Sufficient Conditions for Node Expansion in Bidirectional Heuristic Search

Jürgen Eckerle
Berner Fachhochschule
Departement Technik
und Informatik
Switzerland

Jingwei Chen
Nathan R. Sturtevant
University of Denver
USA
(jingchen@cs.du.edu)
(sturtevant@cs.du.edu)

Sandra Zilles
Computer Science Dept.
University of Regina
Canada
(zilles@uregina.ca)

Robert C. Holte
Computing Science Dept.
University of Alberta
Canada
(rholte@ualberta.ca)

Abstract

In this paper we study bidirectional state space search with consistent heuristics, with a focus on obtaining sufficient conditions for node expansion, i.e., conditions characterizing nodes that must be expanded by any admissible bidirectional search algorithm. We provide such conditions for front-to-front and front-to-end bidirectional search. The sufficient conditions are used to prove that the front-to-front bidirectional search algorithm BDS1 is optimally efficient, in terms of node expansion, among a broad class of bidirectional search algorithms, for a specific class of problem instances. Dechter and Pearl's well-known result on sufficient conditions for node expansion by unidirectional algorithms such as A* is shown to be a special case of our results.

1 Introduction

Dechter & Pearl (1985) proved that A* is optimally efficient ("0-optimal" in their terminology), among all admissible, equally informed, unidirectional search algorithms, when its heuristic function is consistent and the problem instance being solved is non-pathological.[1] The key to this proof is having a sufficient condition for node expansion, i.e. a characterization of nodes that must be expanded by any admissible unidirectional search algorithm when the heuristic is consistent. Our aim in this paper is to give a sufficient condition for node expansion for admissible bidirectional search algorithms when they are given consistent heuristics.

Our interest in this question is sparked by the recent development of a bidirectional heuristic algorithm, MM (Holte et al. 2016), that is admissible and has a strong worst-case guarantee because it only expands nodes that are within distance $\frac{1}{2}C^*$ of the $start$ or $goal$ states, where C^* is the cost of an optimal solution. This raises the question whether MM is optimally efficient (when specific conditions hold). We do not answer that question in this paper, but our contributions here provide important insight into the question.

[1] They define a problem instance to be "non-pathological" if there exists an optimal path from start to goal, $U = u_0, \ldots, u_n$, such that $h(u_i)$ is strictly smaller than the true distance from u_i to the goal for all non-goal states in U (p. 522). A search algorithm is "admissible" if it is guaranteed to return an optimal solution whenever its heuristic is admissible (p. 520).

Bidirectional heuristic search algorithms fall into two categories depending on the type of heuristic function they use. Front-to-front algorithms use one heuristic function, $h(u, v)$, that estimates the distance between any two states (u and v). Examples of front-to-front algorithms are BHFFA2 (de Champeaux 1983), SFBDS (Felner et al. 2010; Lippi, Ernandes, and Felner 2012), and BIDA* (Manzini 1995). By contrast, front-to-end bidirectional heuristic search algorithms use two heuristic functions, h_F and h_B, with $h_F(u)$ estimating the distance from u to $goal$ and $h_B(v)$ estimating the distance from $start$ to v. Examples of front-to-end algorithms are MM, BHPA (Pohl 1971), and BS* (Kwa 1989).

We analyze these two types of bidirectional algorithms separately and derive sufficient conditions for each (Sections 4 and 6). We show (Section 7) that the sufficient conditions for front-to-front algorithms logically imply those for front-to-end systems. Both analyses are similar to Dechter & Pearl's analysis of unidirectional search algorithms in their overall approach, which can be summarized as follows.

If, in optimally solving a particular problem instance I, algorithm B fails to expand a state u that satisfies the proposed sufficient conditions then a new instance I' can be created by adding a new edge e emanating from u such that the optimal solution paths in I' contain e and are strictly cheaper than the optimal solution paths in I. But B will behave exactly the same on I' as it did on I, so it will return the same solution path for I' as it did for I. This path is not optimal for I' and therefore B is not an admissible algorithm.

There is one point in this proof sketch that was not adequately addressed by Dechter & Pearl: why must B behave exactly the same on I' as it did on I? The two instances are "obviously" different (I' has an edge that I does not have), why cannot B detect this and behave differently? The answer lies in certain assumptions being made about the search algorithms that were not clearly articulated by Dechter & Pearl. In Section 3 we give a full, explicit statement of these assumptions. We regard this as an important contribution since these assumptions are, in fact, somewhat limiting.

We use the sufficient conditions for node expansion that we derive in two different ways. First, in Section 5 we use the sufficient conditions to prove that the front-to-front algorithm BDS1 (Eckerle and Ottmann 1994) is optimally efficient, for a certain class of problem instances, among admissible search algorithms satisfying the assumptions of

Section 3. This is analogous to the corresponding result by Dechter & Pearl for A*.

Secondly, in Section 6 we show how unidirectional search systems can be seen as a special case of front-to-end bidirectional search algorithms and derive Dechter & Pearl's sufficient conditions for node expansion by unidirectional search algorithms as a special case of the sufficient conditions we derived for front-to-end bidirectional search.

In Section 8 we expose the limitations of our analysis. Several of these are shared by Dechter & Pearl's analysis but some are specific to bidirectional search.

2 Terminology and Notation

Our terminology and notation in Sections 2 through 4 is for front-to-front bidirectional search. A few small modifications are required for front-to-end bidirectional search (see Section 6).

A state space G is a finite directed graph whose vertices are states and whose edges are pairs of states.[2] Each edge (u, v) has a cost $c(u, v) \geq 0$. A forward path in G is a finite sequence $U = (U_0, \ldots, U_n)$ of states in G where (U_i, U_{i+1}) is an edge in G for $0 \leq i < n$. We say that forward path U contains edge (u, v) if $U_i = u$ and $U_{i+1} = v$ for some i. Likewise, a backward path is a finite sequence $V = (V_0, \ldots, V_m)$ of states where (V_i, V_{i+1}) is a "reverse" edge, i.e. (V_{i+1}, V_i) is an edge in G for $0 \leq i < m$. Backward path V contains reverse edge (u, v) if $V_i = u$ and $V_{i+1} = v$ for some i. The reverse of path $V = (V_0, \ldots, V_m)$ is $V^{-1} = (V_m, \ldots, V_0)$. The cost of a reverse edge equals the cost of the corresponding original edge. A path pair (U, V) has a forward path (U) as its first component and a backward path (V) as its second component.

