A* Algorithm - Report

Nicolas Boileau, Simon Stastny

September 26, 2012

1 General A* algorithm

We decided to create our own implementation of A* algorithm for better learning outome. Our general A* algorithm implementation can be found in the package `edu.ntnu.simonst.tdt4136.astar` and contains the following classes:

- Class `BestFirstSearch` with general A* algorithm implementation
- Class `SearchNode` for general search-node used in the algorithm
- Class `SearchState` for general search-state used in the algorithm
- Class `Fringe` used to store nodes in agenda (list of unexpanded nodes)

Apart from those general classes, this package contains class `App` which is used to run both puzzles from command-line using maven.

To run the puzzles, please use maven command `mvn exec:java` inside the project folder, or run the project within NetBeans (maven enabled).

2 Fractions puzzle

2.1 Initial state of the puzzle

The initial state of the puzzle is state identified by permutation `123456789` which represents following fraction:

$$\frac{1234}{56789}$$

2.2 Description of a goal state

This puzzle involves finding fractions equivalent to the following fractions:

$$\frac{1}{2} \quad \frac{1}{3} \quad \frac{1}{4} \quad \frac{1}{5} \quad \frac{1}{6} \quad \frac{1}{7} \quad \frac{1}{8} \quad \frac{1}{9}$$

For example for the first fraction (½) the goal state would have permutation `729314586`, because following two fractions are equal:

$$\frac{7293}{14586} = \frac{2^2 \times 3 \times 641}{2^3 \times 3 \times 641} = \frac{1}{2}$$

Similarily for other fractions there exist other permutations of `123456789` which represent fractions equal to them respectively.

2.3 Method of assessing arc costs

Since we are not really interested in the path to the goal node as much as we are interested in the goal-node's state itself, we are not concerned about the cost of the solution (lenght of the path). However, we are still trying to work in an optimal way and thus we would like to get to the goal taking as few steps as possible.

For this reason, the arc cost for transition from one state to another is fixed value 1 (in the inner code 100 000 000, because of precision problems related to our heuristic function), hence the total cost of path from root to goal is equal to number of steps we have taken.

2.4 Heuristic function description

As long as states are representing fractions, i.e. numbers, the heuristic for estimating cost from current state to goal state could be something so simple as mathematical difference of those two fractions.

Since all the fractions possible in this puzzle have values greater or equal to 0.0124943046625829 and lesser of equal to 0.8, the mathematical difference of any two of them would be a double value somewhere between 0 and 1.

$$\forall x \in possiblefractions, \frac{1234}{98765} \leq x \leq \frac{9876}{12345}$$

We need to map those differences to cardinal numbers in a way that preserves the order and where no difference would be equal to 0, the smallest difference would be equal to 1 and big differences would be mapped to big numbers.

Our solution calculates the difference of these two fractions (as a `double`), multiplies the value by 100 000 000, takes the integer part of this new value and subtracts 10. This results into the smallest difference being assigned to an estimation of 1, and bigger differences scaled up to somewhere below nearly 800 000 000.

Those big numbers are the reason why the arc costs are multiplied by 100 000 000. We know that to get from the initial state (`123456789`) to any other state, we need less than 8 permutations. Then we considered the fact that the bigger difference that we can get from our method `getDistance(fraction a, fraction b)` is around 800 000 000. So we decided to map these two variables together: 1 permutation costs 100 000 000. Our heuristic function is thus based on this: we consider that the bigger the mathematical difference between the fraction and the goal is, the more permutations we will have to make to reach our goal. This is naturally not always true, but it is an estimation that we considered acceptable.

$$costestimate(A, GOAL) = |GOAL - A| \times 100000000 - 10$$

2.5 Successor generation procedure

Successor nodes are generated from current node with a simple switch operator. This switch operator takes the permutation of the node's state (which is a permutation of digits from 1 to 9) and switches two digits to make a new state. Since the first digit can be switched with 8 others to make a new permutations, the second digit with 7 others (because switching with the first digit was done already), third one with 6 others..., this generates 36 (8+7+...+1) new permutations, and each one of them is assigned to a successor node.

This leaves us with 36 successor nodes to each node. At least one of those successor nodes is already closed (because it is the current node's parent). We evaluate the remaining nodes' cost estimates and decide which node to expand in the next turn.

2.6 Overview description of a solution

This puzzle is searching for solutions for 8 different fractions, described in 2.2. Those solutions are listed in the subsections below.

2.6.1 Detailed solution for ½

For the first fraction (½) the solution permutation 729314586 was found. This is a solution, because the fraction it represents is equal to goal-state's fraction.

$$\frac{7293}{14586} = \frac{1 \times 7293}{2 \times 7293} = \frac{1}{2}$$

This solution was found in just 5 steps listed here.

