Tetromino BinPacking Project
By Robert Jackson

Brief Summary

- **Functionality implemented in Final System, including description of AI methods**
  - AI Search Methods
    - Heuristics
      - Number of Pieces left
      - Number of Holes
    - Search Algorithms
      - Depth First Search
      - Breadth First Search
      - Recursive Depth Limited Search
      - Iterative Deepening Search
      - Simulated Annealing
      - Hill Climbing
      - A*
      - Greedy Best First Search

  - UI
    - The "Bin" in which the blocks will be placed
    - Options to choose how many of each tetromino will need to be placed in the "Bin".
    - An option to randomly generate a starting problem set of tetrominos
    - An option to start the search algorithm.
    - Options to set height and width of bin

  - Model
    - Bin – a rectangular 2 Dimensional bin in which pieces are added at the top
    - Tetrominos – may be rotated and placed on floor or on top of another piece. An open source tetris game will be used for its tetris board and tetromino classes (found here: http://www.ibm.com/developerworks/java/library/j-tetris/).

- **Functionality implemented in final system**
  - Optimizations to cut runtime and runspace.
    - Node expansion now uses a **multithreaded actions() function**. On machines with multiple CPUs, speedup can be quite great on a large board.
    - 2D storage of a board state is now done using **Parallel Colt's "Sparse Matrix"** implementation. Using a sparse matrix as the storage method greatly lowers run space usage. By using a parallel version of the sparse matrix, the performance hit is marginalized.

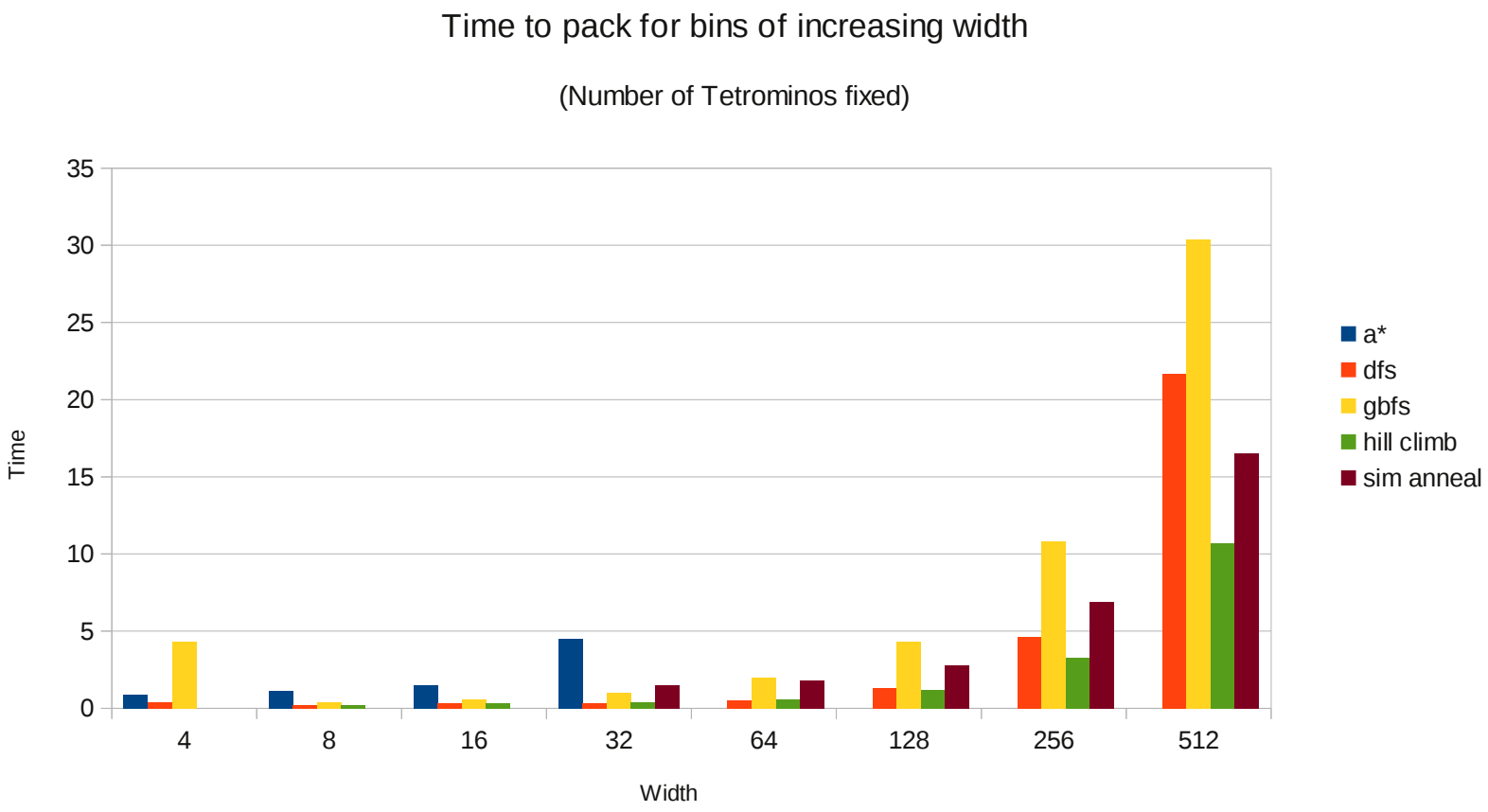
  - Search Metrics
    - Run Time to find solution for each search algorithm.
    - Run Space need to find solution for each search algorithm.

Results of System Evaluation

- **Evaluation of AI methods in final system**
  - Benchmark suite of test cases
  - Comparison of search results for benchmark showing Runtime and Runspace for each test.