If U is a path (forward or backward), $|U|$ is the number of edges in U, $c(U)$ is the cost of U (the sum of the costs of all the edges in U), U_i is the i^{th} state in U ($0 \leq i \leq |U|$), and $c(U, i)$ is the sum of the costs of the first i edges in U (the cost to reach state U_i via path U). $U_{|U|}$ is the last state in path U, which we also denote $end(U)$. $\lambda_F = (start)$ and $\lambda_B = (goal)$ are the empty forward and backward paths from $start$ and $goal$, respectively. Note that $end(\lambda_F) = start$ while $end(\lambda_B) = goal$. Both λ_F and λ_B have a cost of 0. Forward (backward, resp.) path U is optimal if there is no cheaper forward (backward, resp.) path from U_0 to $end(U)$. $d(u, v)$ is the distance from state u to state v, i.e., the cost of the cheapest forward path from u to v. If there is no forward path from u to v then $d(u, v) = \infty$.

Given two states in G, $start$ and $goal$, a solution path is a forward path from $start$ to $goal$. $C^* = d(start, goal)$ is the cost of the cheapest solution path.

Definition 1. *A front-to-front heuristic h maps a pair of states in G to a non-negative real number or to ∞. h is* bi-admissible *iff $h(u, v) \leq d(u, v)$ for all states u and v in G, and is* bi-consistent *iff for all u and v in G:*
(1) $h(u, v) \leq d(u, u') + h(u', v)$ for all u' in G and
(2) $h(u, v) \leq d(v', v) + h(u, v')$ for all v' in G.

[2] If G has multiple edges from state u to state v, we ignore all but the cheapest of them.

If front-to-front heuristic h is bi-consistent and $h(v, v) = 0$ for all v, then it is also bi-admissible.

Given a front-to-front heuristic h, for any path pair (U, V) we define

$$f(U, V) = c(U) + c(V) + h(end(U), end(V)) .$$

A problem instance is defined by specifying a state space G, a start state ($start$), a goal state ($goal$), and a front-to-front heuristic h. A problem instance is solvable if there is a forward path in G from $start$ to $goal$. Following Dechter & Pearl's notation, we use I_{AD} to refer to the set of solvable problem instances in which the heuristic is bi-admissible. I_{CON} is the set of solvable problem instances in which the heuristic is bi-admissible and bi-consistent. A search algorithm is admissible iff it is guaranteed to return an optimal solution for any problem instance in I_{AD}.

3 Assumptions About Search Algorithms

In order to derive sufficient conditions for a node to be expanded, it is necessary to make certain assumptions about the search algorithms being considered. For example, Dechter & Pearl (p. 520) make the following assumptions:

> "We assume that each algorithm compared with A* uses the primitive computational step of node expansion, that it only expands nodes that were generated before, and that it begins the expansion process at the start node."

We make the same assumptions but generalized to bidirectional search. To express these assumptions precisely, we will require that the state space G be represented in a certain manner and restrict our analysis to search algorithms that instantiate the template shown in Algorithm 1. Although Algorithm 1 is presented as a front-to-front algorithm, unidirectional search and front-to-end bidirectional search easily fit into this framework, as will be detailed in Section 6.

In particular, we require a state space, G, to be represented implicitly by a 5-tuple $(start, goal, c, expand_F, expand_B)$ consisting of a start state ($start$), a goal state ($goal$), an edge cost function (c), a successor function ($expand_F$), and a predecessor function ($expand_B$). The input to $expand_F$ is a forward path U. Its output is a sequence $(U^1, \ldots, U^n)$, where each U^k is a forward path consisting of U followed by one additional state ($end(U^k)$) such that $(end(U), end(U^k))$ is an edge in G. There is one U^k for every state s such that $(end(U), s)$ is an edge in G. Likewise, the input to $expand_B$ is a backward path V and its output is a sequence $(V^1, \ldots, V^m)$, where each V^k is a backward path consisting of V followed by one additional state ($end(V^k)$) such that $(end(V^k), end(V))$ is an edge in G. There is one V^k for every state s such that $(s, end(V))$ is an edge in G. For reasons that will be given at the end of this section, we assume that $expand_F$ and $expand_B$ are deterministic, i.e. each produces identical output (the same paths in the same order) when given the same input.

A problem instance is then fully specified by (G, h), the 5-tuple representing G and a front-to-front heuristic h. This is the input to Algorithm 1.

With the exception of S_0, each S_t in Algorithm 1 simply records the result of the expansion performed on iteration t. S_t is therefore a finite sequence of path pairs. S_0

Algorithm 1: Generic Search Algorithm

Input: $(start, goal, c, expand_F, expand_B), h$
Output: a least-cost path from $start$ to $goal$

$S_0 := ((\lambda_F, \lambda_B))$
for $t := 1$ *to* ∞ **do**
  if `StoppingCondition`$(t, S_0, \ldots, S_{t-1}, c, h)$ **then**
    return `Solution`$(S_0, \ldots, S_{t-1}, c)$
  $Dir, (U, V) :=$ `Choose`$(t, S_0, \ldots, S_{t-1}, c, h)$
  if Dir = *Forward* **then**
    $(U^1, ..., U^n) := expand_F(U)$
    $S_t := ((U^1, V), (U^2, V)...(U^n, V))$
  else
    $(V^1, ..., V^m) := expand_B(V)$
    $S_t := ((U, V^1), (U, V^2)...(U, V^m))$

contains just one path pair, (λ_F, λ_B). We call $S_0, \ldots, S_{t-1}$ the expansion sequence at the beginning of iteration t or, if t is implied by context, we call it the current expansion sequence. We say that state pair (u, v) occurs in an expansion sequence if state u occurs in some forward path in the expansion sequence and state v occurs in some backward path in the expansion sequence. Similarly, edge (u, v) occurs in an expansion sequence if it occurs in some forward path in the expansion sequence or if its reverse, (v, u), occurs in some backward path in the expansion sequence.

