

Solving the Rush Hour Game: A Genetic Approach Using Variable Length Individuals

Dan Schlegel (dschlege@cs.oswego.edu)

Computer Science Program
Oswego, NY 13126 USA

Abstract

A genetic algorithm which is built around the concept of individuals of varying length is introduced as a method to solve the PSPACE-complete Rush Hour game. Also discussed are similar attempts at using variable length genetic algorithms to solve problems other than Rush Hour and existing methods of solving the Rush Hour game.

Keywords: Variable length individuals, Rush Hour, genetic algorithm, solving PSPACE-complete games.

Introduction

This paper describes in progress work surrounding using a genetic algorithm to solve the Rush Hour board game. To do this we use a concept of individuals of variable length. The effects of this variable length concept will be discussed in detail in regards to its effect on the population, mutation, and crossover operators. This as far as the author can tell has not been done before with respect to the Rush Hour game, with only human solved (interactive) games (Puzzles.com) and brute force algorithms (Theiling 2007) being developed so far.

Background

Genetic Algorithms

Genetic Algorithms work on the idea from biology that organisms evolve over time to behave optimally in their environment. Genetic algorithms follow this idea fairly closely, including traditionally a population of

individuals which undergo operations to imitate mating and are subjected to a fitness metric which is akin to seeing how they survive in their environment. Some of these operations to which the individuals are subjected are crossover and mutation, commonly used words in biology to discuss what DNA undergoes during mating. These algorithms are used in AI often to find solutions to problems which may not have a readily available algorithm for determining the solution. Perhaps the most common such problem is the traveling salesperson problem in which a salesperson must travel to several cities in different geographic regions and wonders what the optimal path would be.

The Rush Hour Game

The Rush Hour game is a board game in general played on an n -by- n grid (in our case 6 by 6) with various car pieces placed on the board. These cars can be either two or three units in length and their movement respects their orientation (i.e. a car facing North can move North or South) as would be expected. There is one special car which is colored red and is aligned with an exit space on the border of the grid. The object of the game is to move the other vehicles such that the red car can escape unobstructed.

The Rush Hour game is itself classed with a group of games which are known as PSPACE-complete. PSPACE-completeness is defined as a problem that can be solved by a nondeterministic Turing machine in polynomial space (Servais 2005). This requires the user to cognitively determine a potential move sequence to test or to

determine some way to determine whether they are “getting closer” to a solution (Flake 2001). This clearly places Rush Hour outside of the realm of puzzles which are usually NP-complete, and in the realm of games.

The PSPACE-completeness of the game makes it extremely hard to find a solution to via a brute-force algorithm so finding a solution must be left to alternate methods. If one could articulate this “closer to” a solution algorithm then a genetic algorithm could be used to determine an acceptable, although not necessarily optimal, solution.

On Rush Hour Puzzles

Rush hour boards have the red car aligned with the escape route so that it can leave the board after the other cars have been removed such that the red cars path is unobstructed. Although this project at present is concerned mostly with solving trivial or near-trivial boards it is worth mentioning how harder level boards are created since at least in some cases this can be analogous to the process for solving the boards.

PuzzleBeast is one web site which generates rush hour boards which are of some difficulty. It first generates several random boards then picks out the hardest of them and mutates it randomly. After this completes if the new board is higher ranked than the original then the original is replaced and the process continues until the specified number of cycles or the specified difficulty is reached. (Stephens 2003).

As you can see this is fairly analogous to how a genetic algorithm might solve a puzzle, generating a move list then mutating it until the desired result is achieved.

Implementation

A Piece

The piece object defines the characteristics of a piece on the board. It includes a facing direction, a color, and size. A property list is also defined on the piece which contains the current coordinates of the piece. Strictly speaking the direction of the piece isn’t necessary but adds an extra level of abstraction and may be helpful if a GUI for the program is ever developed. The color provides a way to reference the piece easily. It is very useful in comparing pieces since it is the only unique value a piece has.

The Board

The board isn’t an object as one might think it should be since maintaining it would be difficult and redundant since a current move list along with the pieces present is already maintained. Displaying the board is simply a matter of scanning through the piece list to determine which pieces are in each position and displaying it, as is shown in figure 1.

```
[2]>(display-board)
( * * * * * )
( * * S S S * )
( * G G * * * )
( * * R * * * )
( * * R * * * )
( * * * * * )
```

Figure 1

An individual in the sense of the rush hour game is a move sequence which is a candidate for a solution. This move sequence is of the form:

$$(((a_1, b_1) d_1 q_1) \dots ((a_n, b_n) d_n q_n))$$

Where we define each a,b to be the coordinate of the “front” of the piece, d is a direction, N S W or E and q is a distance to travel.

