{T}_q(n)$ merely prunes away duplicate search rooted at n, even if marked as solved.

Assume that n via p is marked as always solved. Let ST be the (optimal) solution tree rooted at n and $\mathrm{T}_q(n) = C(n,n)$. By Lemma 3 ST has no cyclic path, since the solution cost at each internal OR node m in ST is $C(m,m)$ and $c(m,a) > 0$.

Let $Q(n,p)$ be the set of OR nodes on the non-cyclic path p from s_0 to node n, and $P(n,p) = Q(n,p)\backslash\{n\}$. The correctness of the theorem is ensured as follows: (1) If $s \notin ST$ for every $s \in P(n,p)$, $C(n,p) = C(n,n)$. No cycle is created from s_0 to reach any node in ST. (2) Otherwise, a node $s \in P(n,p)$ may be in ST. That is, ST cannot be used to calculate $C(n,p)$, since s creates a cycle. Let $m \in P(n,p)$ be the closest node to s_0 on p which creates a cycle in ST. Let p_m be the prefix of p from s_0 to m. Since m is included in ST and $c(n,a) > 0$ for $a \in A(n)$, $C(m,m) < \mathrm{T}_q(n) = C(n,n)$ holds. In addition, no node in $P(m,p_m)$ creates a cycle in ST. Therefore, $C(m,p_m) = C(m,m) < \mathrm{T}_q(n)$, indicating that $\mathrm{T}_q(n)$ is never a part of the optimal solution at s_0. □

We prove the admissibility of BLDFS$_r$ with more relaxed assumptions than [Bonet and Geffner, 2005b].

Lemma 5. *If Algorithm 6 (BLDFSDriver) starts with $s_0.th = b \ll \infty$ at the root s_0, then it terminates in finite time with either $s_0.r > b$ or $s_0.slvd = \top$.*

Theorem 6 (Admissibility)**.** *BLDFS$_r$ finds an optimal solution cost in finite time, if one exists in the finite search space.*

Proof sketch. If $s_0.th \geq C(s_0,s_0)$, BLDFSDriver returns an optimal solution cost $C(s_0,s_0)$ in finite time, since: (1) Theorems 2 and 4 ensure solution optimality. (2) Each node n in the optimal solution tree is examined, which is easily proven by induction on the number of chains of the recursive calls.

If $s.th < C(s_0,s_0)$, by Lemma 5, BLDFSDriver strictly increases $s_0.th$ in finite time, since BLDFSDriver cannot mark s_0 as solved (see line 12 in BLDFSDriver). Since $C(s_0,s_0) \ll \infty$, in a finite number of iterations, BLDFS$_r$ sets $s_0.th = b \geq C(s_0,s_0)$, enabling to optimally solve any solvable instance in finite time. □

The results in [Kishimoto and Marinescu, 2014] hold only for finite DAGs (i.e., no cycles involved). On the other hand, RBFAOO$_r$ focuses on finite cyclic spaces.

Lemma 7. *Assume that RBFAOO$_r$ examines a node n via path p with $th(n) = b \ll \infty$. Then, RBFAOO$_r$ backtracks to n in finite time with either $T_r(n,p) > b$ and/or n is marked as solved.*

Theorem 8 (Admissibility)**.** *RBFAOO$_r$ finds an optimal solution cost in finite time, if one exists in the finite search space.*

