Proceedings of the Seventeenth International Symposium on Combinatorial Search (SoCS 2024)

On Parallel External-Memory Bidirectional Search (Extended Abstract)

Lior Siag[1], Shahaf S. Shperberg[1], Ariel Felner[1], Nathan R. Sturtevant[2,3]

[1] Ben-Gurion University of the Negev
[2] University of Alberta
[3]Alberta Machine Intelligence Institute (Amii)
siagl@post.bgu.ac.il, {shperbsh, felner}@bgu.ac.il, nathanst@ualberta.ca

Abstract

Parallelization and External Memory (PEM) techniques significantly enhance the capabilities of search algorithms for solving large-scale problems. While previous research on PEM has primarily centered on unidirectional algorithms, this work presents a versatile PEM framework that integrates both uni- and bi-directional best-first search algorithms.

Introduction

The intersection of parallel and external memory (PEM) within BiHS has only been explored in the context of the meet-in-the-middle (MM) algorithm (Holte et al. 2017), yielding a variant called PEMM (Sturtevant and Chen 2016) which this work builds upon. However, recent advancements in BiHS algorithms necessitate a framework for converting BiHS algorithms into corresponding PEM variants. Therefore, we introduce a flexible framework capable of integrating any UniHS or BiHS algorithm into the PEM paradigm. Subsequently, we leverage this framework to develop a PEM variant of BAE* (Sadhukhan 2013), resulting in PEM-BAE*. Empirical evaluation shows that PEM-BAE* outperforms the PEM variants of A* and the MM algorithm, as well as a parallel variant of IDA*, in solving challenging problems with significantly improved efficiency.

The PEM-BiHS Framework

We introduce a high-level framework called *Parallel External Memory Bidirectional Heuristic Search* (PEM-BiHS). We give a high-level description of PEM-BiHS together with the pseudo-code presented in Algorithm 1. PEM-BiHS initializes an OPEN and CLOSED list for each direction (line 3). These lists do not explicitly store search nodes; instead, they maintain references to files (buckets) that contain the corresponding nodes. PEM-BiHS interates through the following stages:

Halting condition (line 6): During each expansion cycle, PEM-BiHS evaluates the cost U of the current incumbent solution in comparison to the calculated lower bound LB, derived from the nodes within the open lists. If $U \leq LB$ or one of the open lists is empty, PEM-BiHS halts and returns the current solution cost. Otherwise, the search continues.

Algorithm 1: PEM-BiHS General Framework

1: **procedure** PEM-BIHS $(start, goal)$
2: $U \leftarrow \infty, LB \leftarrow$ ComputeLowerBound()
3: OPEN_F, OPEN_B, CLOSED_F,$\text{CLOSED}_\text{B} \leftarrow \emptyset$
4: Push($start$, OPEN_F) \\ create bucket and record
5: Push($goal$, OPEN_B)
6: **while** $\text{OPEN}_\text{F} \neq \emptyset \wedge \text{OPEN}_\text{B} \neq \emptyset \wedge U > LB$ **do**
7: $D \leftarrow$ ChooseDirection()
8: $b \leftarrow$ ChooseNextBucket(OPEN_D)
9: ParallelReadBucket(b, D) \\
10: RemoveDuplicates(b, CLOSED_D)
11: CheckForSolution(U, b, $\text{CLOSED}_{\overline{D}}$)
12: ParallelExpandBucket(b, OPEN_D)
13: WriteToClosed(b, CLOSED_D)
14: $LB \leftarrow$ ComputeLowerBound()
15: **return** U

Choose direction and bucket (line 7–8): Choosing the search direction D and a bucket from OPEN_D to expand.
Retrieving the bucket: Performing a parallel reading of the file containing the bucket from external memory into the internal memory (RAM). This stage involves eliminating duplicate states within the bucket.
Duplicate Detection (line 10): Eliminate duplicate nodes with other buckets in CLOSED_D.
Detect Solution (Line 11): Check if a solution was found.
Expansion (Line 12): Nodes from memory are concurrently expanded, generating children. These children are then written to their respective buckets.
Writing to disk (Line 13): Finally, the expanded nodes are written to disk, creating a new duplicate-free bucket. A reference to this bucket is inserted into CLOSED.

Experimental Results

We tested the PEM-BiHS instantiations of BAE*, A* (start to goal and the reverse), and MM. In addition, we used a public implementation of Asynchronous Parallel IDA* (AIDA*), (Reinefeld and Schnecke 1994)). We experimented on 3 domains: 15- and 24-Puzzle, and 4-Peg Towers of Hanoi (ToH4). All experiments were executed on 2 Intel Xeon Gold 6248R Processor 24-Core 3.0GHz with 2 threads each, 192 GB of 3200MHz DDR4 RAM, and 100TB SSD.