This representation is useful for output to the user as a final set of moves which solve the game. In processing this is only partially useful since it contains all data about a move but is limited in that it doesn’t provide direct access to the piece object to change it. It is for this reason among others we will show shortly that a parallel list is maintained of the piece objects being moved.

These lists are of length n due to the variable nature of the length of these move sequences. This variable nature begins from creation and remains as such until the program ends. The length of an individual at initial creation is random from one to a predefined upper limit. I have found that three seems to be a reasonable upper limit for relatively simple boards, but more testing needs to be done in this area.

Through the use of the crossover method the lists are allowed to grow and shrink in length beyond this predefined limit.

I was unable to find another example of a board game which used a variable length individual; in fact the only problem I could find close to this one solved with genetic algorithms is the n-piece problem which is known to be solvable in 13 moves (Ahmadi 2004). There was one game I found using a variable length individual, the Minority Game developed by the University of Fribourg which is based on strategy involving cooperative/competitive players but was not directly applicable to this problem (Wei-Song 2004).

The individual object itself contains not only the move sequence in the list mentioned above, but also a parallel list which contains the objects which are to be moved, this ensures that we don’t move the same piece twice in a row making an artificially longer move list than is necessary.

This seems to be the best way to get move lists of multiple lengths which are comparable.

Algorithmic Components

There are two main algorithmic components to the genetic algorithm besides the fitness metric, the crossover and mutation methods. A mutation does not modify the length of the move sequence; it only flips the direction or increases/decreases the movement distance by an appropriate amount. This can though result in invalid move sequences, in which case the probability of mutation is applied again and if necessary mutation is attempted again. Following is a basic example of how the mutation operator works. Consider the move ((4 3) N 1) and assume this is the red car facing North. Applying the mutation operator could yield any of the following: ((4 3) S 1), ((4 3) N 2), or ((4 3) N 3).

Crossover isn’t far removed from the standard definition. First the pieces are chosen, generally two lists of high fitness from a representative sample of the population. The algorithm chooses zero or one place off from the center of the both lists and chooses that place for the crossover to occur. This allows the lists to change in length between negative two and two places. This effectively limits the growth to an acceptable margin so lists do not end up extremely long in the first few generations.

As with mutation the generated list is checked for validity by attempting to take the move. Crossover has a far higher likelihood of not yielding a valid move list than mutation so the fail rate here is much higher.

Any problems with the algorithm with relation to the crossover metric (which do exist) do not seem to be related to an inherent flaw in the variable length individual paradigm but instead a problem with the metric itself which requires further refinement as was shown in Cavill’s look at the onemax

problem using variable length individuals which claims that crossover is the key part in keeping the algorithm functioning (2006).

Figure 2 gives an example of how the crossover function works.

```

Selecting two lists to cross over...
21 (((4 3) S 1) ((2 3) W 1)) 98
35 (((3 2) W 1) ((4 3) S 1) ((3 1) E 2)) 98
Result:
0 (((3 2) W 1) ((4 3) S 1)) 99
Valid? Yes.

```

Figure 2

Fitness Metric

The fitness metric allows for any number of cars to be placed on the board and for any number of moves to be in the move list. It is computed by actually “taking” the moves and determining the fitness of the final board state. This fitness is defined by how “easy” it is for the red car to escape from the board, which is consistent with how the game is played in reality.

We define how “easy” it is by how many cars are in the way of the red car, and recursively how many cars are in the path of those cars movement so that they can move sufficiently for the red car to move. Naturally this algorithm takes into account which cars have already been accounted for to avoid infinite loops.

The fitness metric has a value of 100 if the game has been solved. For each of the recursive cars in the way it subtracts one point from this. 100 may not be the best maximal value, but the true max is dependant on how many cars are present and their sizes so we use 100 because it is high enough so that the value will never be negative.