The template in Algorithm 1 is instantiated by supplying three functions, `StoppingCondition`, which determines if the algorithm will stop or continue to search, `Solution`, which extracts a solution path from the current expansion sequence, and `Choose`, which inspects the current expansion sequence and chooses a search direction for the next expansion (Dir is either "Forward" or "Backward") and a path pair (U, V) to expand, where U is a forward path occurring in the given expansion sequence $(S_0, \ldots, S_{t-1})$, and V is a backward path occurring in the given expansion sequence. These user-supplied functions may only apply the heuristic function h to states that occur in the given expansion sequence and may only apply the cost function c to edges that occur in the given expansion sequence. Internally these functions can store and share private data structures such as the traditional Open and Closed lists. It is important that none of these three functions has access to $expand_F$ or $expand_B$. Moreover, we assume that they are deterministic. $start$ and $goal$ are not explicitly passed to these three functions but can be computed by them from S_0 by applying the end function to λ_F or λ_B.

Definition 2. *If u is a state, we say that u is* forward expanded *iff $expand_F$ is applied to some forward path U for which $end(U) = u$ and we say that u is* backward expanded *iff $expand_B$ is applied to some backward path V for which $end(V) = u$. To* expand a state pair (u, v) *is to either forward expand u or backward expand v.*

The assumption that all of the search algorithm's internal operations are deterministic is essential for the proofs of Theorems 1 and 6 below. It is also essential for Dechter & Pearl's proof of the analogous theorem for A* (Theorem 8 in (Dechter and Pearl 1985)), although they do not explicitly state it as an assumption. All three proofs use the same reasoning. If algorithm B optimally solves problem instance $I \in I_{CON}$ without expanding a state u (or state pair (u, v)) that satisfies the proposed sufficient conditions then a new instance $I' \in I_{AD}$ can be created by adding a new edge emanating from u such that the optimal solution paths in I' contain this new edge and are strictly cheaper than the optimal solution paths in I. The proofs then assert that B will behave exactly the same on I' as it did on I, so it will not expand u (or v) and therefore not find the optimal solution for I'.[3] This assertion is based on two assumptions. The first is that B's component functions are all deterministic. The second is that the only way B can detect the new edge is by expanding u (or v); for example B is not able to notice that the two instances have different successor and predecessor functions simply by inspecting the $expand$ functions' definitions. We call this the "black box" assumption, since B has only black box access to $expand_F$ and $expand_B$. Likewise, we require that B only has black box access to functions c, h, and end.

Definition 3. *A bidirectional search algorithm satisfying all the conditions of this section is called a* Deterministic, Expansion-based, Black Box (DXBB) algorithm.

4 A Sufficient Condition for Node Expansion for Front-to-Front Bidirectional Search

This section contains our first main result, which provides a sufficient condition for a state pair to be expanded by an admissible DXBB algorithm. The theorem and its proof bear strong similarities to Dechter & Pearl's Theorem 8 and its proof, but there are technical differences imposed by the bidirectional setting.

Theorem 1. *Let $I = (G, h) \in I_{CON}$ have an optimal solution cost of C^*. If U is an optimal forward path and V is an optimal backward path such that $U_0 = start$, $V_0 = goal$, and $f(U, V) < C^*$ then, in solving problem instance I, any admissible DXBB algorithm must expand $(end(U), end(V))$.*

Proof. We prove the contrapositive. Suppose there is a problem instance $I = (G, h) \in I_{CON}$ whose optimal solution cost is C^*, an optimal forward path U and an optimal backward path V such that $U_0 = start$, $V_0 = goal$, and $f(U, V) < C^*$, and a DXBB algorithm B that solves I correctly (returns a path $B(I)$ costing C^*) without expanding $(end(U), end(V))$. Then a new instance $I' = (G', h') \in I_{AD}$ can be constructed having an optimal solution strictly cheaper than C^* on which B also returns path $B(I)$ (costing C^*), thereby showing that B is not admissible.

$I' = (G', h')$ is defined as follows: $h' = h$, and G' has all the vertices in G and all the edges in G except the edge, if

[3] In the proof of their Theorem 8 (p. 524) Dechter & Pearl say "Algorithm B, on the other hand, if it avoids expanding $[u]$, must behave the same as in problem instance I halting with cost C^*".

there is one, from $end(U)$ to $end(V)$, plus one new edge e from $end(U)$ to $end(V)$ costing

$$c(e) = \frac{C^* - f(U,V)}{2} + h(end(U), end(V)).$$

$c(e)$ is positive because $h(end(U), end(V)) \geq 0$ and $C^* > f(U,V)$. This new edge creates a solution path UV^{-1} whose total cost is

$$\frac{C^* + f(U,V)}{2},$$

which is strictly less than C^* (because $f(U,V) < C^*$). The new edge is therefore an essential part of any optimal solution to I'.

We show that $I' \in I_{AD}$, i.e. that h is bi-admissible on G'. Let x and y be any two states in G' and let W be any acyclic forward path in G' from x to y. We claim that $h(x,y) \leq c(W)$. If W does not contain the new edge e, the claim trivially follows from the bi-admissibility of h on G. Suppose W does contain e, i.e. $W = XY$ for some forward paths X, from x to $end(U)$, and Y, from $end(V)$ to y. Denoting the distance from x to y in G by $d_G(x,y)$, we have

$$\begin{aligned}
h(x,y) &\leq d_G(x, end(U)) + h(end(U), y) \\
&\quad \text{(by part (1) of bi-consistency of } h \text{ on } G) \\
&\leq c(X) + h(end(U), y) \\
&\quad \text{(since } X \text{ is a path from } x \text{ to } end(U) \text{ in } G) \\
&\leq c(X) + \\
&\quad d_G(end(V), y) + h(end(U), end(V)) \\
&\quad \text{(by part (2) of bi-consistency of } h \text{ on } G) \\
&\leq c(X) + c(Y) + h(end(U), end(V)) \\
&\quad \text{(since } Y \text{ is a path from } end(V) \text{ to } y \text{ in } G) \\
&< c(X) + c(Y) + c(e) \\
&\quad \text{(by definition, } c(e) > h(end(U), end(V))) \\
&= c(W).
\end{aligned}$$

This proves that h is bi-admissible on G', i.e., $I' \in I_{AD}$.