- **root** state: 1234/56789 (0.02172956) We expand the root node to create 36 children. Switching 1 and 5 seems to be the best solution (the estimated cost from that state to the solution is 47 827 033).
- state: 5234/16789 (0.311751742) We expand this node. Switching 5 and 7 seems to be the best solution (the estimated cost from that state to the solution is 18 824 815).
- state: 7234/16589 (0.436072096) We expand this node. Switching 3 and 9 seems to be the best solution (the estimated cost from that state to the solution is 6 392 780).

- state: 7294/16583 (0.439848037) We expand this node. Switching 6 and 3 seems to be the best solution (the estimated cost from that state to the solution is 6 015 186).
- state: 7294/13586 (0.536876196) We expand this node. Switching 4 and 3 seems to be the best solution (the estimated cost from that state to the solution is 3 687 609).
- **goal** state: 7293/14586 (0.5) We reached our goal!

2.6.2 Solution for ⅓

For the fraction ⅓, the solution permutation 582317469 was found. This is a solution, because the fraction it represents is equal to goal-state's fraction.

$$\frac{5823}{17469} = \frac{1 \times 5823}{3 \times 5823} = \frac{1}{3}$$

.

This solution was found in just 6 steps listed here.

- **root** state: 1234/56789 (0.02172956)
- state: 5234/16789 (0.311751742)
- state: 5234/17689 (0.295890101)
- state: 5834/17629 (0.330931987)
- state: 5834/17269 (0.337830795)
- state: 5824/17369 (0.335310035)
- **goal** state: 5823/17469 (0.333333333)

2.6.3 Solution for ¼

For the fraction ¼, the solution permutation 579623184 was found. This is a solution, because the fraction it represents is equal to goal-state's fraction.

$$\frac{5796}{23184} = \frac{1 \times 5796}{4 \times 5796} = \frac{1}{4}$$

.

This solution was found in just 6 steps listed here.

- **root** state: 1234/56789 (0.02172956)
- state: 1534/26789 (0.057262309)
- state: 7534/26189 (0.287678033)
- state: 5734/26189 (0.218946886)

- state: 5736/24189 (0.237132581)
- state: 5796/24183 (0.239672497)
- **goal** state: 5796/23184 (0.25)

2.6.4 Solution for ⅕

For the fraction ⅕, the solution permutation 9237/46185 was found. This is a solution, because the fraction it represents is equal to goal-state's fraction.

$$\frac{9237}{46185} = \frac{1 \times 9237}{5 \times 9237} = \frac{1}{5}$$

.

This solution was found in just 4 steps listed here.

- **root** state: 1234/56789 (0.02172956)
- state: 9234/56781 (0.162624822)
- state: 9235/46781 (0.197409205)
- state: 9235/46187 (0.199948037)
- **goal** state: 9237/46185 (0.2)

2.6.5 Solution for ⅙

For the fraction ⅙, the solution permutation 294317658 was found. This is a solution, because the fraction it represents is equal to goal-state's fraction.

$$\frac{2943}{17658} = \frac{1 \times 2943}{6 \times 2943} = \frac{1}{6}$$

.

This solution was found in just 6 steps listed here.

- **root** state: 1234/56789 (0.02172956)
- state: 2134/56789 (0.0375777)
- state: 2534/16789 (0.150932158)
- state: 2934/16785 (0.174798928)
- state: 2934/17685 (0.165903308)
- state: 2934/17658 (0.166156983)
- **goal** state: 2943/17658 (0.166666667)

2.6.6 Solution for ⅐

For the fraction ⅐, the solution permutation 761453298 was found. This is a solution, because the fraction it represents is equal to goal-state's fraction.

$$\frac{7614}{53298} = \frac{1 \times 7614}{7 \times 7614} = \frac{1}{7}$$

.

This solution was found in just 5 steps listed here.

- **root** state: 1234/56789 (0.02172956)
- state: 7234/56189 (0.12874406)
- state: 7634/52189 (0.146276035)
- state: 7624/53189 (0.143337908)
- state: 7624/53198 (0.143313658)
- **goal** state: 7614/53298 (0.142857143)

2.6.7 Solution for ⅛

For the fraction ⅛, the solution permutation 817465392 was found. This is a solution, because the fraction it represents is equal to goal-state's fraction.

$$\frac{8174}{65392} = \frac{1 \times 8174}{8 \times 8174} = \frac{1}{8}$$

.

This solution was found in just 5 steps listed here.

- **root** state: 1234/56789 (0.02172956)
- state: 8234/56719 (0.145171812)
- state: 8234/65719 (0.125291012)
- state: 8134/65729 (0.123750552)
- state: 8134/65792 (0.123632053)
- **goal** state: 8174/65392 (0.125)

2.6.8 Solution for ⅑

For the last fraction (⅑) the solution permutation 638157429 was found. This is a solution, because the fraction it represents is equal to goal-state's fraction.

$$\frac{6381}{57429} = \frac{1 \times 6381}{9 \times 6381} = \frac{1}{9}$$

.

