Front-to-Front Heuristic Search for Satisficing Classical Planning

Ryo Kuroiwa and **Alex Fukunaga**
Graduate School of Arts and Sciences, The University of Tokyo
mhgeoe@gmail.com, fukunaga@idea.c.u-tokyo.ac.jp

Abstract

Although symbolic bidirectional search is successful in optimal classical planning, state-of-the-art satisficing planners do not use bidirectional search. Previous bidirectional search planners for satisficing planning behaved similarly to a trivial portfolio, which independently executes forward and backward search without the desired "meet-in-the-middle" behavior of bidirectional search where the forward and backward search frontiers intersect at some point relatively far from the forward and backward start states. In this paper, we propose Top-to-Top Bidirectional Search (TTBS), a new bidirectional search strategy with front-to-front heuristic evaluation. We show that TTBS strongly exhibits "meet-in-the-middle" behavior and can solve instances solved by neither forward nor backward search on a number of domains.

1 Introduction

Heuristic search algorithms such as A* [Hart *et al.*, 1968] and Greedy Best-First Search (GBFS) [Doran and Michie, 1966] are widely used in state-of-the-art solvers for classical planning. Many current planners use *forward search* algorithms which start at the initial state and seek a path to a goal state. However, it is possible to search backward from a goal state to a start state by *regression*, the process of computing the possible predecessor states for a state (or a set of states). Furthermore, it is possible to perform *bidirectional search*, which simultaneously searches in both forward and backward directions. Bidirectional heuristic search, particularly focusing on admissible algorithms which guarantee optimal-cost solutions, has recently been a very active area of research in the search community [Sturtevant and Felner, 2018]. In admissible (optimal) planning, symbolic bidirectional heuristic search, which applies actions to sets of states, is one of the state-of-the-art methods [Torralba *et al.*, 2016].

However, the majority of recent work in satisficing planning has focused on forward, state-space search. Recently, Alcázar *et al.* (2013) bridged the gap between forward and backward search for satisficing classical planning, enabling some techniques originally developed for forward search to be used in backward search. They also extended their techniques to bidirectional planning, but found that although their bidirectional search could improve upon a unidirectional search, the improvements could be attributed to a portfolio effect – domains which are suited for forward search are solved by the forward search component, while domains which are suited for backward search are solved by the backward search component. They found little evidence of the forward and backward search frontiers "meeting in the middle", and concluded that it was unlikely that bidirectional satisficing planning would outperform a simpler portfolio consisting of a forward search component and a backward search component [Alcázar *et al.*, 2014].

We revisit bidirectional search for satisficing classical planning. We propose Top-to-Top Bidirectional Search (TTBS), a new front-to-front search strategy which expands nodes that have the smallest estimated distance to the opposite frontier. We show that on a number of standard IPC benchmark domains, TTBS outperforms both forward and backward search by successfully "meeting in the middle", as opposed to a trivial portfolio effect. We show that a portfolio combining TTBS, forward search, and backward search outperforms forward search, backward search, as well as a forward/backward portfolio.

2 Background and Related Work

2.1 Bidirectional Search

In bidirectional search, search starts at both the start state and the goal state (or, in the case of regression planning, a set of goal states derived from the goal condition). Standard bidirectional searches have $Open_f$ and $Closed_f$ for forward direction, and $Open_b$ and $Closed_b$ for the backward search. The search succeeds and terminates when the forward and backward search frontiers intersect – for optimal search, it is further necessary to guarantee that the intersection results in the shortest path.

In principle, if the search frontiers tend to grow wider as they get further from their start points, bidirectional search can save exponential time and space over unidirectional search if the search frontiers "meet in the middle", i.e., the intersection occurs somewhere between the start and goal [Sturtevant and Felner, 2018]. In brute-force bidirectional search without a heuristic evaluation function, this behavior is relatively easy to achieve because both forward and backward search never expand states deeper than the solution midpoint [Barker and Korf, 2015]. On the other hand, for heuristic search, the forward and backward frontiers often pass by each other without intersecting, and the bidirectional search merely behaves like a portfolio consisting of independent forward and backward search components, terminating when either the forward search reaches a goal or the backward search reaches the start state. The recent development of MM, an algorithm which guarantees meet-in-the-middle behavior and obtains state-of-the-art performance on a number of domains, has sparked a significant amount of activity on admissible, bidirectional search algorithms [Holte *et al.*, 2017].

The most widely studied class of bidirectional search algorithms, including MM, uses a *front-to-end* strategy with two heuristic functions, where nodes in $Open_f$ are evaluated according to h_{goal}, which estimates their distance to a goal, while nodes in $Open_b$ are evaluated according to h_{start}, which estimates their distance to the start state.

In contrast, *front-to-front* algorithms evaluate states according to how close they are to the opposite search frontier. For example, in principle, by computing the heuristic $h(u, v)$ between all pairs of states u and v in the frontiers, it would be possible to expand nodes in order of how close they seem to the opposite search frontier, which possibly leads to meet-in-the-middle behavior.

2.2 Backward Search for Classical Planning

In a classical planning task, the initial state, the set of actions, and the goal condition are given. The solution of a classical planning task is a sequence of actions which makes the initial state transitions to a state satisfying the goal condition. The standard SAS+ formalization [Bäckström and Nebel, 1995] of a classical planning task represents states using multi-valued variables. Each action a in the planning task has a precondition pre(a) and effects eff(a), as partial value assignments to the state variables. The goal condition is also given as a partial value assignment. An action a is applicable in a state s iff pre$(a) \subseteq s$, and if a is applied to the state, s transitions to a state s' which is the same as s except for that eff$(a) \subseteq s'$. Considering each state as a node and each state transition by applying an action as an edge, forward search can solve a classical planning task.