Proof sketch. Let s_0 be the root and ST be an optimal solution tree rooted at s_0. If no path exists which makes RBFAOO$_r$ deviate from ST (i.e., ST is the whole search space), since $C(s_0,s_0) \ll th(s_0) = \infty - 1$, RBFAOO$_r$ finds an optimal solution cost in finite time. This is easily verified by the fact that (1) RBFAOO$_r$ requires only reasonable memory and (2) $\mathrm{T}_r(n,p) \leq C(n,p) \ll th(n) = \infty - k$ for each $n \in ST$, where k is a small number. This $th(n)$ is large enough to cover all terminal nodes in ST. Otherwise, let n via path p be the first OR node in ST at which RBFAOO$_r$ generates two children F_{n,a_1} and F_{n,a_2}, where $F_{n,a_1} \in ST$ but $F_{n,a_2} \notin ST$. Here, we show only the case with two children, since more general cases are proven in an analogous way. F_{n,a_2} may or may not lead to another optimal solution, but because of the optimality of ST, $C(n,p) = c(n,a_1) + C(F_{n,a_1}, p_{a_1}) \leq c(n,a_2) + C(F_{n,a_2}, p_{a_2})$ holds, where $p_a = p+a$. RBFAOO$_r$ reaches n with $th(n) = \infty - k$ where k is a small number.

$\mathrm{T}_r(F_{n,a_2}) = \infty$ means that RBFAOO$_r$ cannot deviate from ST. Hence, we consider the case where $\mathrm{T}_r(F_{n,a_2}) \ll \infty - k$, i.e., $c(n,a_2) + \mathrm{T}_r(F_{n,a_2}) \ll \infty - k$. Since F_{n,a_1} has an optimal solution, $c(n,a_1) + \mathrm{T}_r(F_{n,a_1}, p_{a_1}) \leq C(n,p) \ll \infty - k$ holds (see Lemma 1 as well). Therefore, the threshold t set to examine F_{n,a_1} and F_{n,a_2} always satisfies $t \ll \infty - k$.

If RBFAOO$_r$ examines F_{n,a_1} with $t < C(n,p) - c(n,a_1) = C(F_{n,a_1}, p_{a_1})$, by Lemmas 7, RBFAOO$_r$ backtracks to n in finite time. It holds that $\mathrm{T}_r(F_{n,a_1}, p_{a_1})$ is strictly increased, or/and F_{n,a_1} is solved. If F_{n,a_1} is solved but $v = c(n,a_1)+C(F_{n,a_1}, p_{a_1}) > c(n,a_2)+\mathrm{T}_r(F_{n,a_2}, p_{a_2})$ still holds, F_{n,a_2} is examined with $th(F_{n,a_2}) = v - c(n,a_2) \ll \infty$. By Lemma 7, this step completes in finite time, proving that a_1 leads to an optimal solution cost.

Analogously, if RBFAOO$_r$ examines F_{n,a_2} with $t < C(n,p) - c(n,a_2)$, by Lemma 7, RBFAOO$_r$ backtracks to n in finite time and $\mathrm{T}_r(F_{n,a_2}, p_{a_2})$ is strictly increased, or/and F_{n,a_2} is solved. If F_{n,a_2} is solved, $\mathrm{T}_r(F_{n,a_2}, p_{a_2}) = C(F_{n,a_2}, p_{a_2}) \geq C(n,p) - c(n,a_2)$.

Since RBFAOO$_r$ requires only reasonable memory, it remembers $\mathrm{T}_r(F_{n,a_1}, p_{a_1})$ and $\mathrm{T}_r(F_{n,a_2}, p_{a_2})$. By Lemmas 1 and 7, even if F_{n,a_1} remains unsolved, RBFAOO$_r$ strictly increases $\mathrm{T}_r(F_{n,a_1}, p_{a_1})$ and $\mathrm{T}_r(F_{n,a_2}, p_{a_2})$, eventually leading to $c(n,a_1) + \mathrm{T}_r(F_{n,a_1}, p_{a_1}) \leq C(n,p) \leq c(n,a_2) + \mathrm{T}_r(F_{n,a_2}, p_{a_2}) = b$ in finite time. When this happens, RBFAOO$_r$ examines F_{n,a_1} with $th(F_{n,a_1}) = \min(b, th(n)) - c(n,a_1)$. If $\mathrm{T}_r(F_{n,a_2}, p_{a_2}) = \infty$ (i.e. F_{n,a_2} is proven unsolvable), RBFAOO$_r$ cannot deviate from ST. If $b \ll \infty$, $th(F_{n,a_1}) = b - c(n,a_1)$, since $th(n) = \infty - k$. By Lemma 7, RBFAOO$_r$ returns to F_{n,a_1} in finite time. Since $\mathrm{T}_r(F_{n,a_1}, p_{a_1}) \leq C(F_{n,a_1}, p_{a_1}) \leq b - c(n,a_1)$, $\mathrm{T}_r(F_{n,a_1}, p_{a_1})$ cannot be strictly increased. Therefore, RBFAOO$_r$ must mark F_{n,a_1} as solved, resulting in solving n in finite time. If a_2 also leads to another optimal solution, RBFAOO$_r$ may choose F_{n,a_2} rather than F_{n,a_1}. However, the same discussion holds, resulting in RBFAOO$_r$ making F_{n,a_2} as solved in finite time.

