



# ***Optimizing LPA\* Searches for Certain Domains***

cs7612 Project.

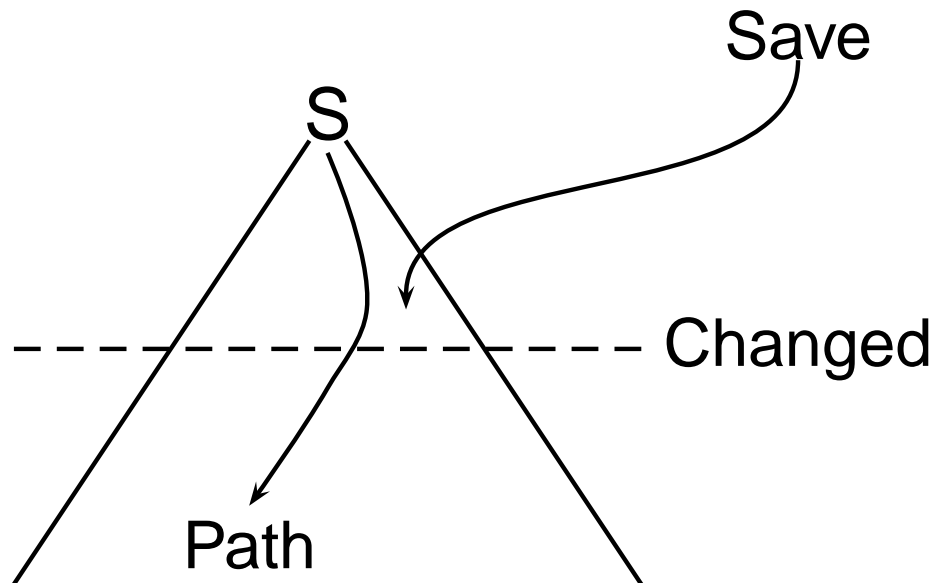
David “Dave” Dagon

dagon@cc.gatech.edu

College o’ Computing  
Georgia Institute of Technology

- Replanning occurs when knowledge changes
- E.g., robots learn new terrain; environments with multiple actors
- Fast replanning endeavors to use as much of the previous plan as possible.

- General Idea



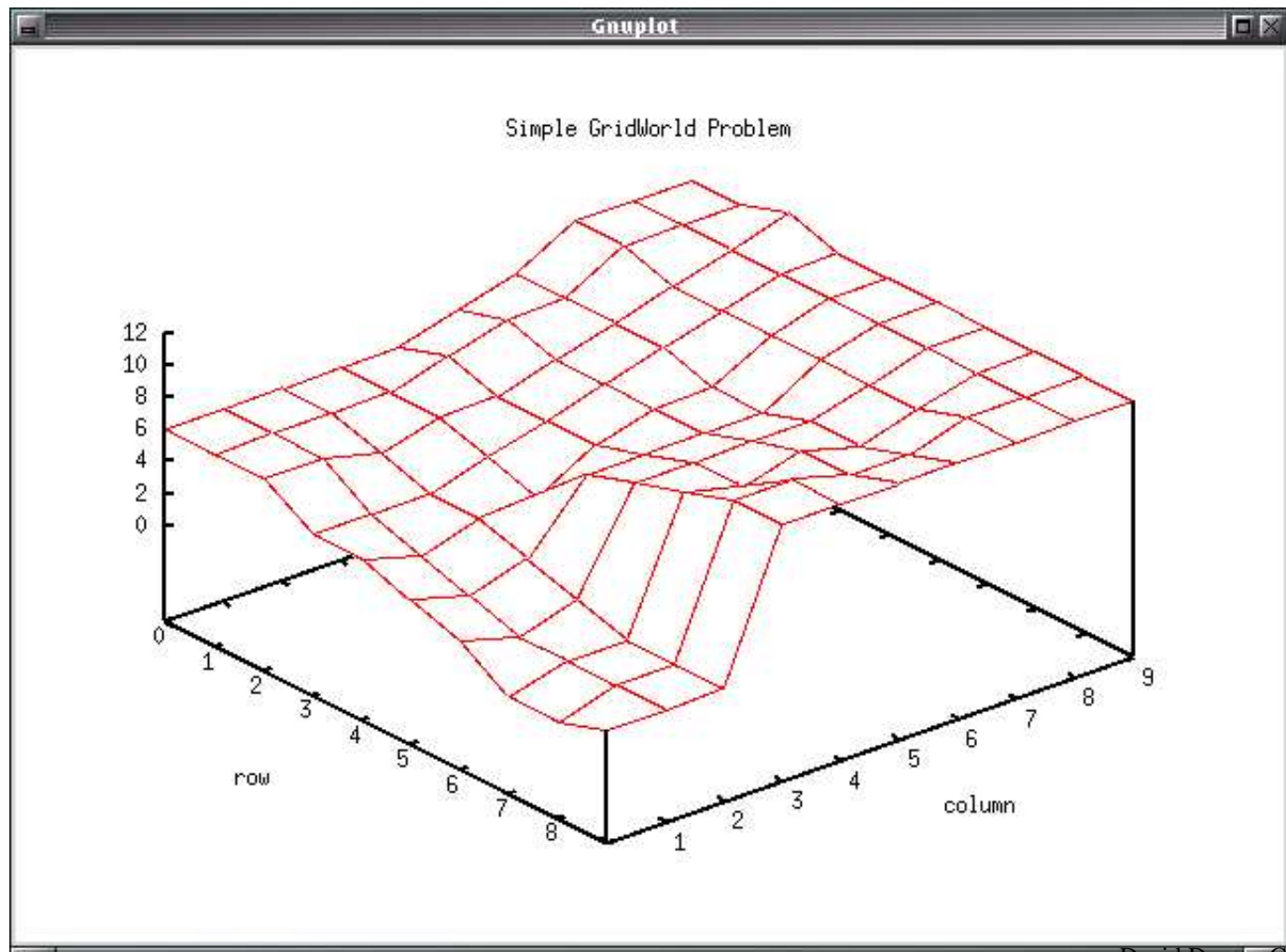
- Goal: use as much of the old search as possible