For some search problems such as the 15-puzzle, implementing backward search is as straightforward as forward search, because a goal state is explicitly given and state transitions are trivially invertible. However, for planning tasks where a goal condition is given instead of an explicit goal state, backward search is more complex than forward search. First, multiple goal states can satisfy the goal condition. Second, given an action a and a state s, there can be multiple predecessors of s for which applying a results in s. Thus, in contrast to forward search which focuses on individual states, backward search (and therefore bidirectional as well) may need to perform search among sets of states in the backward direction.

Regression planning for satisficing planning has a long history going back to the original work on theorem-proving based approaches by Green (1969), as well as the seminal work on HSPr by Bonet and Geffner (2001). In regression planning, starting from the goal condition, backward search generates sets of possible predecessor states until a set including the initial state is found. Recent work on heuristic search approaches has focused primarily on forward search. Alcázar *et al.* (2013) bridged the gap between forward and backward approaches by proposing several techniques for handling sets of states in regression search similarly to explicit forward search spaces. For regression, they introduced an additional, *undefined* value for each variable, allowing regression planning to treat sets of states as search nodes, e.g., the goal node is represented as a value assignment where values are *undefined* if they are not specified in the goal condition. An action a is applicable in a state (or a set of states represented by a value assignment including *undefined* values) S in regression, if eff$(a) \subseteq S$ and pre$(a) \subseteq S$ for variables not specified in eff(a). When a is applied to S, the variables are changed so that they satisfy pre(a). These techniques were implemented in FDr [Alcázar *et al.*, 2013], a regression planner built on top of the forward search-based Fast Downward planning system (FD) [Helmert, 2006], and it was shown that FDr can directly use several heuristic functions used in FD, obtaining competitive performance on some domains.

2.3 biFD: A Bidirectional Classical Planner

Alcázar *et al.* (2014) developed biFD, a bidirectional search for satisficing classical planning and evaluated several bidirectional search strategies and methods for detecting forward-backward frontier intersection. biFD interleaves front-to-end forward search and regression, allocating search effort in the direction which has used less time so far. A variation of biFD using a Binary Decision Diagram (BDD) [Bryant, 1986] for frontier detection was investigated. They also investigated front-to-front variants which maintain Backward Generated Goals (BGGs), a set of states in the opposite frontier (one variant used all the expanded states as BGGs, another used 1000-2000 states among the expanded states). In the BGG-based front-to-front search, the h-value of a state s is the estimated cost from s to the state with the lowest $h_{\max}$ [Bonet and Geffner, 2001] among BGGs. To detect the intersection of frontiers, biFD with BGGs compares a state vs. all states in the opposite BGGs. This intersection detection incur significant overhead. In addition, the state with the lowest $h_{\max}$ among BGGs is not necessarily a good target (it may not be close to the opposite frontier), and may not be sufficient to induce "meet-in-the-middle" behavior.

Although biFD achieved better overall coverage than regression planning, they observed that the intersection of frontiers almost always occurred very close to the opposite goal, i.e., meet-in-the-middle behavior was not observed, and the improvement of biFD over FDr was ascribed to a trivial portfolio effect.

2.4 Single-Frontier Bidirectional Search

Single-Frontier Bidirectional Search (SFBS) is a front-to-front heuristic search method proposed by Felner *et al.* (2010). Instead of separate *Open* and *Closed* for forward and backward search, SFBS uses a single queue ordered according to $h(u, v)$, shifting focus between forward and backward expansions according to a jump policy.

While the idea of using a single *Open* prioritized according to $h(u, v)$ is elegant, it poses a problem. In standard unidirectional search, as well as other bidirectional searches such as MM, for a state s, its heuristic value $h(s)$ only needs to be evaluated at most once (in algorithms where s can be regenerated, $h(s)$ can be cached and reused). In contrast, in SFBS, a search *node* corresponds to a pair of states (x, y), and there are multiple search nodes which contain states x and y. Thus, SFBS requires the evaluation of $h(s, w)$ for multiple w (in the worst-case, all nodes in the opposite frontier), which can be a major overhead especially when h is expensive to compute. Similarly, SFBS also can expand multiple search nodes (s, w) for many states w.

2.5 D-Node Retargeting

D-node retargeting (DNR) is a front-to-front heuristic search method proposed by Politowski and Pohl (1984). DNR alternately performs forward and backward search using $Open_f$ and $Open_b$. After expanding n states in one direction, DNR reverses direction. In DNR, the h-value of a state s is the estimated distance between s and a *d-node* d, a state in $Open_b$. In forward search, d is a state s with the highest $g_b(s)$ in $Open_b$, where $g_b(s)$ is the cost of the found path from s to the goal state, and in backward search, d is a state s with the highest $g_f(s)$ in $Open_f$, where $g_f(s)$ is the cost of the found path from the initial state to s. DNR reevaluates all states in $Open_f$ and $Open_b$ whenever the d-node changes, which can incur a major overhead.

3 Top-to-Top Bidirectional Search

We propose Top-to-Top Bidirectional Search (TTBS), a simple front-to-front bidirectional search strategy. TTBS can be considered as a variant of DNR but in order to avoid the large overhead of evaluating states multiple times which SFBS and DNR incurs (see above), TTBS uses the following scheme to approximate this behavior.

Intuitively, in front-to-front heuristic search, if the heuristic function successfully guides the search toward the opposite frontier, the top of $Open_f$ should be close to the top of $Open_b$. When evaluating a state s, TTBS computes heuristic estimate of the distance between s and d, where d is the top (lowest h-value) state in the opposite *Open*, at the time when s is being evaluated. In other words, TTBS uses the top of opposite *Open* at that time as a d-node for s.