Because B is DXBB it will behave exactly the same on I' as it did on I. In particular it will not forward expand $end(U)$ and it will not backward expand $end(V)$, so it will not discover the edge e, and will incorrectly return $B(I)$ as the optimal solution for I'. Hence, B is not admissible. □

5 An Optimally Efficient Front-to-Front Bidirectional Search Algorithm

Algorithm 2 gives the pseudocode for the front-to-front bidirectional search algorithm BDS1 (Eckerle and Ottmann 1994). It is a standard front-to-front algorithm except that its Open and Closed lists contain paths, not nodes (forward paths in $Open_F$ and $Closed_F$, backward paths in $Open_B$ and $Closed_B$). On each iteration it selects a path pair (U,V) for expansion for which $f(U,V)$ is minimum (line 8) and then chooses a search direction (line 9). It can be computationally expensive to find a path pair with minimum f-value; an alternative method avoiding this expense is implemented in BDS2 (Eckerle and Ottmann 1994) and SFBDS (Felner et al. 2010) (see Section 8). If there exists a path from $start$ to $goal$, BDS1 is guaranteed to terminate (line 7) before $Open_F$ or $Open_B$ become empty; line 24 is therefore only reached if there is no path from $start$ to $goal$. Eckerle and Ottmann (1994) proved that BDS1 is admissible.

BDS1 can be rewritten in the form of Algorithm 1 and is DXBB. In this section, we define a specific class of problem instances, denoted by I^-_{CON}, and prove that BDS1 is optimally efficient on instances in that class in the following sense: If, in solving any problem instance $I \in I^-_{CON}$, BDS1 expands state pair (u,v), then *any* admissible DXBB algorithm must expand (u,v) when solving I. Recall that expanding state pair (u,v) means to either forward expand u or backward expand v. Full details of the optimality proof are omitted due to space constraints and are available from the authors on request.

Definition 4. *State pair (u,v) is* FF-surely expanded *(or just* surely expanded *when front-to-front is clear from the context) if there exist a forward path (not necessarily optimal) U from $start$ to u and a backward path (not necessarily optimal) V from $goal$ to v such that*

$$c(U,i) + c(V,j) + h(U_i, V_j) < C^*$$

for all $0 \leq i \leq |U|$ and all $0 \leq j \leq |V|$.

Algorithm 2: Pseudocode for BDS1

```
   Input: (start, goal, c, expand_F, expand_B), h
   Output: cost of the cheapest path from start to goal
1  Open_F := {λ_F}; Open_B := {λ_B};
2  Closed_F := ∅; Closed_B := ∅;
3  C := ∞;
4  while (Open_F ≠ ∅) and (Open_B ≠ ∅) do
5      fmin := min {f(U,V) | U ∈ Open_F, V ∈ Open_B};
6      if C ≤ fmin then
7          return C
8      choose (U,V) ∈ Open_F × Open_B for which
       f(U,V) = fmin
9      select one of the paths in (U,V) to expand
10     if U was chosen for expansion then
11         // Expand U in the forward direction
12         move U from Open_F to Closed_F
13         for each W ∈ expand_F(U) do
14             if ∃X ∈ Open_F ∪ Closed_F such that
               end(X) = end(W) then
15                 if c(X) ≤ c(W) then
16                     continue
17                 else
18                     remove X from Open_F ∪ Closed_F
19             add W to Open_F
20             if ∃Y ∈ Open_B ∪ Closed_B such that
               end(Y) = end(W) then
21                 C := min(C, c(WY^-1))
22     else
23         // Expand V in the backward direction, analogously
24 return ∞
```

The following lemma is easily proven from the definitions of "bi-consistent" and "FF-surely expanded".

Lemma 2. *If front-to-front heuristic h is bi-consistent, then state pair (u, v) is FF-surely expanded if and only if there exist an optimal forward path U from $start$ to u and an optimal backward path V from $goal$ to v such that $f(U, V) < C^*$.*

This lemma allows us to produce an alternative form of Theorem 1 by replacing "If U is an optimal forward path and V is an optimal backward path such that $U_0 = start$, $V_0 = goal$, and $f(U, V) < C^*$ then..." by "If state pair (u, v) is FF-surely expanded then ...". Making this substitution in the statement of Theorem 1, we get

Theorem 3. *Let $(G, h) \in I_{CON}$. If state pair (u, v) is FF-surely expanded then, in solving problem instance (G, h), any admissible DXBB algorithm must expand (u, v).*

Definition 5. *A problem instance (G, h) is* non-pathological *if there exists an optimal solution path U such that $h(U_i, U_j) < d(U_i, U_j)$ for all i, j such that $0 \leq i < j \leq |U|$. Such a path is called a* non-pathological solution *for (G, h). The set of non-pathological instances in I_{CON} is denoted I^-_{CON}.*

Theorem 4. *If problem instance $(G, h) \in I^-_{CON}$ then, in solving (G, h), BDS1 will not expand any state pair that is not FF-surely expanded.*

As an immediate consequence of Theorems 3 and 4, we obtain that BDS1 is optimally efficient ("0-optimal" as defined by Dechter & Pearl (p. 521)) over the set of admissible DXBB algorithms relative to I^-_{CON}:

Theorem 5. *If problem instance $(G, h) \in I^-_{CON}$ then, in solving (G, h), every admissible DXBB algorithm expands a superset of the state pairs expanded by BDS1.*

Note that expanding a superset of the state pairs expanded by BDS1 does not necessarily require as many single state expansions as made by BDS1. This is because a state pair (u, v) is expanded as soon as either u is forward expanded or v is backward expanded. Thus, a specific set of state pairs can typically be expanded in more than one way, using sets of single state expansions that differ in cardinality. We revisit this issue in Section 8.

