Near-Optimal Bidirectional Search

Jingwei Chen and Nathan R. Sturtevant, University of Denver
Robert C. Holte, University of Alberta, Sandra Zilles, University of Regina

UNIVERSITY of DENVER
DANIEL FELIX RITCHIE SCHOOL OF ENGINEERING & COMPUTER SCIENCE

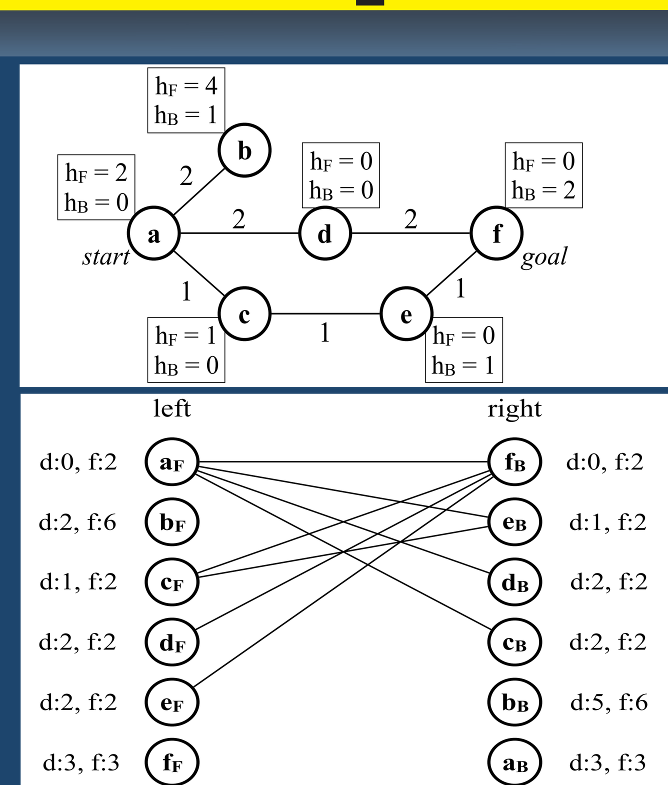
Moving AI Lab

NSF Award 1551406

Necessary Node Expansions

- Unidirectional algorithms must expand all states with $f < C^*$. Such states are called "surely expanded" (s.e.).
- Bidirectional algorithms do not have s.e. states, but s.e. pairs (Eckerle *et al.*, 2017): at least one of u and v must be expanded for all pairs with $lb(u,v) < C^*$, where

  $lb(u,v) = max\{f_F(u), f_B(v), g_F(u)+g_B(v)\}$.
- Surely Expanded pairs can be represented by a bipartite graph (Must-Expand Graph).



A sample problem instance and its Must-Expand Graph.

What Must-Expand Graph gives us:

- The set of states expanded by any admissible algorithm must be a vertex cover in Must-Expand Graph.
- A new framework to analysis the necessary node expansions for all bidirectional algorithms:
- The minimum number of node expansions among all possible algorithms is equal to the size of minimum vertex cover in Must-Expand Graph.

New Algorithm: NBS

We present a new bidirectional front-to-end algorithm, Near-Optimal Bidirectional Search (NBS), with following properties:

- It will always return optimal solution.
- It will never do more than twice the minimum necessary node expansions.
- Its bound, two, is tight.
- It can be implemented with efficient data structure to be practical.

Pseudocode of NBS

While lb_{min} < currentSolution

  Choose the pair (u,v) with min lb

  $lb_{min} = lb(u,v)$

  Forward-expand u, backward-expand v

End While

Highlights of NBS implementation:

- NBS is adapted from a greedy vertex cover algorithm (Papadimitriou and Steiglitz, 1982).
- The tie-breaking matters. We choose to break ties towards pairs with low f-cost.
- Naive implementation of pair selection needs all-pair computation, which requires $O(n^2)$ time per selection operation.
- Our efficient data structure for pair selection only needs $O(\log n)$ amortized time per selection.

Performance

The general trend:

- When the heuristic is very strong, A* performs best.
- As the heuristic get weaker, or the problems get harder, the bidirectional approaches become competitive.

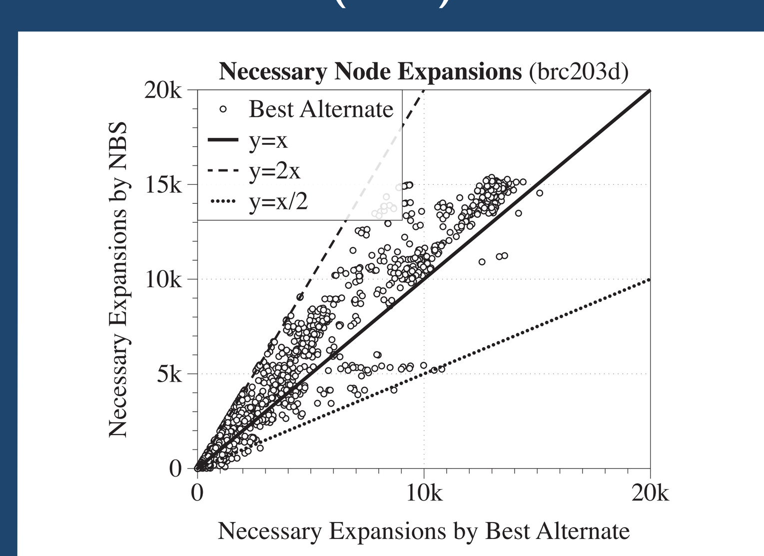
NBS is practical:

- NBS can do less node expansion and can run faster on some domains.

NBS is an insurance:

- When NBS is not the best approach, it is never far from the best; when NBS is the best approach, it could be much better than existing alternatives. NBS is the only algorithm with bounded suboptimality.

A comparison between necessary expansions by NBS(y-axis) and best alternative(x-axis) on each instance



Average running time to solve an instance (in seconds)

Domain	h	A*	BS*	MMe	NBS
DAO	Octile	0.005	0.006	0.015	0.007
Mazes	Octile	0.035	0.022	0.060	0.019
TOH4	12+2	3.23	2.44	4.17	3.54
TOH4	10+4	52.08	23.06	30.64	16.60
Pancake	GAP	0.00	0.00	0.00	0.00
Pancake	GAP-2	14.16	4.91	5.25	5.23
Pancake	GAP-3	*N/A*	212.33	72.13	77.17
15 puzzle	MD	47.68	29.59	41.38	37.67

Average states expansions to solve an instance

Domain	h	Strength	A*	BS*	MMe	NBS	MM0
DAO	Octile	+	9,646	11,501	13,013	12,085	17,634
Mazes	Octile	-	64,002	42,164	51,074	34,474	51,075
4 peg TOH	12+2	++	1,437,644	1,106,189	1,741,480	1,420,554	12,644,722
4 peg TOH	10+4	-	19,340,099	8,679,443	11,499,867	6,283,143	12,644,722
16 Pancake	GAP	+++	125	339	283	335	*unsolvable*
16 Pancake	GAP-2	-	1,254,082	947,545	578,282	625,900	*unsolvable*
16 Pancake	GAP-3	--	*unsolvable*	29,040,138	7,100,998	6,682,497	*unsolvable*
15 puzzle	MD	+	15,549,689	12,001,024	13,162,312	12,851,889	*unsolvable*