While DNR reevaluates all states in *Open* when the d-node is changed, TTBS adopts a more moderate reevaluation mechanism. At the start of a state expansion phase, TTBS first removes s, the highest priority state from the open list. Let $D(s)$ be the state which was the d-node when s was evaluated. TTBS checks if the current d-node d is "similar" to $D(s)$. If d is similar to $D(s)$, TTBS expands s. Otherwise, TTBS reevaluates s against d (i.e., $h(s, d)$), inserts s to *Open*, and repeats the process. In our current implementation, d is defined as similar to $D(s)$ if d is $D(s)$ or a successor of $D(s)$. To try to prevent unnecessarily frequent reevaluations, TTBS prioritizes the reevaluated state over the other states with the same h-value, i.e., the TTBS *Open* is prioritized first according to h, then ties broken according to whether it was reevaluated (higher priority) or not (lower priority), and then further ties are broken according to standard tie-breaking strategies (e.g., FIFO/LIFO). Thus, if the h-value of the top priority state s remains the lowest in *Open* after the reevaluation, s is immediately expanded. During the reevaluation process, TTBS does not need to reevaluate the same state against d more than once since d is fixed. However, in the worst case, TTBS reevaluates all the states in the open list.

In summary, TTBS differs from DNR in that (1) TTBS uses the top of the opposite *Open* as the d-node while DNR uses the state with highest g-value, and (2) while DNR reevaluates all states in *Open* whenever the d-node is changed, TTBS uses a more limited reevaluation strategy.

The procedure TTBS in Algorithm 1 shows the pseudocode of TTBS for classical planning. TTBS alternates between forward and backward search. For backward search, TTBS uses regression as biFD does in [Alcázar *et al.*, 2014]. s_0 is the initial state and S_g is the set of goal states. While s_0, s, and s' are states represented as value assignments to state variables, S_g, S, and S' are sets of states where some variables are possibly assigned 'undefined' values. $Open_f$ and $Open_b$ are priority queues for the forward and backward search, respectively, where $push(Open, s, h)$ inserts a state s to *Open* with the priority value h, $top(Open)$ returns a state in *Open* with the lowest priority value according to the tie-breaking strategy, and $pop(Open)$ returns $top(Open)$ and removes it from *Open*. gen_f and gen_b are hash sets containing states and sets of states represented as value assignments to state variables. The forward search steps correspond to standard best-first forward search, except for the use of the top-to-top heuristic $h(s, d)$ and the reevaluation mechanism.

To detect forward/backward frontier intersection, TTBS checks if a generated state was already generated in the opposite direction. In addition, TTBS compares an expanded state s with the opposite top state in line 16 and line 30. The intersection detections in line 18 and line 32 are implemented as looking up of hash sets. Thus, even if an entry S in gen_b subsumes a forward generated state s, it is possible that $s \notin gen_b$ when S has undefined values, i.e., TTBS may overlook intersections. Previous approaches which used BGGs for forward/backward frontier intersection and incurred significant overheads for intersection/subsumption checking [Alcázar *et al.*, 2014], so TTBS trades off the possibility of missed intersections for fast intersection checks.

In this paper, we assume that t is unreachable from s if $h(s, t) = \infty$. Even when the front-to-front h-value of s is infinity, TTBS inserts s into *Open* in line 3 if the front-to-end h-value is not infinity. TTBS always checks if s is a goal state when expanding s in line 13 and 16, and if S includes the initial state when expanding S in line 27 and 30. Therefore, a path from the initial state s_0 to a goal will eventually be found, if one exists, and TTBS is complete. On the other hand, if $Open_f$ or $Open_b$ becomes empty, there is no possible path from the initial state to a goal. In such a case, TTBS concludes that there is no solution (line 22 and 36.

Algorithm 1 Top-to-Top Bidirectional Search (TTBS)

```
1: procedure PUSH($Open$, $s$, $d$, $h$, $h_e$)
2:     if $h = \infty$ then $h \leftarrow h_e$
3:     if $h \neq \infty$ then $D(s) \leftarrow d$; $push(Open, s, h)$
4: procedure TTBS
5:     $D(s_0) \leftarrow S_g$; $D(S_g) \leftarrow s_0$
6:     $push(Open_f, s_0, h(s_0, S_g))$; $gen_f \leftarrow \{s_0\}$
7:     $push(Open_b, S_g, h(s_0, S_g))$; $gen_b \leftarrow \{S_g\}$
8:     loop
9:         $d \leftarrow top(Open_b)$
10:        for $i = 1, ..., n$ do
11:            $s \leftarrow pop(Open_f)$
12:            while $D(s)$ is not similar to $d$ do
13:                if $S_g \subseteq s \lor d \subseteq s$ then return a solution
14:                PUSH($Open_f$, $s$, $d$, $h(s, d)$, $h(s, S_g)$)
15:                $s \leftarrow pop(Open_f)$
16:            if $S_g \subseteq s \lor d \subseteq s$ then return a solution
17:            for $s' \in$ successors of $s$ do
18:                if $s' \in gen_b$ then return a solution
19:                if $s' \in gen_f$ then continue
20:                $gen_f \leftarrow gen_f \cup \{s'\}$
21:                PUSH($Open_f$, $s'$, $d$, $h(s', d)$, $h(s', S_g)$)
22:            if $Open_f = \emptyset$ then return no solution
23:        $d \leftarrow top(Open_f)$
24:        for $i = 1, ..., n$ do
25:            $S \leftarrow pop(Open_b)$;
26:            while $D(S)$ is not similar to $d$ do
27:                if $S \subseteq s_0 \lor S \subseteq d$ then return a solution
28:                PUSH($Open_b$, $S$, $d$, $h(d, S)$, $h(s_0, S)$)
29:                $S \leftarrow pop(Open_b)$
30:            if $S \subseteq s_0 \lor S \subseteq d$ then return a solution
31:            for $S' \in$ predecessors of $S$ do
32:                if $S' \in gen_f$ then return a solution
33:                if $S' \in gen_b$ then contine
34:                $gen_b \leftarrow gen_b \cup \{S'\}$
35:                PUSH($Open_b$, $S'$, $d$, $h(d, S')$, $h(s_0, S')$)
36:            if $Open_b = \emptyset$ then return no solution
```

