Consistent Heuristic Bidirectional Search

Songun Lee

Feb 2021

Abstract

Inspired by recent advances in bidirectional heuristic search, we present consistent heuristic variant of two state-of-the-art algorithms. The CH-NBS algorithm has similar near-optimality guarantee as with NBS algorithm, while the CH-DVCBS algorithm expands less node in real-world examples. These algorithms can be seen as efficient front-to-front search algorithm when the heuristic is decomposable. Computational experiments on sliding tile puzzles show state-of-the-art results.

1 Introduction

Historically, it was conjectured that with strong heuristics bidirectional search will expand more nodes than A*. Recent algorithms such as NBS[1], DVCBS[5] and DIBBS[4] challenged this belief, often outperforming A*. While NBS has optimality guarantee up to constant in DXBB[2] setting, DVCBS outperforms NBS in computational experiments. DIBBS utilizes consistency of heuristic, and shows great performance in experiments, but effectively symmetrizes the heuristic and does not utilize all information given. Our work extends ideas of DIBBS and gives sufficient condition for node expansion in consistent heuristics setting. We present variant of state-of-the-art algorithms adapted to consistent heuristics setting, one with near-optimality guarantee of NBS, and the other with great real-world performance of DVCBS. Our work contains DIBBS as special case when the heuristic is symmetrized as following.

$$(h_f(u), h_b(u)) \leftarrow (\frac{h_f(u) + h_b(t) - h_b(u)}{2}, \frac{h_b(u) + h_f(s) - h_f(u)}{2})$$

2 Notation

Let $G = (V, E)$ be a directed graph with finite edges and $c : E \to \mathbb{R}$ be an edge cost function. The problem is to find the lowest cost path from s to t for given node pair (s, t). Assume front-to-end consistent heuristic given, that is $h_f, h_b : V \to \mathbb{R}$ satisfying:

$$h_f(u) - h_f(v) \leq c(e)$$
$$h_b(v) - h_b(u) \leq c(e)$$

for all $e : u \to v \in E$. We do not assume $h_f(s) = h_b(t) = 0$, but it can be restored by subtracting $h_f(s)$ and $h_b(t)$ from h_f and h_b respectively.

We define several variables as following. By $u \to v$ we denote a directed edge, and by $u \rightsquigarrow v$ we denote a path from u to v. Our algorithms have two values in role of f-value, which are called σ and δ respectively.

$$
\begin{aligned}
s &= \text{source node} \\
t &= \text{target node} \\
d &= \text{direction} = \begin{cases} f & \text{(searching from } s) \\ b & \text{(searching from } t) \end{cases} \\
-d &= \text{reverse direction} = \begin{cases} b & (d = f) \\ f & (d = b) \end{cases} \\
c(-) &= \text{cost of an edge or a path} \\
C^* &= \text{optimal cost for } s \rightsquigarrow t \\
g_d(u) &= \text{minimum cost to reach } u \\
h_d(u) &= \text{heuristic/estimate of } g_{-d}(u) \\
\delta_d(u) &= g_d(u) - h_{-d}(u) \\
\sigma_d(u) &= g_d(u) + h_d(u) \\
\eta(u) &= h_f(u) + h_b(u) \\
lb(u, v) &= \max(\delta_f(u) + \sigma_b(v), \sigma_f(u) + \delta_b(v))
\end{aligned}
$$

We also introduce non-optimal variants of above definitions, such as

$$
\begin{aligned}
\delta_d(u, P) &= c(P) - h_{-d}(u) \\
\sigma_d(u, P) &= c(P) + h_d(u)
\end{aligned}
$$

where P is a path $s \rightsquigarrow u$ when $d = f$, or $u \rightsquigarrow t$ when $d = b$.