This solution was found in just 5 steps listed here.

- **root** state: 1234/56789 (0.02172956)
- state: 6234/51789 (0.120373052)
- state: 6234/57189 (0.109006977)
- state: 6231/57489 (0.108385952)
- state: 6831/57429 (0.118946874)
- **goal** state: 6381/57429 (0.111111111)

3 Checkers puzzle

3.1 Initial state of the puzzle

The initial states of this puzzle are :

- `[BBB AAA]` for a game of 3 versus 3 checkers
- `[BBBBBB AAAAAA]` for a game of 6 versus 6 checkers
- `[BBBBBBBBBBBB AAAAAAAAAAAA]` for a game of 12 versus 12 checkers

We named `A` the black checkers and `B` the red ones.

3.2 Description of a goal state

The goal states are respectively :

- `[AAA BBB]`
- `[AAAAAA BBBBBB]`
- `[AAAAAAAAAAAA BBBBBBBBBBBB]`

3.3 Method of assessing arc costs

As every arc is equivalent to moving 1 checker, we fixed the value of an arc to 1.

3.4 Heuristic function description

Our heuristic function calculates the number of differences between the current state and our goal, i.e the number of checkers that are not at the right position. To do this, we start by taking the current state and we assign numbers to each checker to create a chain of characters made of numbers. A `red` checker is equal to 1, the `space` is equal to 2 and a `black` checker is equal to 3. We do the same thing with the goal state. Then we sum the absolute value of the substraction of each corresponding case of the two chains. It gives us the number of differences between the two states (that we can divide by 2 because the number of differences is always an even number).

$$costestimate(A, GOAL) = \frac{1}{2} \times \sum_i |A_i - GOAL_i|$$

where A_i is i-th digit in permutation A, and $GOAL_i$ is i-th digit in permutation GOAL.

3.5 Successor generation procedure

To generate the successors states when expanding a node, we use a function that adds children by trying to apply the 4 different kinds of move that we could do on the current state. These moves are:

- `switch the empty space with checker 1 case to the left`
- `switch the empty space with checker 1 case to the right`
- `switch the empty space with checker 2 cases to the left`
- `switch the empty space with checker 2 cases to the right`

If we end up in an unvalid case, then the child is not created. If we end up on a node that already exists, then we don't add it.

3.6 Overview description of a solution

For the first checkers puzzle with only 6 checkers, our algorithm found a solution in 15 steps, which is the optimal solution.

Path reconstruction step by step from the root `[AAA BBB]` to the goal `[BBB AAA]`:

- **root** state 0: `[AAA BBB]` We expand the root node to create 4 children:
  - `[AA ABBB]` -> estimated cost from this node to the goal is 6
  - `[AAAB BB]` -> estimated cost from this node to the goal is 6
  - `[A AABBB]` -> estimated cost from this node to the goal is 6
  - `[AAABB B]` -> estimated cost from this node to the goal is 6

As all solutions are equal, we choose the first one. We move the empty space 1 case to the left.

- state 1: `[AA ABBB]` We expand this node to create 2 new children:
  - (`[A AABBB]` -> Already created)
  - (`[AAA BBB]` -> Already expanded)
  - `[ AAABBB]` -> estimated cost from this node to the goal is 6
  - `[AABA BB]` -> estimated cost from this node to the goal is 5

  Moving the empty space 2 cases to the right seems to be the best solution.
- state 2: `[AABA BB]` We expand this node to create 3 new children:
  - `[AAB ABB]` -> estimated cost from this node to the goal is 4
  - `[AABAB B]` -> estimated cost from this node to the goal is 5
  - (`[AA ABBB]` -> Already expanded)
  - `[AABABB ]` -> estimated cost from this node to the goal is 5

  Moving the empty space 1 case to the right seems to be the best solution.
- state 3: `[AABAB B]` And so on...
- state 4: `[AAB BAB]`
- state 5: `[A BABAB]`
- state 6: `[ ABABAB]`
- state 7: `[BA ABAB]`
- state 8: `[BABA AB]`
- state 9: `[BABABA ]`
- state 10: `[BABAB A]`
- state 11: `[BAB BAA]`
- state 12: `[B BABAA]`
- state 13: `[BB ABAA]`
- state 14: `[BBBA AA]`
- **goal** state 15: `[BBB AAA]`

For the variant of the checkers puzzle using 12 checkers, our algorithm finds a solution in 48 moves.

For the variant of the checkers puzzle using 24 checkers, our algorithm runs for a very long time... Too long actually, so we never ended up with a solution.