4 Experimental Evaluation

We experimentally evaluated the following search algorithms, all of which were implemented by modifying the Fast Downward planning system. (1) **FD**: forward GBFS, (2) **FDr**: backward GBFS using regression, (3) **biFD**: interleaves FD and FDr, (4) **BDD**: biFD + BDD. BDDs are used for duplicate detection of backward search and intersection detection, (5) **BGG**: biFD + BGGs. We used all states generated by backward search as BGGs. States generated by forward search are evaluated against the state s with the lowest $h_{\max}(s_0, s)$. If multiple states have the lowest $h_{\max}$ value, we use the state that was generated first. (6) **DNR(PP84)**: the original version of D-node retargeting [Politowski and Pohl, 1984] ($n = 75$), (7) **SFBS**: SFBS using front-to-front h-values as priority values where the jump policy is direction alternation: choose the opposite direction to the direction chosen at the parent node, (8) **TTBS** : TTBS with $n = 1$ where d is assumed to be similar to $D(s)$ if d is $D(s)$ or $D(s)$ is the parent of d in the reevaluation mechanism. Differently from Alcázar *et al.* (2014), we used eager GBFS instead of lazy GBFS and did not use preferred operators.

All methods used the unit-cost version of h_{ff} [Hoffmann and Nebel, 2001], which is straightforwardly applicable in regression and bidirectional search [Alcázar *et al.*, 2013; Alcázar *et al.*, 2014], using eager evaluation, and the FIFO tie-breaking strategy. All code will be made available on a public repository.

We used 1229 solvable instances in 47 planning domains from classical satisficing tracks in International Planning Competition (IPC) 98-2018, whose SAS+ representations do not contain axioms and conditional effects. In the case of domains with overlaps in multiple years, we used the newest set. All runs were given a 5 min. time limit and a 4GB memory limit on Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz processors.

Table 1 shows the results per domain. In addition to coverage (cov, the number of problems solved), for the bidirectional searches, we also show the number of instances solved by the bidirectional algorithm on which both FD and FDr failed (i.e., cases where bidirectional search succeeded but both forward and backward unidirectional searches failed). Furthermore, for the bidirectional searches, we show the $meet$ metric, which indicates to which extent the search frontiers are meeting in the middle, and is defined as the average (over instances in the domain) of min(#forward/#plan, 1 - #forward/#plan) where #plan is the length of the resulting plan and #forward is the number of states generated by forward search in the plan.

Table 2 summarizes the number of domains where one method solved more instances than another.

Our results of FDr and biFD seem different from that in Alcázar *et al.* (2014). This is possibly because we used newer versions in some domains and eager GBFS without preferred operators instead of lazy GBFS with preferred operators.

While biFD performed comparably to forward search (FD), the $meet$ metric shows that biFD almost never "meets in the middle" (almost all domains have a $meet$ score close to 0, and no domains with $meet \geq 0.2$), and behaves essentially like a portfolio consisting of independently executing forward (FD) and backward (FDr) search components. Furthermore, the variants of biFD using BGGs and BDD also exhibit weak "meet in the middle" behavior. This confirms the previous observations in that these algorithms "commit strongly to subtrees in the search state and often do not collide with the opposite frontier until they are close to the goal" (Alcázar *et al.* 2014, p.353, col.1, par.2). biFD did not solve any instances which were not solved by either FD or FDr. BDD solved 1 such instance, and BGG solved 3 such instances. Thus, none of these biFD variants exhibit performance complementary to both FD and FDr.

TTBS exhibited meet-in-the-middle behavior ($meet > 0.2$) on a majority (26/47) of the tested IPC domains and strong meet-in-the-middle behavior ($meet > 0.4$) on 18 domains. While TTBS was not competitive with FD with re-