	MD		PDB	
	Time	Expansions	Time	Expansions
All instances				
AIDA*	3.45	451,421,959	0.43	7,762,927
rAIDA*	**2.44**	335,167,556	**0.37**	6,118,084
PEM-A*	102.33	56,542,721	2.01	2,724,974
PEM-rA*	84.38	43,451,519	1.85	2,302,668
PEM-MM	16.49	26,771,047	5.2	2,572,780
PEM-BAE*	6.11	**3,113,271**	3.06	**626,440**
The 10 hard instances: 3, 15, 17, 32, 49, 56, 60, 66, 82, 88				
AIDA*	22.18	2,943,505,999	2.13	46,314,389
rAIDA*	16.67	2,695,821,070	**1.93**	41,047,358
PEM-A*	901.19	350,840,875	7.8	17,124,704
PEM-rA*	786.58	308,829,220	6.67	14,371,919
PEM-MM	74.14	165,459,580	13.07	14,989,610
PEM-BAE*	**13.31**	**15,749,202**	6.05	**3,199,891**

Table 1: 15-puzzle Results. Time in seconds.

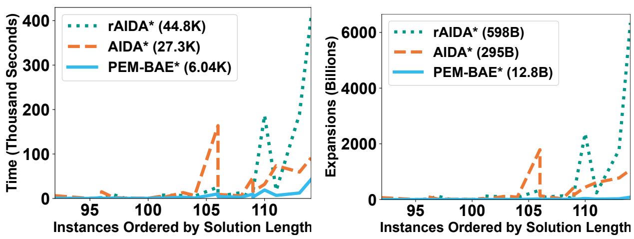

Figure 1: 24-puzzle results

15-Puzzle. Experiment on the 100 problem instances of Korf (1985). For heuristics, we used Manhattan Distance and a 3-4-4-4 additive pattern database (Felner, Korf, and Hanan 2004). As seen in Table 1, when looking at all instances, rAIDA* had the lowest runtime, while PEM-BAE* had the lowest number of expansions when using either MD or PDBs. When looking at at the 10 hardest instances, when using PDBs the trend continued, but in MD PEM-BAE* had the lowest runtime, suggesting that as the problem becomes harder, PEM and BiHS can provide an advantage.

24-Puzzle. We experimented with the first 20 24-puzzle problems of the 50 created by Korf and Felner (2002), using a 6+6+6+6 additive PDB heuristic coupled with its reflection about the main diagonal. Due to the domain size, we only compared PEM-BiHS with the AIDA* variants. Figure 1 illustrates the runtime (left) and the number of expanded nodes (right) for each instance. The instances are sorted in ascending order of solution length, serving as a (noisy) indicator of the difficulty level of each problem. The legends of the plots include the average result of each algorithm across all instances.

In general (with a few exceptions), PEM-BAE* performs the best in both node expansions and runtime. On average, PEM-BAE* expands only 4.4% of the nodes expanded by AIDA* and runs 4.5 times faster. These findings align with the observed trend in the 15-puzzle, indicating that on challenging problems, PEM-BAE* outperforms UniHS algorithms even when equipped with state-of-the-art (or near state-of-the-art) heuristics.

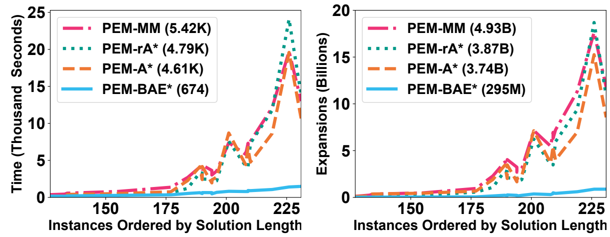

Figure 2: ToH4 16+4 results

ToH4. We examined 20 random instances (random start and goal) with 20 disks, utilizing a 16+4 additive PDB heuristic. In this domain, numerous cycles exist, posing a challenge for algorithms that lack duplicate detection, as already noted by Felner, Korf, and Hanan (2004). This issue is so severe that neither AIDA* nor rAIDA* could solve a single problem after running for days. Consequently, we only compared PEM-BAE*, PEM-A*, PEM-rA*, and PEM-MM.

The results, presented in Figure 2, highlight a significant performance gap between PEM-BAE* and the other algorithms. On average, PEM-BAE* runs 7 times faster than its UniHS counterparts and expands a factor of 12.9 fewer nodes. Notably, PEMM was approximately 1.17 times slower than both PEM-A* and PEM-rA*, and it expanded more nodes than both of them.

Acknowledgments

This work was supported by ISF grant #909/23 awarded to Shahaf Shperberg and Ariel Felner, by Israel's MOST grant #1001706842, awarded to Shahaf Shperberg, and by United States-Israel Binational Science Foundation (BSF) grant #2021643 awarded to Ariel Felner. This work was also partially funded by the Canada CIFAR AI Chairs Program. We acknowledge the support of the National Sciences and Engineering Research Council of Canada (NSERC).

References

Felner, A.; Korf, R. E.; and Hanan, S. 2004. Additive Pattern Database Heuristics. *J. Artif. Intell. Res.*, 22: 279–318.

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.

Korf, R. E.; and Felner, A. 2002. Disjoint pattern database heuristics. *Artif. Intell.*, 134(1-2): 9–22.

Reinefeld, A.; and Schnecke, V. 1994. Work-load balancing in highly parallel depth-first search. In *SHPCC*, 773–780. IEEE.

Sadhukhan, S. K. 2013. Bidirectional heuristic search based on error estimate. *CSI Journal of Computing*, 2(1-2): S1.

Sturtevant, N. R.; and Chen, J. 2016. External Memory Bidirectional Search. In *IJCAI*, 676–682. IJCAI/AAAI Press.