3 Must Expand Pair

Theorem 1. *$\sigma_d(u, P)$ and $\delta_d(u, P)$ monotonically increases along path.*

Proof. Proposition 5 of [4]. □

Theorem 2. *$lb(u, v)$ gives lower bound of cost of path $s \rightsquigarrow u \rightsquigarrow v \rightsquigarrow t$.*

Proof.

$$
\begin{aligned}
&c(s \rightsquigarrow u) \geq g_f(u) \\
&c(u \rightsquigarrow v) \geq \max(h_f(u) - h_f(v), h_b(v) - h_b(u)) && (\because \text{consistency}) \\
&c(v \rightsquigarrow t) \geq g_b(v)
\end{aligned}
$$

□

We prove results similar to [2]. We define MEP(Must-Expand-Pair) as $(u, v) \in V^2$ such that $lb(u, v) < C^*$. We assume the algorithm has no further knowledge about problem.

Theorem 3. *Let u, v be two nodes. To prove $lb(u, v) < C^*$, the search algorithm must expand either u or v.*

Proof. It suffices to show that we can add a directed edge from u to v of cost $C = \max(h_f(u) - h_f(v), h_b(v) - h_b(u))$ without violating consistency of heuristics, as it will produce a path $s \rightsquigarrow u \rightarrow v \rightsquigarrow t$ of cost $g_f(u) + C + g_b(v) = lb(u, v)$. Also, it is sufficient to check between u and v, which is trivial from the definition of C. □

We denote the bipartite graph formed by MEPs as G_{MX}. From theorem 3, it follows that the minimum number of node expansion needed is the size of minimum vertex cover of G_{MX}. We call this number MVC.

Theorem 4. *Fix a minimal vertex cover of G_{MX}. If v has smaller or equal (σ_d, δ_d) than a node in the cover, the cover contains v.*

Proof. Let u be the node in the cover with larger or equal (σ_d, δ_d) than v. If the cover does not contain v, every w with $lb(v, w) < C^*$ must be contained. Therefore, u can be safely removed from the cover, which is contradiction to minimality. □

4 Search Algorithm CH-NBS

The Algorithm CH-NBS is presented as Algorithm 1 and 2. The Backward-Expand function is analogous to Forward-Expand, and hence not shown here. In further analysis, we omit the case where $s = t$ for simplicity.

Theorem 5. *During execution, $g_d(u)$ does not change after d-ward expansion of u. And this value is correct.*

Proof. For u to be expanded d-ward there should be contained in a pair achieving minimum $lb(u, v)$. Therefore, no node in $Open_D$ could have strictly smaller (σ_d, δ_d) compared to n. By theorem 1 every node expansion results in larger or equal (σ_d, δ_d) compared to its parent, therefore such node cannot exist in the future. Hence no further expansion can update $g_d(u)$, as for it to happen, the expanded node should have strictly smaller (σ_d, δ_d), as it means $(\sigma_d(n), \delta_d(n))$ being updated to a strictly smaller pair.

Now assume there was an expansion of a node with incorrect g_d. Let u be the first such node expanded. For u to be expanded d-ward, all node with strictly smaller (σ_d, δ_d) pair has been expanded. Therefore every unexplored path to u would cost more or equal than $g_d(u)$, and as the obtained $g_d(u)$ is the minimum cost of all the explored paths to u, it conforms to its definition, which contradicts the assumption of u. □

Algorithm 1: CH-NBS

if $s = t$ **then**
  return *0*
end
$UB \leftarrow \infty$
$Open_F \leftarrow \{s\}, Open_B \leftarrow \{t\}$
$\forall_{x \in V} g_f(x) \leftarrow \infty, g_b(x) \leftarrow \infty$
$g_f(s) \leftarrow 0, g_b(t) \leftarrow 0$
while $Open_F \neq \emptyset$ *and* $Open_B \neq \emptyset$ **do**
  $(u, v) \leftarrow \arg\min_{Open_F \times Open_B} lb$
  if $lb(u, v) \geq UB$ **then**
    break
  end
  Forward-Expand(u)
  Backward-Expand(v)
end
return UB