	FD	FDr	biFD			BDD			BGG			SFBS			DNR(PP84)			TTBS		
	cov	cov	cov	+	meet	cov	+	meet	cov	+	meet	cov	+	meet	cov	+	meet	cov	+	meet
agricola18 (20)	**9**	0	0	0	-	0	0	-	0	0	-	0	0	-	0	0	-	0	0	-
airport (50)	33	29	**37**	0	0.00	26	0	0.22	27	0	0.02	17	0	**0.49**	33	**2**	0.03	30	1	0.21
blocks (35)	35	35	35	0	0.19	35	0	0.19	35	0	0.28	35	0	**0.49**	35	0	0.33	35	0	0.45
data18 (20)	**4**	0	**4**	0	0.00	3	0	0.05	1	0	0.00	0	0	-	1	0	**0.24**	0	0	-
depot (22)	**15**	7	13	0	0.03	12	0	0.05	9	0	0.05	9	0	0.41	11	0	0.09	9	0	**0.43**
driverlog (20)	17	18	19	0	0.00	18	0	0.03	17	0	0.00	17	0	0.46	17	0	0.08	**20**	**1**	**0.47**
elevators11 (20)	17	**18**	17	0	0.00	15	0	0.00	12	0	0.03	9	0	0.45	1	0	0.40	14	**1**	**0.49**
floortile14 (20)	2	**20**	**20**	0	0.02	**20**	0	0.02	19	0	0.03	1	0	**0.40**	10	0	0.30	**20**	0	0.02
freecell (80)	**78**	6	73	0	0.00	65	0	0.00	15	0	0.01	1	0	**0.36**	63	0	0.00	41	0	0.00
ged14 (20)	**20**	0	16	0	0.00	15	0	0.00	12	0	0.00	0	0	-	2	0	**0.15**	0	0	-
grid (5)	4	4	4	0	0.00	4	0	0.17	4	0	0.00	**5**	**1**	0.39	4	0	0.09	**5**	**1**	**0.41**
gripper (20)	20	20	20	0	0.03	20	0	0.03	20	0	0.03	20	0	0.48	20	0	0.13	20	0	**0.49**
hiking14 (20)	**20**	6	17	0	0.02	16	0	0.04	13	0	0.03	7	0	0.29	18	0	0.10	19	0	**0.40**
logistics00 (28)	28	28	28	0	0.04	28	0	0.05	28	0	0.11	28	0	0.48	28	0	0.07	28	0	**0.49**
mainte14 (20)	**11**	7	**11**	0	0.00	9	0	0.00	5	0	0.00	2	0	**0.50**	6	0	0.01	2	0	0.49
movie (30)	30	30	30	0	0.00	30	0	0.00	30	0	0.00	30	0	**0.43**	30	0	0.00	30	0	**0.43**
mprime (35)	**30**	11	29	0	0.00	29	0	0.12	21	**1**	0.00	17	0	**0.45**	**30**	**1**	0.02	24	0	0.30
mystery (19)	17	11	17	0	0.00	17	0	0.11	13	0	0.00	12	0	**0.43**	17	0	0.00	**18**	**1**	0.28
nomystery11 (20)	**9**	4	**9**	0	0.02	**9**	0	0.02	4	0	0.01	2	0	**0.47**	6	0	0.07	6	0	0.11
org-split18 (20)	**10**	0	5	0	0.00	4	0	0.00	1	0	0.00	0	0	-	5	0	0.00	2	0	0.00
org18 (20)	**2**	**2**	**2**	0	0.00	**2**	0	0.00	**2**	0	0.00	1	0	**0.50**	**2**	0	0.00	1	0	**0.50**
parcprinter11 (20)	**20**	0	**20**	0	0.00	**20**	0	0.02	9	0	0.00	0	0	-	**20**	0	**0.06**	11	0	0.00
parking14 (20)	**2**	0	**2**	0	0.00	**2**	0	0.00	0	0	-	0	0	-	0	0	-	0	0	-
pathways (30)	9	5	9	0	0.00	9	0	0.13	5	0	0.08	6	1	**0.50**	9	0	0.01	**12**	**3**	0.39
pegsol11 (20)	**20**	16	**20**	0	0.08	**20**	0	0.08	15	0	0.12	9	0	**0.33**	18	0	0.11	19	0	0.16
pipes-notank (50)	**27**	1	22	0	0.00	14	0	0.01	7	0	0.00	3	0	**0.13**	15	**1**	0.01	10	0	0.08
pipes-tank (50)	**19**	8	16	0	0.00	13	0	0.06	10	**1**	0.00	8	0	**0.41**	12	0	0.05	10	0	0.31
psr-small (50)	**50**	49	**50**	0	0.00	**50**	0	0.32	**50**	0	0.04	47	0	**0.35**	49	0	0.08	48	0	0.20
rovers (40)	23	33	31	0	0.00	32	1	0.03	23	0	0.01	31	1	0.48	19	0	0.18	**38**	**5**	**0.48**
satellite (36)	**26**	12	**26**	0	0.00	**26**	0	0.02	8	0	0.04	24	0	0.41	21	0	0.16	25	0	**0.49**
scanalyzer11 (20)	15	**20**	**20**	0	0.02	**20**	0	0.16	19	0	0.08	18	0	**0.47**	17	0	0.24	**20**	0	0.47
sokoban11 (20)	**19**	2	17	0	0.00	4	0	0.08	3	0	0.00	2	0	**0.25**	12	0	0.00	14	0	0.00
spider18 (20)	**7**	0	1	0	0.00	0	0	-	0	0	-	0	0	-	0	0	-	0	0	-
storage (30)	18	16	**19**	0	0.00	**19**	0	0.13	17	0	0.00	16	0	0.18	17	0	0.08	16	0	**0.19**
termes18 (20)	11	15	13	0	0.12	12	0	0.13	8	0	0.14	11	0	0.10	9	0	0.18	**16**	**2**	**0.41**
tetris14 (20)	**2**	0	0	0	-	0	0	-	0	0	-	0	0	-	0	0	-	0	0	-
thoughtful14 (20)	**8**	0	**8**	0	0.00	5	0	0.00	1	0	0.00	0	0	-	**8**	**1**	**0.00**	5	0	0.00
tidybot11 (20)	**16**	0	10	0	0.00	4	0	0.00	0	0	-	0	0	-	3	0	**0.11**	0	0	-
tpp (30)	**16**	8	15	0	0.03	15	0	0.13	8	0	0.00	13	0	**0.42**	14	0	0.11	15	0	0.37
transport14 (20)	0	**3**	2	0	0.00	2	0	0.00	0	0	-	1	1	0.18	0	0	-	2	**2**	**0.42**
trucks (20)	14	6	14	0	0.00	13	0	0.00	5	0	0.17	2	0	**0.44**	**15**	**1**	0.00	**15**	**1**	0.00
visitall14 (20)	0	0	0	0	-	0	0	-	0	0	-	**9**	**9**	0.13	0	0	-	**9**	**9**	**0.37**
woodwork11 (20)	2	1	2	0	0.00	2	0	0.00	3	1	0.00	10	9	**0.49**	10	8	0.12	**12**	**11**	0.48
zenotravel (20)	20	20	20	0	0.05	20	0	0.08	20	0	0.09	20	0	**0.45**	20	0	0.14	20	0	0.44
total (1229)	**755**	471	743	0		679	1		499	3		443	22		627	14		641	**38**	