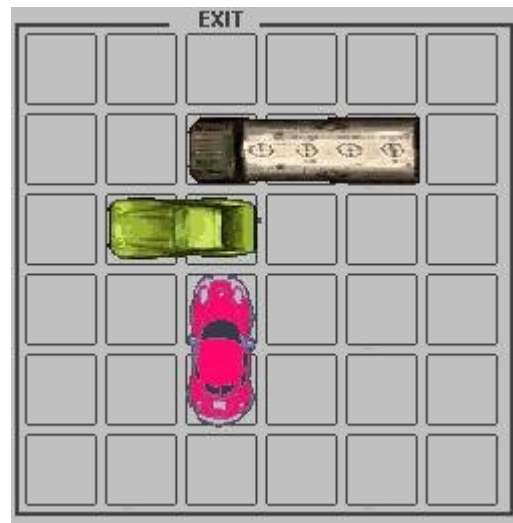
An Illustrative Example

We will now consider an example which shows the execution of the program

from start to finish with a trivial example to which the answer is immediately visible to any human looking at the board (and indeed if the initial population is larger than 5 there is a good chance the computer will generate a solution before the algorithm does any work).

Note that much of the program is still incomplete so some of this work was done by hand in a manner in which the algorithm would have done it had it been completed.

Consider the board given as follows:



This board contains three cars, whose facing positions are (3 2), (2 3), and (4 3). In the generation of a small population the following is generated:

Generation 0:

- 1 (((2 3) W 1) ((3 2) E 1)) 98
- 2 (((2 3) W 2) ((3 2) W 1) ((2 1) E 2)) 99
- 3 (((3 2) E 1)) 98
- 4 (((3 2) W 1)) 99

The individuals with the top two fitnesses are crossed over, so crossovers occur between lists 2 and 4, and by random selection list 1 is mutated in generation 1:

Generation 1:

- 1 (((2 3) E 1) ((3 2) E 1)) 99
- 2 (((2 3) W 2) ((3 2) W 1)) 99

- 3 (((3 2) E 1)) 98
- 4 (((3 2) W 1)) 99

Here we see the crossover did little to help since there was only one real combination of 2 and 4 which made sense, and that was the one which was taken (all others involved moving the piece at (3 2) west more than 1 which isn't valid). Piece 1 has been mutated and now has a higher fitness than before. Now we look at Generation 2 in which there are no crossovers and piece 1 is mutated again after piece 2 fails to be mutated.

- ((2 3) W 2) -> ((2 3) E 2) failed!
 Generation 2:
 1 (((2 3) E 1) ((3 2) W 1)) 100
 2 (((2 3) W 2) ((3 2) W 1)) 99
 3 (((3 2) E 1)) 98
 4 (((3 2) W 1)) 99

Here you can see that piece 2 failed to be mutated and was passed over and piece 1 was mutated randomly instead. This mutation leads to a solution being found. You will note that this mutation says nothing about moving the red car forward the required three spaces since it is assumed that this is obvious. This was a design decision to reduce the length of move sequences and the complexity of finding a solution.

The Algorithm in Use

The example given above is trivial to a human, and it is a bit too trivial for a machine as well. Somewhere around 90% of the time the solution to this problem is found in the initial generation of the population so for actual tests a slightly more complex example was used. This more difficult problem has the same board configuration as the above with another car occupying spaces (1,2) through (1,4) as shown in the below generated output:

```
( * O O O * * )
( * * S S S * )
( * G G * * * )
( * * R * * * )
( * * R * * * )
( * * * * * * )
```

A solution to this problem is always found by the algorithm, and always within the first three generations. I have included in appendix A a sample run of the program and as you can see even though this example only contains two generations there is an increase in average fitness of the population and multiple solutions are found. This shows that the crossovers and mutations are actually yielding a widespread increase in fitness and not just a localized anomaly resulting in a solution.

Reflections

Evaluation

Currently the state of the algorithm is not such that it works to generate a solution for all puzzles, or even particularly easy ones for that matter, but the project is not a failure by any means. In generating move lists and applying the fitness metric we do seem to get a fairly accurate representation of "how good" the board is which takes into account how close the red car really is to being free. The problem currently seems to lie in performing proper crossover and mutations appropriately. There is about a 60% success rate for mutations and around 25% for crossovers. Far too many of these potential lists are invalid so action must be taken further in this area. One potential which shows possible promise is generating a list of all possible mutations and crossover results from the given lists and choosing randomly from those. Before this kind of thing can be done though the system as a whole needs a lot of refinement in usability along with much further testing as

very little has been done in this area and many bugs may still remain.