6 A Sufficient Condition for Node Expansion for Front-to-End Bidirectional Search

Front-to-end bidirectional heuristic search algorithms interleave two separate searches, a search forward from $start$ and a search backward from $goal$. In the notation of Section 3, this is a restriction on the `Choose` function to return, on any given iteration, either (i) Dir set to "Forward" and a path pair of the form (U, λ_B), or (ii) Dir set to "Backward" and a path pair of the form (λ_F, V).

Definition 6. *A* front-to-end heuristic *maps an individual state in G to a non-negative real number or to ∞. Front-to-end heuristic h_F is* forward admissible *iff $h_F(u) \leq d(u, goal)$ for all u in G and is* forward consistent *iff $h_F(u) \leq d(u, u') + h_F(u')$ for all u and u' in G. Front-to-end heuristic h_B is* backward admissible *iff $h_B(v) \leq d(start, v)$ for all v in G and is* backward consistent *iff $h_B(v) \leq d(v', v) + h_B(v')$ for all v and v' in G.*

Forward consistent front-to-end heuristic h_F is forward admissible iff $h_F(goal) = 0$, and backward consistent front-to-end heuristic h_B is backward admissible iff $h_B(start) = 0$.

Instead of a front-to-front heuristic $h(u, v)$, front-to-end bidirectional heuristic search algorithms use two front-to-end heuristics, h_F and h_B, with h_F used to guide the forward search and h_B used to guide the backward search. Only a few small modifications to our terminology and notation in Sections 2 and 3 are needed to accommodate this difference. The main change is to replace all occurrences of h with a pair of front-to-end heuristics. For example, a problem instance is defined by specifying a state space G, a start state ($start$), a goal state ($goal$), and two front-to-end heuristics, h_F and h_B, that are applicable to states in G.

I_{AD} is now the set of solvable problem instances in which h_F is forward admissible and h_B is backward admissible. I_{CON} is the subset of I_{AD} in which h_F is forward consistent and h_B is backward consistent.

For any forward path U with $U_0 = start$ define

$$f_F(U) = c(U) + h_F(end(U)),$$

and for any backward path V with $V_0 = goal$ define

$$f_B(V) = c(V) + h_B(end(V)).$$

Theorem 6. *Let $I = (G, h_F, h_B) \in I_{CON}$ have an optimal solution cost of C^*. If U is an optimal forward path and V is an optimal backward path such that $U_0 = start$, $V_0 = goal$, and:*
(1) $f_F(U) < C^*$
(2) $f_B(V) < C^*$
(3) $c(U) + c(V) < C^*$
then, in solving problem instance I, any admissible DXBB bidirectional front-to-end search algorithm must expand $(end(U), end(V))$.

Proof. We prove the contrapositive. Suppose $I, U,$ and V satisfy the premises of the theorem, and that B is a DXBB bidirectional front-to-end search algorithm that solves I correctly (returns a path $B(I)$ costing C^*) without forward expanding $end(U)$ or backward expanding $end(V)$. Then a new problem instance $I' = (G', h'_F, h'_B) \in I_{AD}$ can be constructed having an optimal solution strictly cheaper than C^* on which B also returns path $B(I)$ (costing C^*), thereby showing that B is not an admissible algorithm.

$I' = (G', h'_F, h'_B)$ is defined as follows: $h'_F = h_F$, $h'_B = h_B$, and G' has all the vertices in G and all the edges in G except the edge, if there is one, from $end(U)$ to $end(V)$, plus one new edge e from $end(U)$ to $end(V)$ costing

$$c(e) = \max\{\quad h_F(end(U)) - c(V),\\ h_B(end(V)) - c(U),\\ \tfrac{1}{2}(C^* - (c(U) + c(V)))\quad\}$$

$c(e)$ is positive because $C^* > c(U) + c(V)$. This new edge creates a solution path UV^{-1} whose total cost is $c(U) +$

$c(V)+c(e)$, which is equal to $\max(f_F(U), f_B(V), \frac{1}{2}(C^* + c(U) + c(V)))$. This is strictly less than C^* because of the theorem's premises, so the new edge is an essential part of any optimal solution to I'.

We begin by proving that $I' \in I_{AD}$, i.e. that h_F is forward admissible on G' and h_B is backward admissible on G'. We give the proof for h_F, the proof for h_B is analogous. Let x be any state in G' and let W be any acyclic forward path in G' from x to $goal$. We claim that $h_F(x) \leq c(W)$. If W does not contain the new edge e, the claim trivially follows from the forward admissibility of h_F on G. Hence, assume W contains e, i.e. $W = YZ$ for some forward paths Y, from x to $end(U)$, and Z, from $end(V)$ to $goal$. Using $d_G(u, v)$ to denote the distance from u to v in G, we have

$$\begin{aligned}
h_F(x) &\leq d_G(x, end(U)) + h_F(end(U)) \\
&\quad \text{(because } h_F \text{ is forward consistent on } G) \\
&\leq c(Y) + h_F(end(U)) \\
&\quad \text{(because } Y \text{ is a path from } x \text{ to } end(U)) \\
&\leq c(Y) + c(e) + c(V) \\
&\quad \text{(by definition, } c(e) \geq h_F(end(U)) - c(V)) \\
&= c(Y) + c(e) + c(V^{-1}) \\
&\quad \text{(because } c(V^{-1}) = c(V)) \\
&\leq c(Y) + c(e) + c(Z) \\
&\quad \text{(by optimality of } V^{-1}) \\
&= c(W)\,.
\end{aligned}$$

Hence h_F is forward admissible on G'. By an analogous proof, h_B is backward admissible on G' and thus $I' \in I_{AD}$.

