External-Memory Bidirectional Search

Nathan R. Sturtevant and Jingwei Chen, University of Denver

University of Denver
Daniel Felix Ritchie School of Engineering & Computer Science
Moving AI Lab
NSF Award 1551406

What is external-memory search?

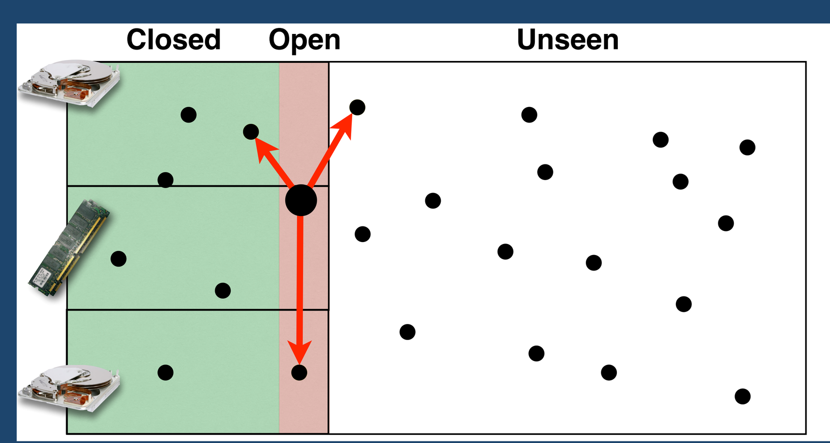
When main memory is too small, we can use external memory (e.g. a hard disk) to store states during a search

External memory has:

- High latency for random access
- High throughput
- High storage (relative to main mem.)

To use external memory, we must engineer algorithms to avoid random access.

In external-memory search, the state space is often split into buckets which can be loaded into memory for expansions. The successors of in a bucket may not be in the same bucket.



Immediately checking successors to see if they are duplicated in open/closed would require random access to disk.

External-memory search delays duplicate detection until it can be efficiently performed on many states at once.

Internal Memory:
generate → remove duplicates → write to Open

External Memory:
generate → write to Open → remove duplicates

What is bidirectional search?

Bidirectional search aims to be more efficient by searching simultaneously from the start and the goal. The MM algorithm (Holte et. al., 2016) guarantees the searches meet in the middle.

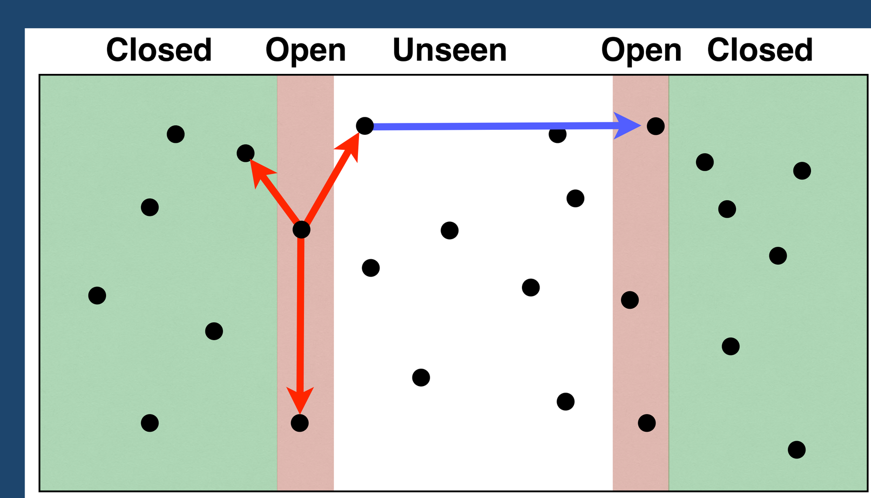
The effectiveness of bidirectional search depends on the heuristic and the state space distribution.

Bidirectional searches must check the opposite search frontier to detect if a goal is found.

Bidirectional Search
Remove next from open
  Generate successors of next
    Look for duplicates in open/closed
    Look for duplicates in opposite frontier
  Add next to closed

We call this step *solution detection.*

There are many open questions: Bidirectional search requires stronger heuristics than unidirectional search. Termination and tie-breaking strategies are not fully understood. Recent work (Holte et. al., Sharon et. al.) has improved termination conditions.



How do we combine them?

