Proceedings of the Twelfth International
Symposium on Combinatorial Search (SoCS 2019)

Enriching Non-Parametric Bidirectional Search Algorithms — Extended Abstract*

Shahaf S. Shperberg
CS Department
Ben-Gurion University
Be'er-Sheva, Israel
shperbsh@post.bgu.ac.il

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

Nathan R. Sturtevant
CS Department
University of Alberta
Canada
sturtevant@cs.du.edu

Solomon E. Shimony
Avi Hayoun
CS Department
Ben-Gurion University
Be'er-Sheva, Israel
shimony@cs.bgu.ac.il

Abstract

NBS is a non-parametric bidirectional search algorithm, proved to expand at most twice the number of node expansions required to verify the optimality of a solution. We introduce DVCBS, a new algorithm based on a dynamic vertex cover that aims to further reduce the number of expansions. Unlike NBS, DVCBS does not have any worst-case bound guarantees, but in practice it outperforms NBS in verifying the optimality of solutions.

1 Bidirectional Search using Dynamic VC

We introduce a new family of algorithms called Dynamic Vertex Cover Bidirectional Search (DVCBS). Like NBS (Chen et al. 2017) DVCBS is based on maintaining a global lower-bound (LB) among all pairs in the open list and terminates when the current solution cost is greater or equal to this LB. However, DVCBS differs conceptually from NBS. While NBS always expands both nodes of a chosen must-expand pair (MEP), DVCBS works by maintaining a *dynamic* version of the must-expand graph (G_{MX}) called DG_{MX} and greedily expanding a minimal vertex cover (MVC) of the DG_{MX} at each step.

DG_{MX} is defined as follows. Its structure resembles G_{MX}, with two main differences: (1) The full G_{MX} is not available during the search. Instead, DG_{MX} contains only nodes in the forward frontier (generated, not expanded) for constructing left vertices, and only nodes from the backward frontier for constructing right vertices. (2) The value of C^* is not known during the search, thus edges of DG_{MX} are defined on pairs $\langle u, v\rangle$ such that $lb(u, v) < LB$. Since $LB \leq C^*$, all such pairs are in fact MEPs of G_{MX}.

Note that DG_{MX} shares all the interesting properties of the full G_{MX}. Thus, vertices with the same g-value can be merged to form a weighted vertex (cluster). More importantly, CalculateMVC() can be directly applied to DG_{MX} in time linear in the number of its clusters. This is done in all low-level variants of DVCBSdiscussed below.

1.1 Choosing Nodes for Expansion

There are many possible policies for choosing nodes for expansion which are based on DG_{MX} and on its MVC. Every node expansion deletes vertices and may add new vertices to DG_{MX}, invalidating the most recently computed MVC. However, computing the MVC every time DG_{MX} changes incurs extra overhead (albeit linear in the number of clusters in DG_{MX}). Thus, an efficient expansion policy should balance between expanding many nodes and maintaining the most up-to-date DG_{MX} and MVC. We experimented with multiple expansion policy variants, and found that an efficient balance between these two extremes is to expand a single cluster (containing all nodes with the same g_F- or g_B-value) before re-computing the DG_{MX}. This results in a manageable amount of MVC computations, while working on reasonably up-to-date information. Furthermore, since all vertices in a cluster have the same g-value, LB may increase only after expanding an entire cluster but never before. We only report experimental results for this variant.

DVCBS contains several other decision points. First, there can be several possible MVCs for a given DG_{MX}. Additionally, as mentioned above, one cluster from MVC should be chosen and expanded. Finally, the way we order nodes within the cluster for expansion may affect the number of expansions before reaching a solution when $LB = C^*$. We have experimented with many possible decision choices but report the results in Section 2 using the best variant as follows. Select the cluster with the smallest number of nodes among the clusters with minimal g_F- and g_B-values, among all MVCs. Tie breaking for a specific node expansion within a cluster orders nodes according to their order of discovery.

Pseudo code of the low level of DVCBS appears in Algorithm 1. The life cycle of DVCBS includes the following steps: (1) initialize DG_{MX}, (2) CalculateMVC(), (3) choose the cluster of nodes to expand from the MVC, and (4) update DG_{MX}. Steps 2-4 are repeated until either an optimal solution is found or no possible solution exists. To execute efficiently, DVCBS uses data structures denoted as $Cwaiting_D$ and $Cready_D$, which are similar to the $waiting_D$ and $ready_D$ queues of NBS, modified to use clusters.

The most important property of NBS is the $2\times$ bound guarantee. While DVCBS outperforms NBS on average (see experiments below), DVCBS is not bounded in its worst case.