- Recall how LPA\* works

```
#####  
#      #  #  
## #### g  
# ... ..#  
#.#. #.# #  
#.#  #  #  
#.#    #  
s # # # #  
# #  ## #  
#####
```

- Suppose we added a barrier?

- Uphill invalidation; corrections from downhill



- What are the costly parts of LPA\* (or any A\* search)?
  - Cost of searching forward (often using modified priority queue structure).
  - Number of nodes that must be addressed

## ***Idea #1: Meta-Layer***

---

- How can we reduce the number of nodes affected by an update?
- One idea is to aggregate nodes that share certain properties.
- Insight: look for monotonically increasing sets of nodes that share other properties.

# Node Sequence Tables

- Insight: identify sequences that share key properties allowing for aggregation.

	0	1	2	3
A	3	2	1	0
B	2	3	4	2
C	2	4	1	3
D	3	2	2	2
E	4	3	3	2
F	5	2	4	2
G	6	2	5	2
H	7	2	6	2
I	8	8	7	7
J	2	3	4	2

# Node Sequence Tables

- Insight: identify sequences that share key properties allowing for aggregation.

	0	1	2	3
A	3	2	1	0
B	2	3	4	2
C	2	4	2	3
D	3	2	3	2
E	4	2	3	2
F	5	2	4	2
G	6	2	5	2
H	7	2	6	2
I	8	4	7	3
J	8	3	4	2

Segment Table:

$\{C1, D1, E1, F1\}$

$\{C3, D3, E3, F3\}$

# *Identifying Segments*

- All node segments must meet the following criteria:
  - All nodes in a segment must have the same order (2).
  - All nodes in the segment must have different distances.
  - The set of nodes must constitute a complete path.
  - Segments must have three or more nodes.
  - The nodes in the segment are sorted from shortest distance to greatest.
  - NEW: Segments cannot stradle the goal node.

# Node Sequence Tables

- Invalidation enters at one end of the segment; correcting values return through the other.
- Only the last (largest) element is added to LPA\* queue.

	0	1	2	3
A	3	2	1	0
B	2	3	4	2
C	2	4	2	2
D	3	2	3	2
E	4	2	4	2
F	5	2	4	2
G	6	2	5	2
H	7	2	6	2
I	8	4	7	3
J	8	8	7	7

Diagram illustrating a Node Sequence Table with rows A through J and columns 0 through 3. The table contains numerical values. A shaded gray cell is present at row B, column 3. A curved arrow labeled  $\infty_{\sigma}$  points from the shaded cell to the right. Vertical hatched bars are present in column 1 (rows C-F) and column 3 (rows C-F). Black shaded regions cover the entire column 0 and the cells (B,0), (C,0), (D,0), (E,0), (F,0), (G,0), (H,0), (I,0), and (J,0).

- For *certain* domains, we reduce the number of nodes handles by LPA\* replanning.
- Savings is (- segment-length 1)
- Applications: maze domains, domains where paths have long sequences, domains that divide actions into very small sequential steps.

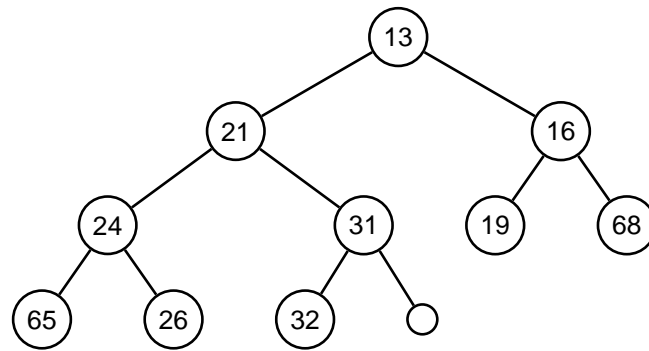
## ***Idea #2: Caching Search Information***