Table 1: Results per domain (5 min, 4GB limit per instance). 'cov' is coverage (the number of solved instances), and '+' is the number of solved instances unsolved by FD or FDr. '*meet*' is the average of min(#forward/#plan, 1 - #forward/#plan) where #plan is the length of the resulting plan and #forward is the number of states generated by forward search in the plan, and indicates meet-in-the-middle behavior. Domains where no instance is solved are omitted.

spect to total coverage across all domains, TTBS solved 38 instances which were solved by neither FD nor FDr (Table 1) and solved more instances than FD in 12 domains and FDr in 28 domains (Table 2). For a more detailed view, Figure 1 shows the node expansions and search time per instance for TTBS vs. FD and FDr. Thus, TTBS, unlike the previous biFD-based bidirectional searches, yields performance which is complementary to both FD and FDr.

Comparing TTBS and SFBS, we observe that although SFBS displayed meet-in-the-middle behavior ($meet > 0.4$) on 21 domains, TTBS significantly outperformed SFBS. As shown in Figure 2, SFBS tends to perform many more heuristic evaluations expand more search nodes, resulting in longer search times than TTBS. This is because in SFBS, each state is involved in multiple evaluations and expansions, as explained in Section 2.1.

4.1 Comparison of TTBS vs. DNR

We evaluate how the differences between TTBS and DNR affect their search behaviors. We compared TTBS with the following DNR variatns. (1) DNR(PP84), (2) DNR(OT): DNR uses the top of the opposite *Open* as the d-node, (3) DNR(re): DNR uses the same reevaluation mechanism as TTBS. Since $n = 75$ was used in the original version of DNR [Politowski and Pohl, 1984], we evaluated each variant with $n = 1$ and $n = 75$.

In Table 3, comparing DNR(re) vs. DNR(PP84) and TTBS vs. DNR(OT) shows that the TTBS reevaluation mechanism significantly increases the number of '+' instances (solved by bidirectional search but unsolved by FD and FDr) and the number of domains containing '+' instances, as well as the total coverage. Comparing DNR(OT) vs. DNR(PP84) shows using the top of the opposite *Open* as the d-node by itself

	FD	FDr	biFD	BDD	BGG	SFBS	DNR(PP84)	TTBS
FD	-	29	14	20	29	31	24	27
FDr	7	-	4	5	12	15	8	5
biFD	8	28	-	18	32	33	25	23
BDD	7	25	1	-	30	33	21	19
BGG	3	16	1	2	-	23	6	6
SFBS	6	11	3	3	9	-	8	0
DNR(PP84)	4	23	4	9	22	25	-	16
TTBS	12	24	10	12	26	26	17	-

Table 2: Summary: the number of domains where a method in a row solved more instances than a method in a column.

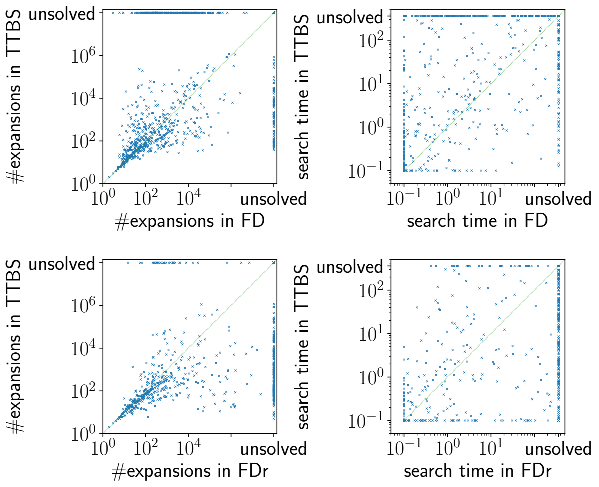

Figure 1: The number of expansions and search time, TTBS vs. FD and FDr. In search time plots, instances solved within 0.1 seconds are shown at 10^{-1}.

	$n = 1$			$n = 75$		
	cov	+	dom	cov	+	dom
DNR(PP84)	538	12	6	627	14	6
DNR(OT)	516	12	6	581	18	6
DNR(re)	630	27	**12**	**673**	28	11
TTBS	641	**38**	**12**	655	34	11

Table 3: Comparison of TTBS and DNR variants. ‘cov’ is the number of solved instances, ‘+’ is the number of solved instances unsolved by FD and FDr, and ‘dom’ is the number of domains where ‘+’ is more than 0.

does not appear to be beneficial to performance. However, comparing TTBS vs. the DNR variants shows that combined with the reevaluation mechanism, using the top of the opposite *Open* as the d-node increases ‘+’ instances.

We also compare the number of expansions, evaluations, and the *meet* metric for TTBS and the DNR variants with $n = 1$ in Figure 3. The TTBS reevaluation mechanism reduces the number of evaluations. In addition, both the use of the top of the opposite *Open* as the d-node and the TTBS reevaluation mechanism promote meet-in-the-middle behavior in many instances.