2 Experimental Evaluation

We ran experiments on four domains: **(1)** 50 **14-Pancake Puzzle** instances with the GAP-n heuristics (Helmert 2010)

*Originally published at AAAI-2019

Algorithm 1: DVCBS Expand a Level

```
while true do
    while min f in Cwaiting_D < LB do
        Move best cluster from Cwaiting_D to Cready_D
    if Cready_D ∪ Cwaiting_D empty then
        Terminate search - no solution was found
    DG_MX ← BuildDGMX(Cready_D)
    if DG_MX is not empty then
        MVC ← findMVC(DG_MX)
        Choose and Expand a cluster from MVC of DG_MX.
    else
        if Cwaiting_D.f ≤ LB then
            Move best cluster from Cwaiting_D to Cready_D
        else
            return true
```

(for $n = 0 \dots 2$), where the n smallest pancakes are left out of the heuristic computation. **(2)** The standard 100 instances of the **15 Puzzle** problem (Korf 1985) using the Manhattan Distance heuristic. **(3) Grid**-based pathfinding: 3150 instances on 156 maps from Dragon Age Origins (DAO) (Sturtevant 2012); (4) 50 instances of the 12-disk **4-peg Towers of Hanoi** (TOH4) problem with (10+2), and (6+6) additive PDBs (Felner, Korf, and Hanan 2004).

Our results, reported in Table 1, confirm that NBS and DVCBS tends to outperform and are more robust than A*İn some cases, e.g., the 15 puzzle, A* failed to solve all instances because memory was exhausted.

Since NBS has a 2x bound guarantee, any other algorithm will expand no fewer than half the nodes of NBS, leaving little leeway. Nevertheless, our new algorithm managed to improve upon NBS DVCBS expands fewer nodes than NBSbefore finding a VC of G_{MX_ϵ}, and before finding a solution.

The node expansion rates of all algorithms were similar, with very low variance. Therefore, the number of node expansions reported in 1 reflect the run-time accurately.

We have also compared DVCBS to A* as well as to MMϵ (Holte et al. 2017) and BS* (Kwa 1989) which are benchmark Bi-HS algorithms. Table 2 presents the average number of node expansions for finding a first solution. As can be seen, DVCBS tends to outperform all others, and is certainly the most robust to weaker heuristics.

Acknowledgements

This work was supported by Israel Science Foundation (ISF) grant #844/17 to Ariel Felner and Eyal Shimony, by BSF grant #2017692, by NSF grant #1815660 and by the Frankel center for CS at BGU.

Domain	Heuristic	Algorithm	VC: G_{MX}	Total
14 Pancake	GAP	A*	32	**57**
		NBS	47	147
		DVCBS	**30**	121
	GAP-1	A*	6,404	6,416
		NBS	5,870	5,915
		DVCBS	**4,321**	**4,344**
	GAP-2	A*	322,099	322,938
		NBS	137,295	137,719
		DVCBS	**86,292**	**87,012**
15 Puzzle	MD	NBS	12,709,517	12,748,107
		DVCBS	**11,589,837**	**11,669,720**
Grids DAO	Octile	A*	5,322	**5,406**
		NBS	6,561	6,677
		DVCBS	**5,158**	5,545
TOH4	10+2	A*	276,081	276,089
		NBS	232,509	232,509
		DVCBS	**224,233**	**224,249**
	6+6	A*	3,239,287	3,268,093
		NBS	663,136	681,995
		DVCBS	**636,375**	**664,469**

Table 1: Experimental results of average node expansions across domains (using $\epsilon = 1$)

Domain	BS*	MMϵ	DVCBS	A*
GAP-0	183	149	121	**57**
GAP-1	5,262	5,048	**4,344**	6,416
GAP-2	266,442	119,310	**87,012**	322,938
10+2	**174,936**	303,189	224,249	276,089
6+6	1,599,018	1,120,392	**664,469**	3,268,093
MD	12,001,024	13,162,312	**11,669,720**	N/A
Octile	6,200	7,396	5,545	**5,406**

Table 2: Average expansions for first solution (using $\epsilon = 1$)

References

Chen, J.; Holte, R. C.; Zilles, S.; and Sturtevant, N. R. 2017. Front-to-end bidirectional heuristic search with near-optimal node expansions. In *Proceedings of IJCAI*.

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

Helmert, M. 2010. Landmark heuristics for the pancake problem. In *SoCS*.

Holte, R. C.; Felner, A.; Sharon, G.; Sturtevant, N. R.; and Chen, J. 2017. MM: A bidirectional search algorithm that is guaranteed to meet in the middle. *Artif. Intell.* 252:232–266.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artif. Intell.* 27(1):97–109.

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

Sturtevant, N. R. 2012. Benchmarks for grid-based pathfinding. *IEEE Trans. Comput. Intellig. and AI in Games* 4(2):144–148.