Pop Vlad
Popa Alex Ovidiu
936/1

PDP Semester Project – 15-Puzzle Problem

1.Problem Statement, Solution

Requirements

15 Puzzle

Proposed solution: use **IDA*** (Iterative deepening A*) to find the solution which requires the minimum number of steps to reach the complete puzzle. (I.e., shortest one)

While the standard iterative deepening depth-first search uses search depth as the cutoff for each iteration, the IDA* uses the more informative **f(n) = g(n) + h(n)**, where g(n) is the cost to travel from the root to node n and h(n) is a problem-specific heuristic estimate of the cost to travel from n to the goal.

In our case, the chosen h(n) is the **Manhattan Distance** of the current board state with respect to the solution one, g(n) is the **number of steps** made so far, meaning, the number of times the empty tile has been moved.

2. Details of implementation

The search has been parallelized using a Java ExecutorService with a thread pool of fixed size, and the idea is that Java Futures are being submitted recursively to the service, and at each submission the current number of steps is increased by one (I.e. we “advance” with one move).

Pop Vlad
Popa Alex Ovidiu
936/1

The workload is assigned evenly to each thread, in the sense that each submitted task receives nrThreads/nrOfUnexploredNextMoves threads, and when nrThreads reaches 0, the search becomes sequential, similarly to what happens in parallelizing algorithms such as Merge Sort/ Karatsuba's Multiplication.

When the futures finish, the smallest length of the path is returned and, if we've reached a solution, the algorithm finishes. If not, a new minimum bound is set the parallel search begins again.

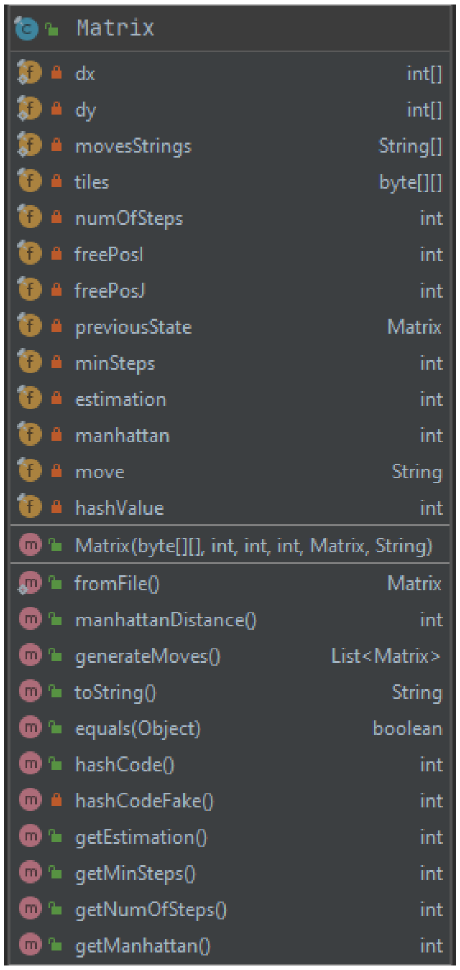
The distributed algorithm uses the MPJ Express Java library which implements the MPI Interface. The number of nodes is passed as a command line argument, with one master node and several worker nodes.

Work is assigned to each worker by performing one or more generateMoves() from the root node recursively, until the total workload gets as close as possible to the number of nodes.

When each worker receives its state, it begins searching from that point on using the iterative A* sequential implementation, and sends each solution along with its number of steps to the master function. The master function checks whether a solution has been found, and if it has, all the workers are signaled to stop searching. If not, the min bound is reset and the search continues.

Our matrix class represents one puzzle state, I.e. board.

Pop Vlad
Popa Alex Ovidiu
936/1



Matrix

dx	int[]
dy	int[]
movesStrings	String[]
tiles	byte[][]
numOfSteps	int
freePosI	int
freePosJ	int
previousState	Matrix
minSteps	int
estimation	int
manhattan	int
move	String
hashValue	int
Matrix(byte[][], int, int, int, Matrix, String)	
fromFile()	Matrix
manhattanDistance()	int
generateMoves()	List<Matrix>
toString()	String
equals(Object)	boolean
hashCode()	int
hashCodeFake()	int
getEstimation()	int
getMinSteps()	int
getNumOfSteps()	int
getManhattan()	int

3.Software, Hardware, Testing

Used software: Java, IntelliJ Idea

Used hardware:

- Intel Core i7-4790 CPU @ 3.60GHz, 4 Cores, 8 Logical Processors
- 16GB RAM

Strategy	Time Elapsed	Difficulty	Nodes/Threads
Parallelized	348ms	40 steps	1
Parallelized	240ms	40 steps	5
Parallelized	4010ms	51 steps	5
Parallelized	4280ms	51 steps	10
Parallelized	8658ms	56 steps	10
Parallelized	67372ms	61 steps	12
Parallelized	58367ms	61 steps	20
Parallelized	3573822ms	68 steps	20
Distributed	387ms	40 steps	2
Distributed	497ms	40 steps	5
Distributed	4197ms	51 steps	5
Distributed	4606ms	51 steps	11
Distributed	9439ms	56 steps	11
Distributed	83328ms	61 steps	13
Distributed	94448ms	61 steps	21
Distributed	3691949ms	68 steps	21

4.Conclusions:

The problem can be solved relatively easy using the IDA* algorithm, and parallelizing the solution helps with the execution time when larger search spaces occur.

Distributing the search across several nodes yields approximately the same time as with one node, although an overhead which represents the time spent creating the nodes and sending/receiving data needs to be considered, as with any distribution environments.

Pop Vlad
Popa Alex Ovidiu
936/1

5.Bibliography/useful resources:

http://kociemba.org/themen/fifteen/fifteensolver.html

http://www.ic-net.or.jp/home/takaken/e/15pz/index.html