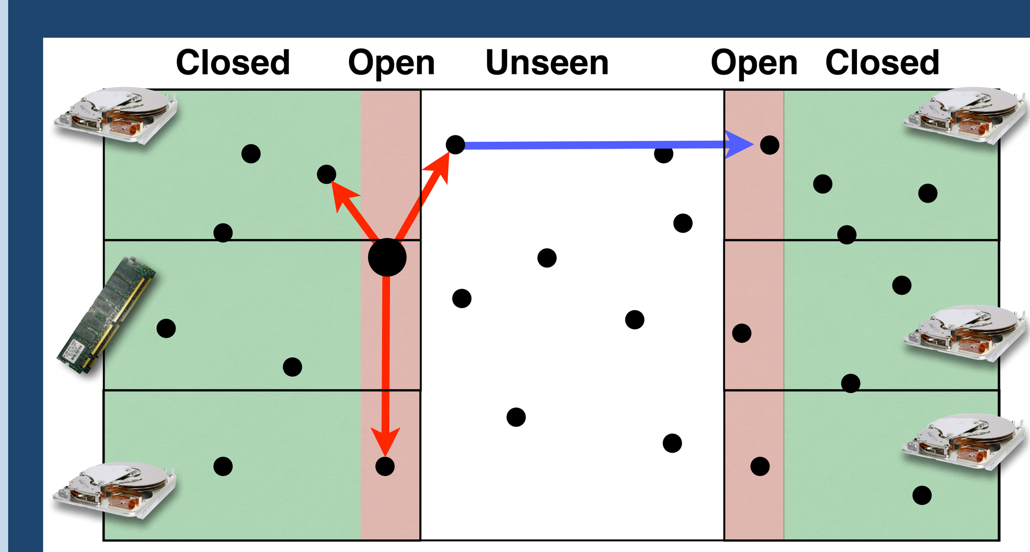
Solution detection requires random access to disk, which doesn't work with external memory search. To be effective, we must *delay solution detection* until it can be efficiently performed on many states at once (during expansion).

External-Memory Bidirectional Search
Load best bucket from open into memory
  Check for duplicates on open/closed
  Look for duplicates in opposite frontier
  For each state in bucket:
    Generate successors
    Add successors to open

We show that delaying this check is correct; it increases the efficiency of solution detection. But, solution detection is still expensive. Also, we cannot use recent improved termination rules.

We create a new algorithm, PEMM, which uses delayed solution detection, delayed duplicate detection, and parallel expansions.

We test PEMM on Rubik's cube using a variety of heuristics. These illustrate the trade-offs and need for further work on delayed solution detection.



		$PEMM_0$	PEMM	IDA*	PEMM	IDA*	PEMM	IDA*
#	Depth	0	1997	1997	888	888	8210	8210
0	16	1.01B	**166M**	0.24B	0.10B	**19.43M**	17.7M	**4.22M**
1	17	2.13B	**1.00B**	1.51B	0.87B	**0.12B**	165M	**29.76M**
2	17	2.78B	**1.54B**	8.13B	1.14B	**0.67B**	202M	**127M**
3	17	2.02B	**0.95B**	6.56B	**0.37B**	0.47B	**18.1M**	85.61M
4	18	5.77B	**2.89B**	29.69B	2.89B	**2.40B**	1.22B	**0.44B**
5	18	3.69B	**4.43B**	15.37B	2.87B	**1.04B**	1.34B	**0.21B**
6	18	3.85B	**15.05B**	41.57B	3.23B	**3.13B**	1.63B	**0.66B**
7	18	3.98B	**4.06B**	45.88B	**3.41B**	3.75B	1.42B	**0.66B**
8	18	2.88B	**2.93B**	58.35B	**2.99B**	5.00B	1.55B	**1.17B**
9	18	12.23B	**2.91B**	70.31B	**2.90B**	4.78B	1.07B	**1.01B**
S	20	38.08B	-	-	**38.08B**	116B	35.61B	**24.59B**

PEMM performs fewer node expansions than IDA* on hard problems / weak heuristics, although it expands states more slowly than IDA*.

		No heuristic: $PEMM_0$					
#	Depth	Time(s)	% Exp	% I/O	% DSD	# Exp.	Disk
0	16	1,063	78.19	21.81	12.28	1.00	133GB
1	17	3,683	45.93	54.07	47.12	2.13	281GB
2	17	6,031	36.47	63.53	58.86	2.78	367GB
3	17	3,362	48.32	51.68	44.03	2.02	266GB
4	18	11,681	49.70	50.30	36.06	5.77	774GB
5	18	8,245	40.33	59.67	49.37	3.69	487GB
6	18	8,031	44.19	55.81	43.57	3.85	506GB
7	18	8,276	45.78	54.22	42.85	3.98	539GB
8	18	6,386	38.42	61.58	55.00	2.88	388GB
9	18	22,643	56.69	43.31	17.33	12.23	1.6TB
S	20	100,816	33.04	66.96	56.97	38.08	5.0TB

With no heuristic, PEMM spends significant time doing solution detection. Current heuristics don't significantly improve PEMM's performance.