Because B is DXBB it will behave the same on I' as it did on I. In particular it will neither forward expand $end(U)$ nor backward expand $end(V)$, will thus not discover the edge e, and will incorrectly return $B(I)$ as an optimal solution for I'. Hence, B is not an admissible search algorithm. □

Dechter & Pearl proved (their Theorem 8) that every admissible unidirectional search algorithm must expand every state surely expanded by A* when the given heuristic is consistent.[4] We will now show that this theorem is a special case of our Theorem 6.

Unidirectional heuristic search algorithms are a special case of front-to-end bidirectional heuristic search algorithms in which $h_B(u) = 0$ for all u (we denote this function by "0") and the `Choose` function always returns Dir set to "Forward" and a path pair of the form (U, λ_B).[5]

Specializing Theorem 6 with these restrictions we obtain the following, which is equivalent to Dechter & Pearl's Theorem 8.

[4]Using our notation, a state u is surely expanded by A* if it can be reached from $start$ by a forward path $U = U_0, U_1, \ldots U_n$ ($U_0 = start$, $U_n = u$) such that $f_F(U_0, \ldots, U_i) < C^*$ for all $i \in [1, n]$. When the heuristic is consistent this condition simplifies to $f_F(U) < C^*$.

[5]This defines a undirectional *forward* search algorithm. A unidirectional *backward* search algorithm is defined analogously.

Theorem 7. *Let $I = (G, h_F, 0) \in I_{CON}$ have an optimal solution cost of C^*. If U is an optimal forward path such that $U_0 = start$ and $f_F(U) < C^*$ then, in solving problem instance I, any admissible DXBB unidirectional search algorithm must forward expand $end(U)$.*

Definition 7. *For a problem instance $I \in I_{CON}$, state pair (u, v) is* FE-surely expanded *(or just* surely expanded *if front-to-end is understood from the context) if there exist paths U and V satisfying the premises of Theorem 6 with $end(U) = u$ and $end(V) = v$. State pair (u, v) is* covered *by search algorithm A if, in solving I, A expands the state pair (u, v).*

7 Sufficient Conditions Compared

Theorems 1 and 6 both characterize state pairs that must be expanded by an admissible DXBB bidirectional search algorithm when its heuristic(s) are admissible and consistent. Theorem 1 describes the state pairs that must be expanded by a front-to-front algorithm if its heuristic, $h(u, v)$, is bi-admissible and bi-consistent, while Theorem 6 describes the state pairs that must be expanded by a front-to-end algorithm if its heuristics, h_F and h_B, are forward and backward admissible and consistent, respectively.

A natural question is how these two sufficient conditions compare: are they equivalent, does one imply the other, etc.? To answer this question fairly, one should ensure that the algorithms being compared are "equally informed". Our analysis in this section assumes both are given the same front-to-front heuristic, $h(u, v)$. The front-to-end algorithm uses this to define $h_F(u) = h(u, goal)$ and $h_B(v) = h(start, v)$.

The following theorem establishes that, if both algorithms are given the same bi-admissible, bi-consistent front-to-front heuristic $h(u, v)$, the conditions in Theorem 1 logically imply the conditions in Theorem 6, i.e. if a path pair satisfies the conditions in Theorem 1 it will also satisfy the conditions in Theorem 6.

Theorem 8. *Let $h(u, v)$ be bi-consistent and define $h_F(u) = h(u, goal)$ and $h_B(v) = h(start, v)$. If U is an optimal forward path and V is an optimal backward path such that $U_0 = start$, $V_0 = goal$, and $f(U, V) = c(U) + c(V) + h(end(U), end(V)) < C^*$, then:*

(1) $f_F(U) < C^*$
(2) $f_B(V) < C^*$
(3) $c(U) + c(V) < C^*$.

Proof. We show $f(U, V) \geq \max\{f_F(U), f_B(V), c(U) + c(V)\}$. The following proves that $f(U, V) \geq f_F(U)$.

$$\begin{aligned}
f_F(U) &= c(U) + h(end(U), goal) \\
&\leq c(U) + d(end(V), goal) \\
&\quad + h(end(U), end(V)) \\
&\quad \text{(part (2) of the definition of bi-consistent)} \\
&= c(U) + c(V) + h(end(U), end(V)) \\
&\quad \text{(because } V \text{ is optimal)} \\
&= f(U, V)\,.
\end{aligned}$$

The proof that $f(U, V) \geq f_B(V)$ is analogous. $f(U, V) \geq c(U) + c(V)$ is true because $h(end(U), end(V)) \geq 0$. □

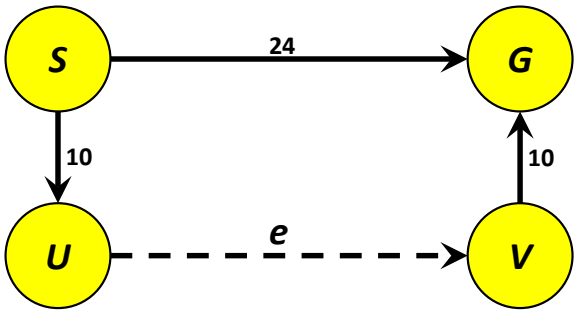

Figure 1: Problem instance on which admissible front-to-end algorithms must expand more nodes than admissible front-to-front algorithms. The heuristic values (h_F, h_B, and h) are given in Table 1.

r	$h_F(r)$	$h_B(r)$	$h(r,c)$			
			S	U	V	G
S	10	0	0	10	10	10
U	10	10	∞	0	9	10
V	10	10	∞	∞	0	10
G	0	10	∞	∞	∞	0

Table 1: The first three columns show the states (S, U, V, G) and their h_F and h_B values based on the bi-consistent front-to-front heuristic $h(r, c)$ shown in the final four columns. Here r and c are the states defining the row and column, resp., e.g., $h(U, V)$ is the entry (9) in row U column V.

The opposite implication does not hold; the set of path pairs that must be expanded according to Theorem 1 may be a proper subset of the set of path pairs that must be expanded according to Theorem 6.