Future Work

In addition to the possible changes already mentioned, I would like very much to implement a user interface for the program in java to illustrate the moves being made and the genetic algorithm running. I believe this would not only be a neat application but it would be a great learning tool about how genetic algorithms work. I am also very impressed by Henrik Theiling's postscript move graphic generator which would be interesting to port to Java. There was not time to explore the possibility in this course but I would like to attempt to abstract the idea of a move list further so that it is linked on a low level to underlying piece objects. This would allow drastic simplification of the code since much of the code has to do with refactoring and updating the move lists. A low level connection would do this refactoring automatically. Also, the board is often reset and new pieces are created before a move is taken so that no issues arise from bad data – I think there should be a more elegant “undo” type function but with the architecture as it currently is this is very difficult to implement.

Alternate Methods

The single alternate method I found that has been implemented is that of a brute force breadth first search of the problem space. As the author states this is a complex procedure, but since the board is small it seems to work fairly well (Theiling 2007). This is of course an acceptable solution and is guaranteed to find a solution within a certain amount of time. In general though with an n by n Rush Hour game this would not be a very likely candidate to solve the game as the complexity would grow very quickly. With refinement I believe fitness metric can be

defined in a general sense that can provide solutions to the game for any size board.

References

- Ahmadi, Seyed, (2004). *An investigation on the performance of a genetic algorithm to solve the 15-puzzle*.
<http://www.alia.ir/school_files/GA_15puzzle/Genetic%20Algorithm%20&%2015-Puzzle.doc>.
- Cavill, Rachel and Smith, Stephen, and Tyrrell, Andy (2006). *Variable Length Genetic Algorithms with Multiple Chromosomes on a Variant of the Onemax Problem*. In Proceedings of the 8th annual conference on Genetic and evolutionary computation, pages 1405-1406.
<http://portal.acm.org/ft_gateway.cfm?id=1144217&type=pdf>.
- Flake, Gary and Baum, Eric. (2001). *Rush Hour is PSPACE-complete, or “Why you should generously tip parking lot attendants.”* In Theoretical Computer Science, vol 270 (2002), p895-911.
- Puzzles.com. *Rush Hour Introduction*,
<<http://www.puzzles.com/products/rushhour.htm>>.
- Servais, Frederic, (2005). *Finding hard initial configurations of Rush Hour with Binary Decision Diagrams.*
- Stephens, James, (2003). *“How the PuzzleBeast Puzzles are Created.”* Accessed April 17, 2007.
<<http://www.puzzlebeast.com/about/index.html>>.
- Theiling, Henrik, (2007). *Rush Hour Solver*.
<<http://www.theiling.de/projects/rushhour.html>>.
- Wei-Song Yang, Bing-Hong Wang, Yi-Lin Wu and Yan-Bo Xie (2004). *Searching good strategies in evolutionary minority game using variable length genetic algorithm*. In “Physica A: Statistical Mechanics and its Applications” Volume 339, Issues 3-4, 15 August 2004, Pages 583-590.