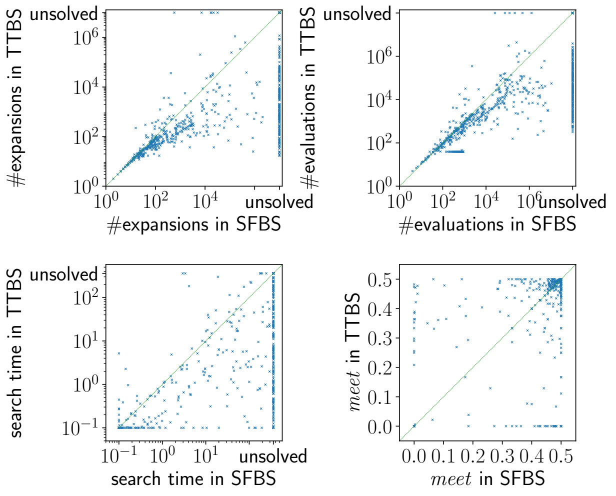

Figure 2: TTBS vs. SFBS: number of expansions, search time, and the *meet* metric. In the search time plot, instances solved within 0.1 seconds are shown at 10^{-1}.

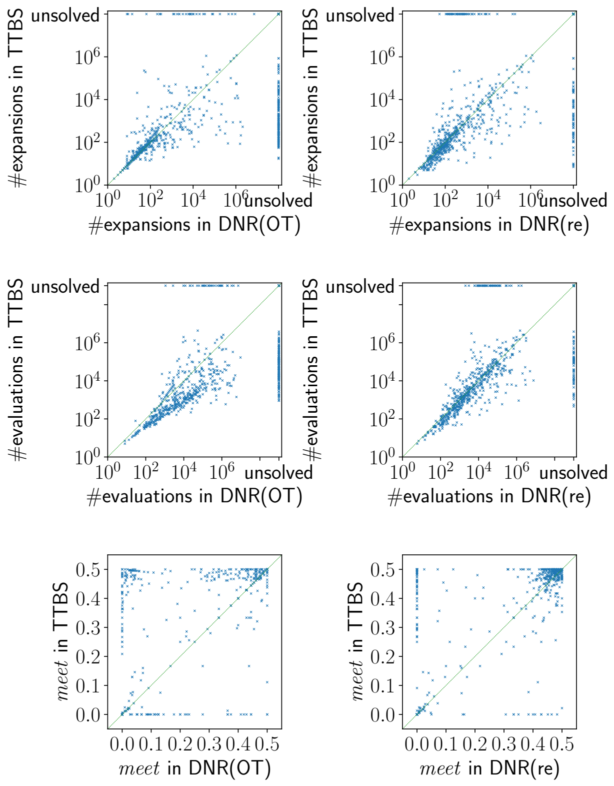

Figure 3: The number of expansions, evaluations and the *meet* metric in TTBS and DNR variants. We use $n = 1$ here.

meet	$= 0.0$	$(0.0, 0.1]$	$(0.1, 0.2]$	$(0.2, 0.3]$	$(0.3, 0.4]$	$(0.4, 0.5]$
+	2	0	0	1	9	26

Table 4: The number of '+' instances, which is solved by TTBS but unsolved by FD and FDr along with ranges of the *meet* metric.

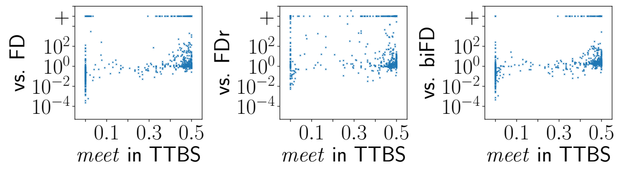

Figure 4: Relation of the *meet* metric and the number of expansions. Each plot compares TTBS and another method A, and x-axis is the *meet* metric and y-axis is expansions(A) / expansions(TTBS) where expansions(A) is the number of expansions by A. Instances solved by TTBS but not solved by A are shown at '+'.

	expansions			
	+	fewer	same	more
meet > 0.4 vs. FD	45	222	39	56
meet ≤ 0.4 vs. FD	31	42	11	195
meet > 0.4 vs. FDr	69	193	52	48
meet ≤ 0.4 vs. FDr	132	84	10	53
meet > 0.4 vs. biFD	33	308	1	20
meet ≤ 0.4 vs. biFD	13	83	12	171

Table 5: Comparison of TTBS vs. other methods, for *meet* > 0.4 and *meet* ≤ 0.4. '+' is the number of instances which is solved by TTBS but unsolved by the other method. 'fewer', 'same', and 'more' is the number of instances where TTBS expands fewer, the same number of, and more states than the other method.

4.2 Meet-in-the-Middle Behavior

Next, we consider the relationship between the degree of "meet-in-the-middle" behavior (as measured by the *meet* metric) and search performance. Of the 38 instances solved by TTBS but unsolved by both FD and FDr, TTBS has *meet* > 0.4 in 26 instances and $0.3 <$ *meet* ≤ 0.4 in 9 instances, as shown in Table 4. We also compared the number of expansions by TTBS vs. FD, FDr, and biFD, (which does not show meet-in-the-middle behavior) in Figure 4 and Table 5. When *meet* > 0.4, TTBS tends to expand fewer states than other methods, while TTBS tends to expand more states than FD and FDr when *meet* ≤ 0.4. Thus, the *meet* metric is highly correlated to the performance of TTBS.

4.3 Portfolios That Include Bidirectional Search: Exploiting the Complementary Behavior of Bidirectional Search

The preceding experiment showed that unlike previous bidirectional approaches, TTBS exhibited much stronger complementary search behavior relative to both forward (FD) and backward (FDr) search. To show how this complementary behavior can be exploited, we evaluate portfolios composed of unidirectional and bidirectional searches.