Algorithm 2: Forward-Expand

Input: u
$Open_F \leftarrow Open_F \setminus \{u\}$
foreach $e : u \rightarrow v$ **do**
  if $g_f(v) > g_f(u) + c(e)$ **then**
    $g_f(v) \leftarrow g_f(u) + c(e)$
    $Open_F \leftarrow Open_F \cup \{v\}$
    $UB \leftarrow \min(UB, g_f(v) + g_b(v))$
  end
end

Theorem 6. *Each node is expanded at most once.*

Proof. By theorem 5, the only way to be expanded more than once is being expanded forward once and backward once. Assume u is a node that is expanded twice. Let (u, v) be the selected pair when u was expanded forward, and (w, v) be the selected pair when u was expanded forward. As UB monotonically decreases as algorithm executes and is upper bounded by $g_f(u) + g_b(u)$ before u expands twice, and the lb of selected pair monotonically increases along execution, we obtain $lb(u, v) < g_f(u) + g_b(u)$, $lb(w, u) < g_f(u) + g_b(u)$, which implies $lb(w, v) \leq lb(u, v) + lb(w, u) - (g_f(u) + g_b(u)) < \min(lb(u, v), lb(w, u))$, therefore either w or v must be expanded earlier in their respective direction. □

Theorem 7. *CH-NBS halts.*

Proof. By theorem 5 each edge can be expanded at most once per direction. As we assumed finite number of edges, the algorithm halts. □

Theorem 8. *CH-NBS finds optimal cost.*

Proof. It is clear from the algorithm that UB is provable upper bound of C^*. Assume that there's a path $p : s \rightsquigarrow t$ of cost lower than returned UB. For any subpath $u \rightsquigarrow v$, as $lb(u, v) \leq c(p) < UB$, either u must be expanded forward or v must be backward expanded forward, as the algorithm halts by theorem 7.

Case 1 : When there's unexpanded node in p, denote the node, its direct ancestor and direct child in p as v, u, w respectively, if they exist. u must be expanded forward and w must be expanded backward. Assume, without loss of generality u was expanded earlier than w. In backward expansion of w, UB is updated to $\max(UB, g_f(v) + g_b(v))$, and by the time of execution, $g_f(v) \leq g_f(v, p)$, $g_b(v) \leq g_f(v, p)$ as $g_f(u)$, $g_b(w)$ is of correct value (Theorem 5), therefore resulting UB is less or equal to $c(p)$, which is a contradiction. If either u or w does not exist, assume without loss of generality that $v = t$. UB is updated to $\max(UB, g_f(v)+g_b(v))$ during expansion of u, and because $g_b(v) = 0$, $g_f(v) \leq c(u \xrightarrow{p} v) + g_f(u) \leq c(p)$ as $g_f(u)$ is optimal when u is expanded by theorem 5, UB is less or equal to $c(p)$. This results in contradiction.

Case 2 : If every node in p is expanded, let u be the farthest node from s in p that is expanded. If $u = t$, the proof concludes as from forward expansion of direct ancestor of u, UB is bounded by $d_f(u) + d_b(u)$, which is less or equal to $c(p)$ as $d_b(t) = 0$ since the start of the algorithm. Else, let v be the direct child of u in p, which must be expanded backward. If u was expanded later than v, UB is upper bounded by $d_f(v) + d_b(v) \leq c(p)$ during expansion of u. If u was expanded earlier than v, UB is upper bounded by $d_f(u) + d_b(u) \leq c(p)$ during expansion of v. In any case, it results in contradiction. □

Theorem 9. *CH-NBS expands at most* $2 \times MVC$ *nodes.*

Proof. By theorem 8, any (u, v) found during algorithm 1 forms a MEP, so any vertex cover of MEPs should include either u or v. By theorem 5, each node is only considered once per direction, therefore the selection is done at most MVC times, and the expansion is done at most $2 \times MVC$ times. □

5 Efficient Implementation of CH-NBS

In this section, we present details to find $\arg\min lb$ efficiently with amortized complexity of $O(\log|Open_F \cup Open_B|)$ per update. We proceed in two steps. First, we find nodes with minimal (σ_d, δ_d), denoted as $Pareto_D$. Second, we find $\arg\min lb(u, v)$ from them.