An example is given in Figure 1. There are four states (S (the start state), U, V, and G (the goal state)) connected by the three weighted, directed edges shown as solid arrows. The dashed arrow labelled "e" is an edge that might or might not exist. As there are unique paths to U and V we also use these states' names to refer to the unique paths to them. To simplify the discussion, we assume $e > 0$ ($e = \infty$ is permitted, meaning the edge does not exist). Now, consider the state/path pair (U, V). In the front-to-front setting, $f(U, V) = 29$ (because $h(U, V) = 9$, see Table 1) and since $C^* \leq 24$, the pair (U, V) does not satisfy the conditions of Theorem 1. By contrast, in the front-to-end setting, $f_F(U) = f_B(V) = c(U) + c(V) = 20 < C^*$ (because $e > 0$) and therefore (U, V) *does* satisfy the conditions of Theorem 6 and must be expanded by any admissible front-to-end algorithm.

The reason a front-to-end algorithm must expand one of U or V is because there could be a sufficiently cheap ($e < 4$) edge connecting U to V so that the optimal path from S to G includes this edge. For example, given a problem instance I based on this example in which the dashed edge does not exist, the construction of I' in the proof of Theorem 6 would set $e = 2$ thereby creating a path from S to G costing only 22. If the algorithm must return optimal solutions whenever h_F is forward admissible and h_B is backward admissible, the only way it can be certain there is not a path from S to G costing less than 24 is to expand one of U or V to see if such an edge exists.

The same reasoning does not apply to a front-to-front algorithm because the definition of bi-admissibility is so much stronger than the definition of forward/backward admissibility. A front-to-front heuristic that is bi-admissible comes with enough guarantees ($h(u, v) \leq d(u, v)$ for all u and v) that it can be used for pruning states that cannot be pruned if the only guarantees are those of front-to-end forward/backward admissibility. In this particular example $h(U, V) = 9$ implies that $e \geq 9$, guaranteeing that the optimal path from S to G cannot pass through U and V without having to expand either of them. Only the front-to-front algorithm has this information, the front-to-end algorithm has no non-trivial lower bound on e.

8 Limitations

In this section we discuss the limitations of the analysis presented in this paper, section by section.

Section 2 (Notation and Terminology). The only notable limitation here is the simplifying assumption, which is not made by Dechter & Pearl, that the goal is a single state rather than a set of states specified by a goal condition. We do not believe this is a fundamental limitation, but there are technicalities that we preferred to avoid in this initial analysis.

Section 3 (Algorithmic Assumptions). The three defining properties of DXBB algorithms are central to the proofs of Theorems 1 and 6, and Dechter & Pearl's Theorem 8, but many of today's systems do not have all of these properties.
Deterministic. This requirement excludes any randomized algorithm, for example one that breaks ties randomly (Asai and Fukunaga 2016) or that uses a randomized method to diversify the kind of nodes selected for expansion (Imai and Kishimoto 2011; Xie, Müller, and Holte 2014).
Expansion-based. Although the majority of today's search and planning systems are expansion-based (i.e. they search in the state space by expanding one state at a time), there are notable exceptions such as symbolic search (Edelkamp, Kissmann, and Torralba 2015), SAT-based planning (Rintanen 2012), and planning based on Integer Linear Programming (Yu and LaValle 2016).
Black Box. In *domain-independent* planning and search, the state space and goal are given in a declarative language (e.g. PDDL[6] or SAS$^+$ (Bäckström 1992)). In such a representation, the differences between problem instances I and I' in the proofs of Theorems 1 and 6, and Dechter & Pearl's Theorem 8, would be readily apparent without expanding any states, there is no reason to expect domain-independent search algorithms to behave identically on the two instances.

Theorems 1 and 6 (Sections 4 and 6). Both these theorems describe which *state pairs* must be expanded, not which specific states must be expanded. To expand state pair (u, v)

[6] http://icaps-conference.org/ipc2008/deterministic/PddlResources.html

means to forward expand u or backward expand v. The "or" here is a major weakness in these theorems. For example, suppose there are 100 pairs these theorems say must be expanded, and they are all of the form (u, v_i) $(1 \leq i \leq 100)$. All 100 pairs can be expanded by expanding u once in the forward direction. Alternatively, these pairs could be expanded by backward expanding each v_i individually. The set of pairs expanded is the same in either case, but the number of calls to an $expand$ function is not, yet BDS1 would be optimally efficient whichever of these options it executed. By contrast, the analogous theorem by Dechter & Pearl (their Theorem 8) defines a specific set of states that must be expanded, there are no options.

Section 5 (Optimal Efficiency of BDS1).
Non-pathological Problem Instances. This concept was introduced by Dechter & Pearl, and is critical to their proof of A*'s optimal efficiency, since on such instances A* expands only those states that absolutely must be expanded. There are common circumstances where no "reasonable" instances are non-pathological. For example, consider any state space in which all edges cost 1. One should never use a heuristic for which $h_F(u) = 0$ for a non-goal state u since $h_F(u) = 1$ is always a safe and superior substitute. If this is done, all states that are one edge away from $goal$ have $h_F(u) = d(u, goal)$, so the instance is not non-pathological.

In the bidirectional setting, the same issue arises, but it is somewhat magnified because, in state spaces with all edges costing 1 and a heuristic h that returns integer values, a non-pathological instance must have $h(u, v) = 0$ for every edge (u, v) on every optimal solution path.
Defining Optimal Efficiency in Terms of Sets of States. Dechter & Pearl defined optimal efficiency in terms of the set of states expanded, and we have generalized this, for bidirectional search, to the set of state pairs expanded. As noted above, in the bidirectional setting there can be a large difference between the number of calls to an $expand$ function even when the set of state pairs expanded is the same. Morever, the expansion of states is not necessarily the dominant factor in determining the run-time efficiency of a bidirectional search algorithm. A major challenge for many front-to-front systems, for example, is the computation involved in finding a path pair (U, V) that minimizes $f(U, V)$.