Theorem 2 ensures the solution optimality, once RBFAOO$_r$ finds it. Hence, RBFAOO$_r$ returns an optimal solution cost for s_0 in finite time if a solution exists. □

Theorem 8 leads to a new corollary for RBFS [Korf, 1993] with the TT and our approach, denoted by RBFS$_r$. While the admissibility of RBFS assumes that the search space is a tree, RBFS$_r$ further reduces duplicate search on cyclic spaces.

Corollary 9. *If a solution exists in the finite search space, RBFS$_r$ finds an optimal solution in finite time.*

RBFAOO$_r$ and BLDFS$_r$ can prove unsolvability for the graph in Figure 1(right), but have limitations in other cases:

Theorem 10. *RBFAOO$_r$ and BLDFS$_r$ are incomplete when proving unsolvability in the cyclic search space.*

Proof sketch. The example shown in Figure 2 cannot be solved. An explicit cycle cannot be detected due to large values of $r(B, A \to B)$ and $r(C, A \to C)$. This is easily proven in by induction in a similar way to [Akagi *et al.*, 2010]. □

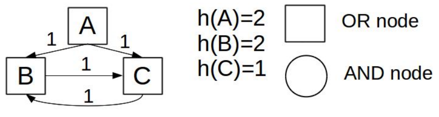

Figure 2: Example for incompleteness, shown in [Akagi et al. 2010].

6 Experimental Results

We use domain-independent nondeterministic planning with unit edge costs and the ADD model to evaluate RBFAOO$_r$ and BLDFS$_r$. Baselines RBFAOO and BLDFS unfold repeated states, ensuring solution optimality with the limited TT size or inconsistent heuristics. We modified slightly the original BLDFS [Bonet and Geffner, 2005a] to delay saving a solution cost in the TT until it is proven optimal, as discussed in Section 4. To the best of our knowledge, this change has not empirically impacted the performance, since a first solution our BLDFS returned is optimal in all cases we observed. We coded all algorithms in C++ (64-bit) and ran experiments on an Intel Xeon CPU X5690 processor at 3.47GHz. The TT is limited to 1GB. The time per instance is 30 minutes. When the TT is full, SmallTreeGC [Nagai, 1999] discards $R\%$ TT entries with small subtrees. As in [Akagi *et al.*, 2010; Kishimoto and Marinescu, 2014], we set R to 30.

We extended the $h^{\max}$ heuristic [Bonet and Geffner, 2001; Haslum and Geffner, 2000] to nondeterministic planning, which is consistent. In constructing an inconsistent heuristic from $h^{\max}$, we perform the perturbation as follows: We represent a state as a bit vector of true or false values of grounded predicates. Among the algorithms compared, the same heuristic values are returned when the same states are evaluated. A set of grounded predicates are randomly selected with the same seed among the algorithms, before search is performed. Whether a small value is subtracted or not is determined by whether one of the selected predicates is true or false. We select 5% of the grounded predicates.