5.1 Pareto Frontier between δ_d, σ_d

We form a *self-balancing segment tree* from nodes of $Open_D \setminus Pareto_D$. First, construct a self-balancing binary search tree along the lexicographic order.

$$n >_{\text{lex}} m \iff (\delta_d(n) > \delta_d(m)) \vee (\delta_d(n) = \delta_d(m)) \wedge (\sigma_d(n) > \sigma_d(m))$$

Then, attach to each node the colexicographical minimum of its children.

$$n >_{\text{colex}} m \iff (\sigma_d(n) > \sigma_d(m)) \vee (\sigma_d(n) = \sigma_d(m)) \wedge (\delta_d(n) > \delta_d(m))$$

This information requires constant recalculation per rotation, allowing the tree to be self-balanced. As a balanced tree, addition or removal of a node and updating attached minimum takes $O(\log|Open_D \setminus Pareto_D|)$ operations. As a segment tree, querying colexicographical minimum inside lexicographical segment takes same complexity. Because of this, finding newly Pareto optimal nodes takes $O((1+m)\log|Open_D \setminus Pareto_D|)$ where m is number of such nodes. As sum of m for such operations is at most $|Open_D|$, this gives amortized complexity of $O(\log|Open_D \setminus Pareto_D|)$. The pseudocode for this, without consideration of edge cases are presented as Algorithm 3. η_d and subsequent operations are explained in next subsection. We present theorem 10 for completeness.

Algorithm 3: Update Pareto Frontier Set

Input: u : the node being expanded/removed from $Open_D$
Output: newly pareto optimal nodes

```
v ← Node of Pareto_D with largest η_d among those smaller than η_d(u)
w ← Node of Pareto_D with smallest η_d among those larger than η_d(u)
while v ≠ w do
    v ← colexicographical minimum inside lexicographical segment (v, w]
    Pareto_D ← Pareto_D ∪ {v}
end
```

Theorem 10. *There is a node pair in $Pareto_F \times Pareto_B$ with lb value of $\min lb(u, v)$, unless either $Open_F$ or $Open_B$ is empty.*

Proof. Let (u, v) be a pair of $Open_F \times Open_B$ achieving minimum lb. As $|Open_F|$ is finite, there exists $u' \in Pareto_F$ such that $\sigma_f(u') \leq \sigma_f(u)$ and $\delta_f(u') \leq \delta_f(u)$. Same thing applies to v, and let v' be respective node. The proof concludes as $lb(u', v') \leq lb(u, v)$. $\square$

5.2 $\arg\min lb$

We store $Pareto_F \coprod Pareto_B$ as a self-balancing binary tree, ordered by $\sigma_d - \delta_d$. Here, $\coprod$ denotes disjoint union. We define η_d as $\sigma_d - \delta_d$ for brevity.

Theorem 11. *Order $Pareto_F \coprod Pareto_D$ by η_d. There is an adjacent pair of it achieving $\min lb(u, v)$, unless either $Pareto_F$ or $Pareto_B$ is empty.*

Proof. Assume $a \in Pareto_F$, $b \in Pareto_B$ achieves the minimum. Define $x > y$ as x comes after y in the ordered array. Without loss of generality, assume $a < b$. There exists $a' \in Pareto_F$, $b' \in Pareto_B$ such that

$$a \le a' < b' \le b$$
$$\nexists_{c \in OP} a' < c < b'$$

, as there must be transition from $Pareto_F$-nodes segment to $Pareto_B$-nodes segment between a and b. In this case, $lb(a, b) \ge lb(a', b')$, therefore a', b' is a adjacent pair that achieves the minimum. □

Therefore, it is enough to store lb for adjacent $Pareto_F$, $Pareto_B$ node pair in a heap. Each addition or deletion of node in $Pareto_D$ makes $O(1)$ update and $O(1)$ query – previous and next node – to the tree and $O(1)$ updates to the heap, running in $O(\log |Pareto_F \cup Pareto_B|)$. Querying the minimum of heap takes same complexity.