FD+FDr	+biFD	+BDD	+BGG	+SFBS	+DNR(PP84)	+TTBS
786	771	771	773	788	781	**801**

Table 6: Coverage of portfolios.

We evaluated the following portfolios. (1) **FD+FDr**, (2) **+biFD**, (3) **+BGG**, (4) **+BGG**, (5) **+SFBS**, (6) **+DNR(PP84)**, (7) **+TTBS**. All portfolios use FD and FDr as components, and the $+X$ indicates the third planner, e.g., "+TTBS" = FD+FDr+TTBS. A total 5 min. time limit per portfolio is distributed evenly among the components. Coverage results are shown in Table 6. Since BGG and BDD exhibited weak complementarity relative to FD and FDr, portfolios that include BGG and BDD do not improve coverage vs. FD+FDr. In contrast, the portfolio combining forward, backward, and front-to-front bidirectional search (FD+FDr+TTBS) outperforms all other configurations, including a portfolio consisting of forward and backward search, as well as portfolios using previous bidirectional searches.

5 Conclusions

We proposed and evaluated TTBS, a new front-to-front bidirectional search strategy for satisficing heuristic search where states are prioritized according to their distance to the top state in the opposite frontier at the time of their evaluation. This new prioritization scheme can be considered as a variant of D-node retargeting [Politowski and Pohl, 1984]. TTBS assumes that if the bidirectional search is successfully moving the frontiers toward each other, the current top of the opposite *Open* is a good approximation for the opposite frontier state closest to s.

Our experimental evaluation showed that unlike biFD, a front-to-end approach which does not exhibit meet-in-the-middle behavior, TTBS exhibits meet-in-the-middle behavior in a majority of the tested IPC domains, and strong meet-in-the-middle behavior in some domains.

While TTBS *by itself* is not competitive with forward search, or a portfolio consisting of forward and backward search, we showed that unlike previous bidirectional searches, TTBS can solve many instances which neither forward nor backward search could solve, and a portfolio combining forward search, backward search, and TTBS outperformed all other configurations. Thus, we have shown that front-to-front bidirectional search can be successfully used as a complementary approach to forward search, backward search, and forward/backward portfolios.

Although TTBS performs better overall than SFBS because SFBS incurs very large overhead due to the need to evaluate heuristics for pairs of states, SFBS demonstrates stronger meet-in-the-middle behavior than TTBS on many domains. This is to be expected, as TTBS in some sense tries to approximate the stricter front-to-front behavior of SFBS without incurring as much overhead. Improved reevaluation strategies to more closely approximate the front-to-front behavior of SFBS is a direction for future work.

References

[Alcázar *et al.*, 2013] Vidal Alcázar, Daniel Borrajo, Susana Fernández, and Raquel Fuentetaja. Revisiting regression in planning. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence IJCAI, Beijing, China, August 3-9, 2013*, pages 2254–2260, 2013.

[Alcázar *et al.*, 2014] Vidal Alcázar, Susana Fernández, and Daniel Borrajo. Analyzing the impact of partial states on duplicate detection and collision of frontiers. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*, 2014.

[Bäckström and Nebel, 1995] Christer Bäckström and Bernhard Nebel. Complexity results for SAS+ planning. *Comput. Intell.*, 11:625–656, 1995.

[Barker and Korf, 2015] Joseph Kelly Barker and Richard E. Korf. Limitations of front-to-end bidirectional heuristic search. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 1086–1092, 2015.

[Bonet and Geffner, 2001] Blai Bonet and Hector Geffner. Planning as heuristic search. *Artif. Intell.*, 129(1-2):5–33, 2001.

[Bryant, 1986] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

[Doran and Michie, 1966] James Doran and D Michie. Experiments with the graph traverser program. In *Proceedings of The Royal Society A: Mathematical, Physical and Engineering Sciences*, volume 294, pages 235–259, 09 1966.

[Felner *et al.*, 2010] Ariel Felner, Carsten Moldenhauer, Nathan R. Sturtevant, and Jonathan Schaeffer. Single-frontier bidirectional search. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, 2010.

[Green, 1969] C. Cordell Green. Application of theorem proving to problem solving. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence, Washington, DC, USA, May 7-9, 1969*, pages 219–240, 1969.

[Hart *et al.*, 1968] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[Helmert, 2006] Malte Helmert. The fast downward planning system. *J. Artif. Intell. Res.*, 26:191–246, 2006.

[Hoffmann and Nebel, 2001] Jörg Hoffmann and Bernhard Nebel. The FF Planning System: Fast Plan Generation through Heuristic Search. *J. Artif. Intell. Res.*, 14:253–302, 2001.

[Holte *et al.*, 2017] Robert C. Holte, Ariel Felner, Guni Sharon, Nathan R. Sturtevant, and Jingwei Chen. MM: A bidirectional search algorithm that is guaranteed to meet in the middle. *Artif. Intell.*, 252:232–266, 2017.

[Politowski and Pohl, 1984] George Politowski and Ira Pohl. D-node retargeting in bidirectional heuristic search. In Ronald J. Brachman, editor, *Proceedings of the National Conference on Artificial Intelligence. Austin, TX, USA, August 6-10, 1984*, pages 274–277. AAAI Press, 1984.

[Sturtevant and Felner, 2018] Nathan R. Sturtevant and Ariel Felner. A brief history and recent achievements in bidirectional search. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 8000–8007, 2018.

[Torralba *et al.*, 2016] Álvaro Torralba, Carlos Linares López, and Daniel Borrajo. Abstraction heuristics for symbolic bidirectional search. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 3272–3278, 2016.