Unsolvable Instances and Inconsistent Heuristics. We use instances from the Uncertainty Track in the 6th International Planning Competition in 2008, consisting of BWD (blocksworld, 30 instances), FLTS (faults, 55 instances) and FRES (first-responders, 100 instances). All these instances admit solutions *with cycles*. They are *unsolvable* under the cycle-free solution requirement. Table 1(top) shows the number of instances proven unsolvable. Without any search, the inconsistent heuristic proves that 26 instances in BWD, and all 100 instances in FRES are unsolvable. That is, no further improvement can be achieved on these instances even with a better search algorithm. For the remaining four instances in BWD, and all 55 instances in FLTS, the heuristic cannot prove unsolvability at their initial states. In these 59 instances, RBFAOO and BLDFS time out, since all they can do is to unfold repeated states until exceeding their thresholds. On the other hand, both RBFAOO$_r$ and BLDFS$_r$ successfully prove unsolvability of all instances in BWD and 14 instances in FLTS (i.e., a total of 18 additional instances proven successfully),

	RBFAOO$_r$	RBFAOO	BLDFS$_r$	BLDFS
BWD (30)	**30**	26	**30**	26
FLTS (55)	**14**	0	**14**	0
FRES (100)	**100**	**100**	**100**	**100**

	RBFAOO$_r$	RBFAOO	BLDFS$_r$	BLDFS
BWD (30)	**10** (2.83)	**10** (3.05)	**10** (2.68)	**10** (2.67)
FLTS (55)	**48** (4344)	**48** (4233)	**47** (2326)	**47** (2247)
FRES (73)	**26** (4452)	**26** (4497)	**27** (5156)	**27** (5072)

Table 1: Performance on unsolvable instances (top) and solvable instances (bottom).

clearly demonstrating the effectiveness of our approach. The p7_2 instance in FLTS is the most difficult that RBFAOO$_r$ and BLDFS$_r$ could prove as unsolvable: RBFAOO$_r$ expands 288,862,627 nodes (589 seconds), while BLDFS$_r$ expands 184,513,049 nodes (351 seconds).

Solvable Instances and Inconsistent Heuristics. Since the instances considered in the previous subsection are unsolvable, we modified them to admit cycle-free solutions, as described in [Fu *et al.*, 2013]. Specifically, in BWD, we make only the *pick-up* action nondeterministic to ensure that the optimal solution is a tree. In FRES, we additionally allow hospitals to accept victims at status *dying*. This change allows to have cases where a solution is not always a sequence of actions but a tree. However, 27 instances are proven to be unsolvable, resulting in 73 solvable instances. These three domains have different characteristics. Nondeterministic actions tend to be selected much more frequently in FRES than in BWD. While the search spaces of BWD and FRES contain cycles, the search space of FLTS is a DAG. For DAGs, since $r(n,p) = q(n)$ always holds for any node n and path p, RBFAOO$_r$ and BLDFS$_r$ have no advantage over RBFAOO and BLDFS but incur extra overhead due to calculating both r-values and q-values. Table 1(bottom) shows the number of instances solved optimally by each algorithm, where the number inside the parentheses is the total search time of the solved instances in seconds. When finding an optimal solution cost, our approach preserves the solving ability of RBFAOO and BLDFS without sacrificing their running time. RBFAOO$_r$ and RBFAOO solve the same set of instances, and so do BLDFS$_r$ and BLDFS. Search time differences range from 1 to 7%, which is a small overhead, compared to benefits pointed out earlier. For example, in FLTS where the search space is a DAG, RBFAOO$_r$ and RBFAOO examine the same portions of the search space, resulting in RBFAOO$_r$ running 2% slower than RBFAOO due to the extra computational overhead. We observed a similar phenomenon for BLDFS$_r$ and BLDFS.