---

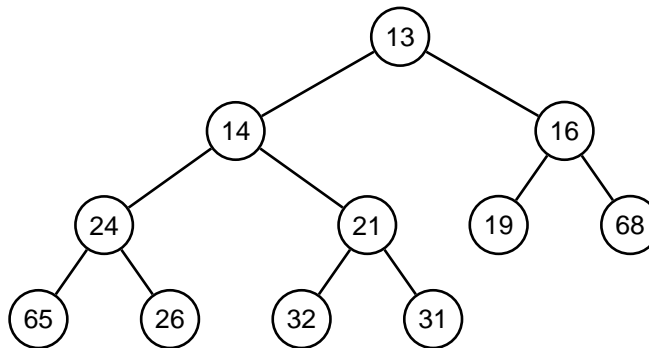
- How can we speed up replanning's use of priority queue structures?
- In many cases, a change causes us to replan many of the same steps over and over again.
- If only there were a heap that 'remembered' what it looked like in a previous state...
- Idea: A heap that caches state

# Caching Heap

- Observe the changes from a single heap insert operation:



	13	21	16	24	31	19	68	65	26	32	-	-	-	-
--	----	----	----	----	----	----	----	----	----	----	---	---	---	---



	13	14	16	24	21	19	68	65	26	32	31	-	-	-
--	----	----	----	----	----	----	----	----	----	----	----	---	---	---

# Caching Heaps

- Create a cache table that stores deltas from insertions.

	13	21	16	24	31	19	68	65	26	32	-	-	-	-	-
--	----	----	----	----	----	----	----	----	----	----	---	---	---	---	---

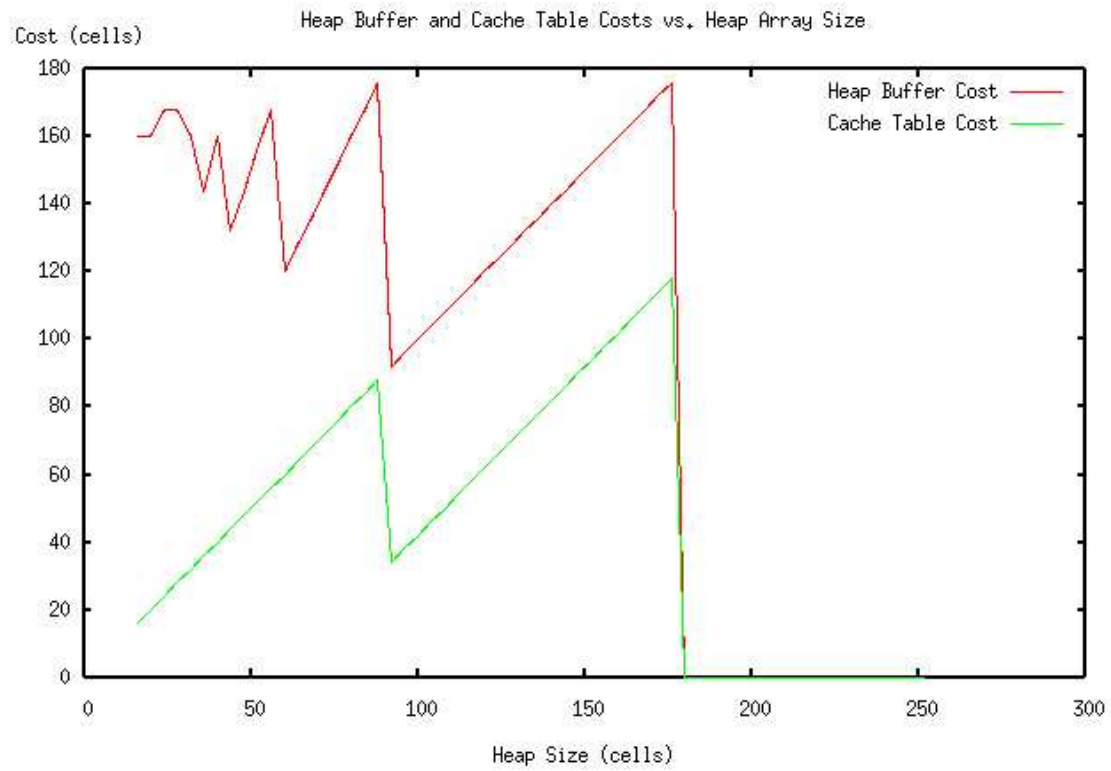
	13	14	16	24	21	19	68	65	26	32	31	-	-	-	-
--	----	----	----	----	----	----	----	----	----	----	----	---	---	---	---

index	prior
2	21
5	31
11	nil

# Cache Heap Operation

- What size should the cache table grow to?
  - We could save a copy of the heap after every insertion.
  - Or we could just save a delta of every insert.
- An intermediate approach is to save complete copies of the heap every so often, such as when the size of the cache table is  $\geq$  the size of the heap.
- There's a balance between the two...

# Implementation



# Conclusions

- Meta-data graph information can help speed up some operations.
- A heap with memory appears promising.
- Still open: testing caching heap in various domains.
  - The effort saved is offset by the size of the data you keep, either stored in the deltas, or in duplicate heap arrays.
  - Some domains might need the speed of saving more frequent backups; some might need the space, and rely on larger (slower) delta tables.