Time to pack a fixed amount of tetrominos, with different bin widths (area fixed). Winner is highlighted in Green.

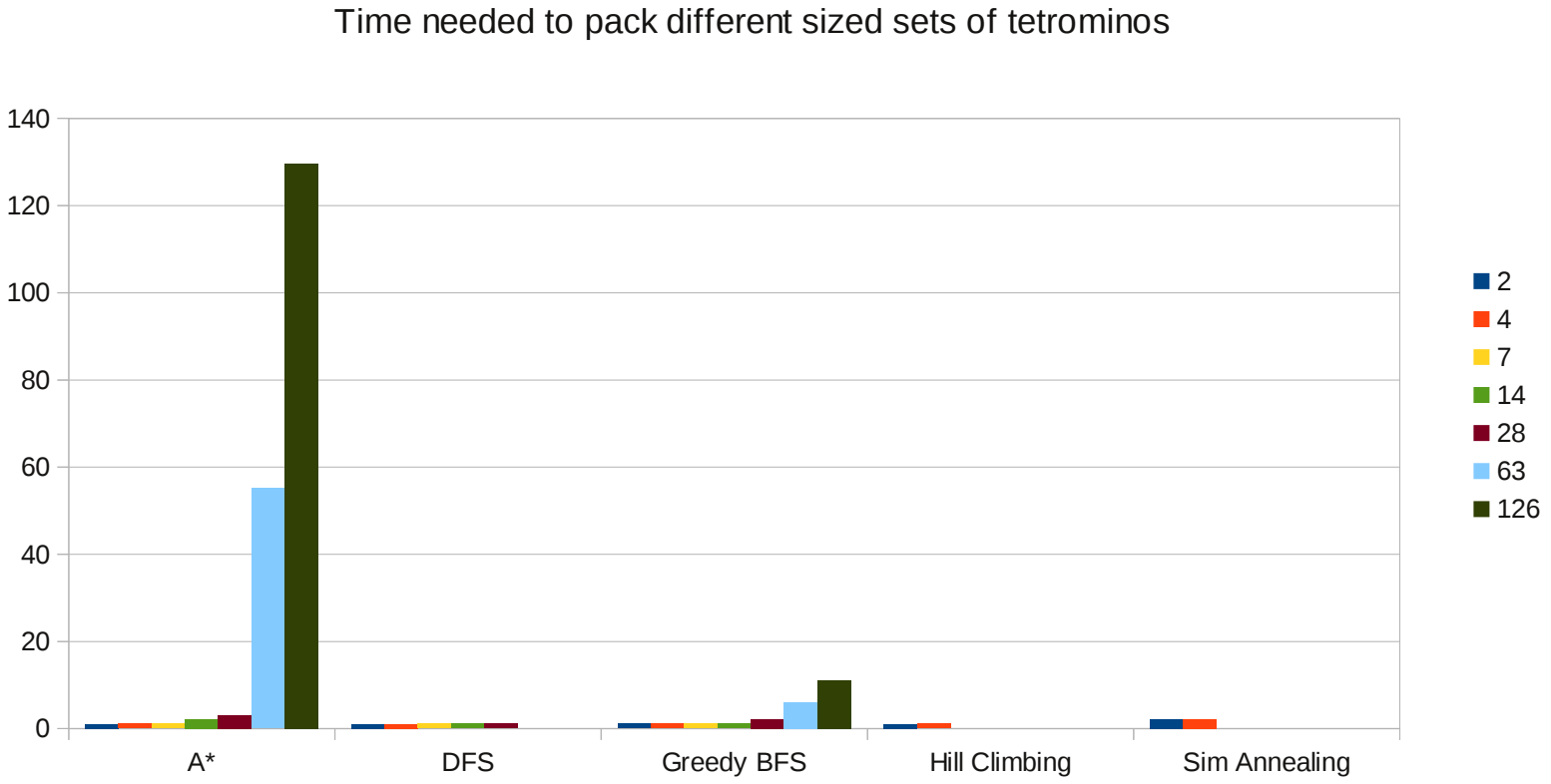
height	width	area	a*	dfs	gbfs	hill climb	sim anneal
300	4	1200	0.9	0.4	4.3		
150	8	1200	1.1	0.2	0.4	0.2	
75	16	1200	1.5	0.3	0.6	0.3	
38	32	1200	4.5	0.3	1	0.4	1.5
19	64	1200		0.5	2	0.6	1.8
10	128	1200		1.3	4.3	1.2	2.8
5	256	1200		4.6	10.8	3.3	6.9
3	512	1200		21.7	30.4	10.7	16.5



Depth First Search and Hill Climbing produced the quickest results.

Time to pack increasing number of tetrominos (using a uniform distribution of tetrominos). Winner is highlighted in Green.

Algorithm	2	4	7	14	28	63	126
A*	1.07	1.09	1.09	2.09	3.14	55.14	129.68
DFS	1.05	1.07	1.08	1.12	1.13		
Greedy BFS	1.13	1.09	1.102	1.09	2.1	6.13	11.13
Hill Climbing	1.07	1.08					
Sim Annealing	2.12	2.12					



Depth First Search and Greedy Best First Search produced the quickest results. Oddly, A* performs worse than Greedy BFS even though they use the same heuristics.

Conclusion

For the task of packing tetrominos, Depth First Search and Greedy Best First Search performed the best given a uniform distribution of tetrominos. To do better than this, better heuristics will need to be developed or a better utility function must be created.

Runspace was greatly reduced by using a sparse matrix in place of a 2D array. This allows for much larger bins to be packed by saving memory.