Limited TT and Consistent Heuristics. We selected 5 difficult instances (four have optimal solutions and one does not), and ran an experiment using the consistent $h^{\max}$, a TT limited to 30 MB, and a 2-hour time limit per instance. Table 2 shows the runtimes in seconds. Both RBFAOO$_r$ and BLDFS$_r$ solve three instances including the one (p_7_2) which has no solution in the cyclic search space. On the other hand, RBFAOO and BLDFS solve only two instances which have optimal solutions due to a lack of the cycle detection scheme. We also see that our approach does not sacrifice the ability to solve solvable instances.

	RBFAOO$_r$	RBFAOO	BLDFS$_r$	BLDFS
FRES p4_2	153	153	113	111
FLTS p8_7	834	818	724	712
FLTS p7_2	1786	$\geq 2h$	1608	$\geq 2h$

Table 2: Runtime with consistent heuristic and limited TT. The instances not shown here are not solved by any method.

7 Related Work

Value Iteration (VI) can optimally solve search problems by solving the Bellman equation using dynamic programming [Bellman, 1957]. However, it is limited to solving problems with relatively small search spaces that fit into memory, and may not be able to prove unsolvability due to cycles.

CFC$_{REV^*}$ [Jiménez and Torras, 2000] is a best-first search like AO* [Nilsson, 1980] that can handle cyclic search spaces to find a cycle-free optimal solution but it needs to keep the examined search space in memory. Liang and Su (2005) presents an algorithm that assumes all terminal nodes are known beforehand and available in memory, which does not hold in domains such as nondeterministic planning.

MAO* [Chakrabarti *et al.*, 1989] is an AO* variant that can operate with limited memory but assumes the search space is a DAG and uses a specific strategy to remove nodes from memory. Our approach makes no specific assumption about the replacement strategies of the TT, and the search space can be cyclic as long as an optimal solution is a tree.

LAO* [Hansen and Zilberstein, 2001] combines AO* with dynamic programming (value or policy iteration) to handle optimal solutions involving cycles. LAO* does not operate with limited memory. In addition, in our setting, cycles need to be regarded as unsolvable, and optimal solutions may not contain cycles if they exist. LDFS(MDP) [Bonet and Geffner, 2006] extends LDFS to solve MDP, allowing for cyclic optimal solutions as LAO* does. Extensions of LAO* and LDFS(MDP) to our scenario remain open.

8 Conclusions

In this paper, we focused on finding optimal acyclic solutions in cyclic AND/OR search spaces. We showed in detail how to integrate our ideas into two state-of-the-art algorithms. Our new theoretical analysis addresses important gaps for cases of insufficient cache memory or inconsistent heuristics. Experiments in domain-independent planning demonstrate a boost in the ability to prove unsolvable difficult instances.

In future work we plan to investigate complete algorithms across unsolvable instances. As previously mentioned, an analysis of the original BLDFS technique in combination with our enhancements is another interesting direction.

Acknowledgments

We would like to thank Masataro Asai for his constructive comments on the paper.

References

[Akagi *et al.*, 2010] Yuima Akagi, Akihiro Kishimoto, and Alex Fukunaga. On transposition tables for single-agent search and planning: Summary of results. In *Proceedings of the 3rd Symposium on Combinatorial Search*, pages 1–8, 2010.

[Bellman, 1957] Richard E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[Bonet and Geffner, 2001] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 120:5–33, 2001.

[Bonet and Geffner, 2005a] Blai Bonet and Héctor Geffner. An algorithm better than AO*? In *AAAI*, pages 1343–1348, 2005.

[Bonet and Geffner, 2005b] Blai Bonet and Héctor Geffner. Learning in depth-first search: A unified approach to heuristic search in deterministic, non-deterministic, probabilistic, and game tree settings. Technical report, Universidad Simon Bolivar, 2005. Available at https://bonetblai.github.io/#publications.