APPENDIX A: A Sample Run

Break 1 [29]> (ga-demo)

```

1 ((1 2) E 2) ((3 2) E 2)) 99
2 (((3 2) E 1) ((2 3) W 2) ((3 3) E 1)) 98
3 (((3 2) E 2) ((2 3) W 1) ((1 2) W 1)) 98
4 (((1 2) W 1) ((2 3) W 2) ((3 2) E 1)) 97
5 (((3 2) W 1) ((1 2) W 1) ((2 3) E 1)) 99
6 (((3 2) W 1) ((2 3) W 2) ((4 3) S 1)) 98
7 (((3 2) W 1) ((4 3) N 1)) 98
8 (((1 2) E 1) ((4 3) S 1)) 97
9 (((1 2) E 1) ((3 2) E 1)) 97
10 (((1 2) W 1)) 97
11 (((3 2) E 3) ((1 2) W 1) ((3 5) W 1)) 98
12 (((2 3) E 1)) 98
13 (((1 2) E 1) ((3 2) E 2)) 98
14 (((1 2) E 2) ((2 3) E 1) ((1 4) W 3)) 98
15 (((2 3) W 2)) 97
16 (((1 2) E 2) ((3 2) E 1)) 98
17 (((2 3) W 2) ((3 2) E 2)) 98
18 (((4 3) S 1) ((2 3) W 2) ((1 2) W 1)) 97
19 (((3 2) W 1) ((2 3) E 1) ((1 2) W 1)) 99
20 (((2 3) W 1)) 97
21 (((3 2) E 3)) 98
22 (((1 2) E 1)) 97
23 (((2 3) E 1) ((3 2) W 1)) 99
24 (((3 2) E 2) ((2 3) E 1)) 99
25 (((3 2) E 2) ((1 2) E 2)) 99
26 (((3 2) E 1) ((2 3) W 2)) 97
27 (((2 3) E 1) ((3 2) W 1) ((2 4) W 2)) 98
28 (((2 3) W 2) ((3 2) E 2) ((1 2) W 1)) 98
29 (((1 2) E 1)) 97
30 (((2 3) E 1)) 98
31 (((1 2) W 1) ((3 2) W 1) ((2 3) W 1)) 98
32 (((3 2) W 1) ((2 3) E 1)) 99
33 (((3 2) E 2) ((1 2) E 1)) 98
34 (((3 2) E 1) ((4 3) S 1)) 97
35 (((4 3) S 1)) 97
36 (((3 2) E 1) ((1 2) W 1)) 97
37 (((1 2) W 1) ((2 3) W 1) ((4 3) S 1)) 97
38 (((2 3) W 1) ((1 2) E 1)) 97
39 (((2 3) W 1) ((3 2) E 2)) 98
40 (((2 3) E 1) ((4 3) S 1) ((2 4) W 1)) 97
41 (((3 2) W 1) ((1 2) W 1)) 98
42 (((1 2) E 1)) 97
43 (((2 3) E 1) ((1 2) E 2)) 99
44 (((1 2) W 1)) 97
45 (((3 2) E 2) ((4 3) S 1) ((3 4) W 1)) 97
46 (((3 2) W 1)) 98
47 (((1 2) E 1) ((3 2) E 3) ((1 3) E 1)) 99
48 (((1 2) W 1)) 97
49 (((3 2) E 1) ((2 3) E 1) ((3 3) E 2)) 99
50 (((2 3) W 2)) 97
51 (((4 3) S 1) ((1 2) E 1) ((2 3) W 1)) 97
52 (((3 2) E 2) ((1 2) E 2)) 99
53 (((3 2) E 3) ((2 3) W 2)) 98

```

```

54 (((3 2) W 1) ((2 3) W 2) ((4 3) S 1)) 98
55 (((3 2) E 1) ((2 3) W 2) ((3 3) E 2)) 98
56 (((4 3) S 1) ((2 3) W 1) ((3 2) E 1)) 97
57 (((3 2) E 3) ((1 2) E 2)) 99
58 (((2 3) W 2)) 97
59 (((1 2) W 1)) 97
60 (((1 2) W 1) ((3 2) E 3)) 98
61 (((2 3) W 1) ((1 2) E 2)) 98
62 (((3 2) E 1) ((2 3) W 2) ((1 2) E 2)) 98
63 (((3 2) E 1) ((1 2) E 2) ((3 3) W 1)) 98
64 (((3 2) E 3) ((4 3) S 1)) 98
65 (((2 3) E 1)) 98
66 (((1 2) E 1) ((3 2) E 2)) 98
67 (((1 2) W 1) ((2 3) W 2) ((4 3) S 1)) 97
68 (((2 3) W 2) ((4 3) S 1) ((2 1) E 2)) 97
69 (((1 2) W 1) ((2 3) W 1) ((4 3) S 1)) 97
70 (((2 3) W 2) ((1 2) E 1)) 97
71 (((2 3) W 2) ((4 3) S 1) ((2 1) E 3)) 98
72 (((4 3) S 1) ((1 2) E 1)) 97
73 (((2 3) E 1) ((1 2) E 1)) 98
74 (((4 3) S 1) ((3 2) E 2) ((2 3) W 2)) 98
75 (((2 3) W 1)) 97
76 (((3 2) E 3) ((1 2) E 1)) 98
77 (((3 2) E 2) ((2 3) W 1)) 98
78 (((1 2) E 1) ((4 3) S 1)) 97
79 (((3 2) E 2) ((4 3) N 1) ((1 2) E 1)) 98
80 (((2 3) W 2) ((3 2) W 1) ((1 2) E 2)) 99
81 (((2 3) E 1) ((1 2) W 1) ((3 2) W 1)) 99
82 (((3 2) W 1) ((1 2) E 2)) 99
83 (((1 2) E 1)) 97
84 (((3 2) W 1) ((2 3) W 2)) 98
85 (((1 2) E 2) ((2 3) W 1)) 98
86 (((2 3) W 1)) 97
87 (((3 2) E 1) ((2 3) W 2) ((4 3) S 1)) 97
88 (((3 2) W 1) ((2 3) E 1) ((4 3) N 1)) 99
89 (((3 2) E 1) ((2 3) E 1) ((1 2) W 1)) 98
90 (((3 2) E 2) ((2 3) W 1) ((1 2) W 1)) 98
91 (((1 2) E 2) ((3 2) E 2) ((4 3) N 1)) 99
92 (((1 2) E 1)) 97
93 (((2 3) E 1) ((1 2) E 1)) 98
94 (((1 2) E 2) ((3 2) E 2)) 99
95 (((4 3) S 1)) 97
96 (((2 3) E 1) ((1 2) E 2)) 99
97 (((3 2) E 3) ((1 2) E 1) ((2 3) E 1)) 99
98 (((1 2) E 1) ((2 3) W 1) ((1 3) E 1)) 98
99 (((3 2) E 1)) 97
100 (((2 3) E 1)) 98
average fitness = 97.82