6 Search Algorithm CH-DVCBS

We present variant of DVCBS algorithm with similar idea. The pseudocode is presented as Algorithm 4. Assuming interger heuristic and cost, there are at most $O(C^*)$ nodes without strictly smaller (σ_d, δ_d) after clustering, and the minimum vertex cover can by found in $O(C^*)$ each iteration, because maximum matching is obtained from greedily matching from nodes with smallest η_d. This requires maintaining list of weakly pareto optimal nodes ordered by η_d, which can be done similarly like previous section. With further optimization, it seems updating the maximum vertex cover is possible with amortized complexity of $O(\log |Open_F \cup Open_B|)$, per removal or addition of weakly pareto optimal nodes, though we do not describe it here.

Theorem 12. *Each node is expanded at most once, the algorithm halts with correct answer.*

Proof. Identical to CH-NBS. □

In the extreme case when node is not clustered, CH-DVCBS is asymmetric generalization of DIBBS. If clustering is further done lossily by $\sigma_d + \delta_d$, it becomes either DIBBS, or DIBBS with better termination condition depending on implementation.

Algorithm 4: CH-DVCBS

if $s = t$ **then**
  return *0*
end
$UB \leftarrow \infty$
$Open_F \leftarrow \{s\}, Open_B \leftarrow \{t\}$
$\forall_{x \in V} g_f(x) \leftarrow \infty, g_b(x) \leftarrow \infty$
$g_f(s) \leftarrow 0, g_b(t) \leftarrow 0$
while $Open_F \neq \emptyset$ *and* $Open_B \neq \emptyset$ **do**
  $lbmin \leftarrow \min_{Open_F \times Open_B} lb$
  if $lbmin \geq UB$ **then**
    break
  end
  $G_{MX} \leftarrow (Open_F, Open_B, \{(u, v) : lb(u, v) = lbmin\})$
  Cluster nodes by (σ_d, δ_d)
  Find a Minimum Vertex Cover of G_{MX}
  Choose and Expand a cluster in the minimum vertex cover
end
return UB

7 Experiments and Analysis

We evaluate algorithms using 100 instances 15-puzzle from [3]. Table 1 shows average node expansion using Manhattan distance heuristic.

algorithm	reported	reimplemented
IDA*	184,336,714	
A*		17,352,439
NBS	12,851,889	
DVCBS	11,669,720	
GBFHS	12,507,393	
DIBBS	1,603,867	2,474,789
CH-NBS		2,352,072
CH-DVCBS		$\leq$ 1,828,138

Table 1: Average node expansion on 100 instances of 15-puzzle from [3].

8 Conclusion

9 Future Work

Our work can be seen as a efficient front-to-front algorithm when the heuristic can be written as $h(u, v) = \max(f(u) - f(v), g(u) - g(v))$. As dynamic shortest pair problem can be solved in $O(|V|)$ in case of euclidean space, in theory, front-to-front algorithm for 15-puzzle with manhattan distance can be made efficiently, in terms of time complexity, as manhattan distance can be seen as 45-dimensional hamming distance. But the constant is impractically large, and we'll address these in the future work.

References

[1] Jingwei Chen, Robert C. Holte, Sandra Zilles, and Nathan R. Sturtevant. Front-to-end bidirectional heuristic search with near-optimal node expansions. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 489–495, 2017.

[2] Juergen Eckerle, Jingwei Chen, Nathan Sturtevant, Sandra Zilles, and Robert Holte. Sufficient conditions for node expansion in bidirectional heuristic search. *Proceedings of the International Conference on Automated Planning and Scheduling*, 27(1), Jun. 2017.

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

[4] E.C. Sewell and S.H. Jacobson. Dynamically improved bounds bidirectional search. *Artificial Intelligence*, 291:103405, 2021.

[5] Shahaf S. Shperberg, Ariel Felner, Nathan R. Sturtevant, Solomon E. Shimony, and Avi Hayoun. Enriching non-parametric bidirectional search algorithms. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):2379–2386, Jul. 2019.