[Bonet and Geffner, 2006] Blai Bonet and Héctor Geffner. Learning depth-first search: A unified approach to heuristic search in deterministic and non-deterministic settings, and its application to MDPs. In *ICAPS*, pages 142–151, 2006.

[Campbell, 1985] Murray Campbell. The graph-history interaction: On ignoring position history. In *1985 ACM Annual Conference*, pages 278–280, 1985.

[Chakrabarti *et al.*, 1989] Partha P. Chakrabarti, Sujoy Ghose, Arup Acharya, and S. C. De Sarkar. Heuristic search in restricted memory. *Artificial Intelligence*, 41(2):197–221, 1989.

[Eriksson *et al.*, 2017] Salomé Eriksson, Gabriele Röger, and Malte Helmert. Unsolvability certificates for classical planning. In *ICAPS*, pages 88–97, 2017.

[Eriksson *et al.*, 2018a] Salomé Eriksson, Gabriele Röger, and Malte Helmert. Inductive certificates of unsolvability for domain-independent planning. In *IJCAI*, pages 5244–5248, 2018.

[Eriksson *et al.*, 2018b] Salomé Eriksson, Gabriele Röger, and Malte Helmert. A proof system for unsolvable planning tasks. In *ICAPS*, pages 65–73, 2018.

[Fu *et al.*, 2013] Jicheng Fu, Andres C. Jaramillo, Vincent Ng, Farokh B. Bastani, and I-Ling Yen. Fast strong planning for FOND problems with multi-root directed acyclic graphs. In *Proceedings of the 25th IEEE International Conference on Tools with Artificial Intelligence*, pages 87–94, 2013.

[Hansen and Zilberstein, 2001] Eric A. Hansen and Shlomo Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129:35–62, 2001.

[Haslum and Geffner, 2000] Patrik Haslum and Héctor Geffner. Admissible heuristics for optimal planning. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems*, pages 140–149, 2000.

[Heifets and Jurisica, 2012] Abraham Heifets and Igor Jurisica. Construction of new medicines via game proof search. In J. Hoffmann and B. Selman, editors, *AAAI*, pages 1564–1570, 2012.

[Holte, 2010] Robert C. Holte. Common misconceptions concerning heuristic search. In *Proceedings of the 3rd Symposium on Combinatorial Search*, pages 46–51, 2010.

[Jiménez and Torras, 2000] Pablo Jiménez and Carme Torras. An efficient algorithm for searching implicit AND/OR graphs with cycles. *Artificial Intelligence*, 124:1–30, 2000.

[Kishimoto and Marinescu, 2014] Akihiro Kishimoto and Radu Marinescu. Recursive best-first AND/OR search for optimization in graphical models. In *UAI*, pages 400–409, 2014.

[Kishimoto and Müller, 2004] Akihiro Kishimoto and Martin Müller. A general solution to the graph history interaction problem. In *AAAI*, pages 644–649, 2004.

[Korf, 1985] Richard E. Korf. Depth-first iterative deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[Korf, 1993] Richard E. Korf. Linear-space best-first search. *Artificial Intelligence*, 62:41–78, 1993.

[Liang and Su, 2005] Qianhui A. Liang and Stanley Y. W. Su. AND/OR graph and search algorithm for discovering composite web services. *International Journal of Web Services Research*, 2(4):48–67, 2005.

[Nagai, 1999] Ayumu Nagai. A new depth-first search algorithm for AND/OR trees. Master's thesis, The University of Tokyo, 1999.

[Nilsson, 1980] Nils J. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing Co, Palo Alto, CA, 1980.

[Plaat *et al.*, 1996] Aske Plaat, Jonathan Schaeffer, Wim Pijls, and Arie de Bruin. Best-first fixed-depth minimax algorithms. *Artificial Intelligence*, 87(1-2):255–293, 1996.

[Reinefeld and Marsland, 1994] Alexander Reinefeld and T. Anthony Marsland. Enhanced iterative-deepening search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(7):701–710, 1994.