```

average fitness of population 1 = 98.69

Solution found!

```

1 (((3 2) E 3) ((1 3) E 2)) 99
2 (((3 2) E 1) ((4 3) N 1)) 98
3 (((1 2) W 2) ((3 2) E 2)) 99
4 (((3 2) W 1) ((2 3) E 1)) 99
5 (((3 2) E 1) ((2 3) E 1) ((3 5) E 2)) 99

```

6 ((3 2) E 3) ((1 2) W 2)) 99
7 ((3 2) E 3) ((1 3) E 2)) 99
8 ((3 2) W 1) ((1 4) W 1) ((2 4) E 1)) 99
9 (((1 2) E 2) ((3 2) W 2)) 99
10 (((3 2) W 3) ((1 2) E 1) ((2 3) E 1)) 99
11 (((2 3) W 2) ((3 2) W 1) ((1 2) E 2)) 99
12 (((2 3) W 2) ((3 2) W 1) ((1 2) E 2)) 99
13 (((2 3) W 2) ((3 2) E 1) ((1 2) E 2)) 98
14 (((2 3) E 1) ((3 2) W 1)) 99
15 (((2 3) E 1) ((1 2) E 1) ((3 2) W 1)) 99
16 (((2 3) W 1) ((3 2) W 1)) 98
17 (((1 2) E 1) ((3 2) E 3) ((1 3) E 1)) 99
18 (((3 2) W 2) ((1 2) E 2)) 99
19 (((1 2) E 2) ((3 3) E 2) ((4 3) N 1)) 99
20 (((3 2) E 3) ((1 3) E 2)) 99
21 (((2 3) W 2) ((3 2) E 1) ((1 2) W 1)) 97
22 (((1 2) E 2) ((3 3) E 2) ((4 3) N 1)) 99
23 (((3 2) E 3) ((1 2) W 1) ((3 5) W 1)) 98
24 (((3 2) E 1) ((2 3) E 1) ((3 5) E 2)) 99
25 (((2 3) W 2) ((3 2) W 1) ((1 2) E 2)) 99
26 (((3 2) W 1) ((2 3) E 1) ((4 3) N 1)) 99
27 (((3 2) W 1) ((2 3) E 1) ((4 3) N 1)) 99
28 (((3 2) W 2) ((1 2) E 2)) 99
29 (((1 2) E 2) ((3 3) E 2) ((4 3) N 1)) 99
30 (((3 2) W 1) ((2 3) E 1) ((3 5) E 2)) 99
31 (((3 2) W 1) ((1 2) E 1) ((2 4) E 1)) 99
32 (((3 2) W 2) ((2 3) E 1)) 99
33 (((3 2) W 1) ((2 3) E 1) ((1 2) W 1)) 99
34 (((2 3) W 1) ((3 2) E 2)) 98
35 (((3 2) W 1) ((1 4) W 1) ((2 3) E 2)) 99
36 (((1 2) E 2) ((3 2) W 2)) 99
37 (((2 3) W 1) ((1 2) E 2)) 98
38 (((2 3) W 1) ((1 2) E 2)) 98
39 (((2 3) W 2) ((3 2) W 1) ((1 2) E 2)) 99
40 (((2 3) E 1) ((3 2) W 1) ((1 2) E 2)) 100
41 (((3 2) E 2) ((2 3) E 1)) 99
42 (((3 2) E 2) ((3 4) E 2) ((1 2) E 2)) 99
43 (((3 2) E 2) ((2 3) E 1)) 99
44 (((3 2) E 1) ((2 3) E 1) ((4 3) N 1) ((1 2) E 1)) 99
45 (((1 2) E 1) ((3 2) E 2)) 98
46 (((3 2) E 3) ((1 3) E 2)) 99
47 (((1 2) E 1) ((2 4) E 1)) 98
48 (((3 2) E 3) ((1 2) E 1) ((4 3) N 1)) 98