This challenge for front-to-front systems was overcome in systems BDS2 (Eckerle and Ottmann 1994) and SFBDS (Felner et al. 2010). The key idea is to place path pairs in an $Open$ list sorted by $f(U, V)$. When a pair (U, V) is expanded the pairs in S_t are added to $Open$. This makes BDS2 and SFBDS as efficient as A* in terms of $Open$ list operations but comes at the price of having to expand the same path multiple times (because (U, V) and (U, W) are separate entries in this $Open$ list). This occurred so frequently in Lippi et al. (2012)'s experiments that they introduced a technique (successor caching) to minimize its computational cost. Again, the set of path pairs expanded is not a particularly good way to define optimal efficiency.

9 Practical Implications

Although this paper is purely theoretical, it has immediate practical consequences. The sufficient conditions for front-to-end search presented here (Theorem 6) have directly inspired a new admissible DXBB front-to-end bidirectional search algorithm, NBS, which comes with strong formal guarantees and performs well in practice (Chen et al.). The formal guarantees are about the number of states expanded to cover all the surely expanded state pairs: (i) NBS is near-optimal in the sense that in the worst case it expands at most twice as many states as the theoretical minimum, and (ii) no other admissible DXBB front-to-end algorithm has a better worst case. The performance of NBS on standard benchmark domains is comparable to the performance of MMe (Sharon et al. 2016) and BS* (Kwa 1989). NBS outperforms A* on hard problem instances or when weak heuristics are being used.

10 Conclusions

This work establishes sufficient conditions for node expansion by deterministic expansion-based black-box bidirectional heuristic search algorithms. We restrict our study to the case of consistent heuristics and admissible algorithms, as was done in Dechter and Pearl's analogous analysis of the 0-optimality of A*. Our main results include sufficient conditions for node expansion for both front-to-front and front-to-end systems, and a proof of the 0-optimality (optimal efficiency) of front-to-front bidirectional search algorithm BDS1 on a certain class ("non-pathological") of problem instances. We also proved that the sufficient condition for node expansion derived by Dechter & Pearl for unidirectional systems is a special case of our sufficient condition for front-to-end bidirectional systems.

A different kind of contribution is to clearly identify the limitations of this analysis. Several of the limitations we identified are shared by Dechter & Pearl's analysis, others arise because of the special nature of bidirectional search.

Despite these limitations, our work provides a useful basis for the analysis of bidirectional search algorithms. For instance, it would be interesting to revisit the definition of non-pathological instances to see whether it can be relaxed so that BDS1 (or some other bidirectional search algorithm) is optimally efficient on a broader class of instances. Another interesting direction for future work is to redo this analysis for algorithms that are specifically designed to operate with consistent heuristics, such as BS*.

11 Acknowledgements

We thank Ariel Felner and the anonymous reviewers for many suggestions that improved the paper. Financial support for this research was in part provided by Canada's Natural Sciences and Engineering Research Council (NSERC). This material is based upon work supported by the National Science Foundation under Grant No. 1551406.

References

Asai, M., and Fukunaga, A. S. 2016. Tiebreaking strategies for A* search: How to explore the final frontier. In *Proc. 30th AAAI Conference on Artificial Intelligence*, 673–679.

Bäckström, C. 1992. Equivalence and tractability results for SAS$^+$ planning. In *Proceedings of the 3rd International Conference on Principles on Knowledge Representation and Reasoning*, 126–137.

Chen, J.; Holte, R. C.; Zilles, S.; and Sturtevant, N. R. Front-to-end bidirectional heuristic search with near-optimal node expansions. *http://arxiv.org/abs/1703.03868*.

de Champeaux, D. 1983. Bidirectional heuristic search again. *J. ACM* 30(1):22–32.

Dechter, R., and Pearl, J. 1985. Generalized best-first search strategies and the optimality of A*. *J. ACM* 32(3):505–536.

Eckerle, J., and Ottmann, T. 1994. An efficient data structure for bidirectional heuristic search. In *ECAI*, 600–604.

Edelkamp, S.; Kissmann, P.; and Torralba, Á. 2015. BDDs strike back (in AI planning). In *Proc. 29th AAAI Conference on Artificial Intelligence*, 4320–4321.

Felner, A.; Moldenhauer, C.; Sturtevant, N. R.; and Schaeffer, J. 2010. Single-frontier bidirectional search. In *Proc. 24th AAAI Conference on Artificial Intelligence*.

Holte, R. C.; Felner, A.; Sharon, G.; and Sturtevant, N. R. 2016. Bidirectional search that is guaranteed to meet in the middle. In *Proc. 30th AAAI Conference on Artificial Intelligence*.

Imai, T., and Kishimoto, A. 2011. A novel technique for avoiding plateaus of greedy best-first search in satisficing planning. In *Proc. 25th AAAI Conference on Artificial Intelligence*.

Kwa, J. B. H. 1989. BS*: An admissible bidirectional staged heuristic search algorithm. *Artificial Intelligence* 38(1):95–109.

Lippi, M.; Ernandes, M.; and Felner, A. 2012. Efficient single frontier bidirectional search. In *Proceedings of the Fifth Annual Symposium on Combinatorial Search, SoCS*.

Manzini, G. 1995. BIDA*: an improved perimeter search algorithm. *Artificial Intelligence* 75(2):347–360.

Pohl, I. 1971. Bi-directional search. In Meltzer, B., and Michie, D., eds., *Machine Intelligence*, volume 6. Edinburgh University Press. 127–140.

Rintanen, J. 2012. Planning as satisfiability: Heuristics. *Artificial Intellligence* 193:45–86.

Sharon, G.; Holte, R. C.; Felner, A.; and Sturtevant, N. R. 2016. Extended abstract: An improved priority function for bidirectional heuristic search. *Symposium on Combinatorial Search (SoCS)* 139–140.

Xie, F.; Müller, M.; and Holte, R. 2014. Adding local exploration to greedy best-first search in satisficing planning. In *Proc. 28th AAAI Conference on Artificial Intelligence*, 2388–2394.

Yu, J., and LaValle, S. M. 2016. Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. *IEEE Trans. Robotics* 32(5):1163–1177.