49 (((2 3) E 1) ((1 2) W 1) ((3 3) E 2) ((4 3) N 1)) 99
50 (((3 2) E 3) ((1 2) E 1) ((1 3) E 2)) 99
51 (((1 2) E 1) ((3 2) E 2)) 98
52 (((1 2) W 1) ((3 2) W 1) ((2 3) W 1)) 98
53 (((4 3) S 1) ((3 2) E 2) ((3 4) E 2) ((1 2) E 2)) 99
54 (((3 2) W 1) ((2 3) E 1) ((1 2) E 2)) 100
55 (((3 2) E 3) ((1 3) E 2) ((3 2) E 2)) 99
56 (((3 2) W 1) ((2 3) E 1)) 99
57 (((1 2) E 2)) 98
58 (((3 2) E 1) ((1 2) W 1) ((3 5) W 1)) 98
59 (((2 3) E 1) ((1 2) E 2)) 99
60 (((2 3) E 1) ((1 2) E 2)) 99
61 (((1 3) E 1)) 98
62 (((3 2) E 2) ((1 2) E 2)) 99
63 (((2 3) E 1) ((2 4) E 1) ((1 2) E 1)) 98
64 (((3 2) W 1) ((1 2) E 2)) 99
65 (((2 3) E 1) ((3 2) E 3) ((1 2) E 2)) 100
66 (((2 3) E 1) ((1 2) W 1) ((2 3) E 1)) 98
67 (((3 2) E 3) ((1 3) E 1)) 99
68 (((2 3) E 1) ((1 2) E 2)) 99
69 (((3 2) W 1) ((2 3) E 1)) 99
70 (((3 2) E 1) ((2 3) W 2) ((3 2) W 1)) 98
71 (((2 3) W 2) ((1 3) E 2)) 98
72 (((3 2) W 1) ((1 2) E 2)) 99
73 (((2 3) E 1)) 98
74 (((3 2) E 2) ((2 3) E 1)) 99
75 (((3 2) W 1) ((2 3) E 1)) 99
76 (((3 2) E 3)) 98
77 (((2 3) E 1) ((3 2) W 1)) 99
78 (((3 2) E 3) ((1 2) E 1) ((2 3) E 1)) 99
79 (((3 2) W 1) ((2 3) E 1) ((4 3) N 1)) 99
80 (((2 3) E 1)) 98
81 (((2 3) E 1)) 98
82 (((2 3) W 2) ((3 2) W 1) ((1 2) E 2) ((3 2) E 2)) 99
83 (((3 2) E 1) ((2 3) E 1) ((1 2) W 1)) 98
84 (((3 2) E 1) ((2 3) E 1) ((1 2) E 2)) 99
85 (((2 3) E 1)) 98
86 (((1 2) E 1) ((3 2) E 3) ((1 3) E 1)) 99
87 (((2 3) E 1) ((3 2) E 3) ((1 2) E 2)) 100
88 (((3 2) E 3) ((3 5) E 2)) 98
89 (((1 2) E 2) ((3 3) E 2) ((4 3) N 1)) 99
90 (((1 2) E 2) ((3 2) E 2)) 99
91 (((1 2) E 2)) 98
92 (((1 2) W 1) ((3 2) E 3)) 98
93 (((3 2) W 1) ((2 3) E 1)) 99
94 (((1 2) E 1)) 97
95 (((3 2) E 1) ((3 3) E 2) ((4 3) N 1)) 98
96 (((3 2) W 1) ((1 4) W 1) ((3 2) E 3) ((1 3) E 2)) 99
97 (((3 2) W 1)) 98
98 (((1 2) W 1) ((3 2) E 3)) 98
99 (((3 2) W 1) ((1 3) E 2)) 99
100 (((2 3) E 1) ((1 2) E 2) ((1 4) W 1) ((2 4) E 1)) 98
average fitness = 98.69