Agent Motion Planning with GAs Enhanced by Memory Models

Martijn C.J. Bot
Vrije Universiteit
Faculty of Science
De Boelelaan 1081
1081 HV Amsterdam
+31 20-4447790
mbot@cs.vu.nl

Neil Urquhart
School Of Computing
Napier University
219 Colinton Road
Edinburgh
+44 0131 455 4432
n.urquhart@napier.ac.uk

Ken Chisholm
School Of Computing
Napier University
219 Colinton Road
Edinburgh
+44 0131 455 4216
k.chisholm@dcs.napier.ac.uk

Abstract

The Tartarus problem may be considered a benchmark problem in the field of robotics. A robotic agent is required to move a number of blocks to the edge of an environment. The location of the blocks and position of the robot is unknown initially. The authors present a framework that allows the agent to learn about its environment and plan ahead using a GA to solve the problem. The authors prove that the GA based method provides the best published result on the Tartarus problem. An exhaustive search is used within the framework as a comparison, this provides a higher score still. This paper presents the two best Tartarus results yet published.

1 Introduction

The Tartarus problem may be considered a benchmark problem in the area of non-Markovian agent motion planning. The agent is placed within an environment, with no prior knowledge of the environment and limited means by which to gather information on the environment (see Figure 1). The task to be undertaken involves moving blocks placed at random positions within the environment to the outer edges of the environment. There is only a finite amount of energy available to the agent, thus limiting the number of moves that can be made.

The challenge is therefore to devise a solution to the problem that can gather information on the environment and solve the problem at the same time. We enhance a genetic algorithm with a long term memory model for incorporating information that was found in previous steps. We will show that our approach outperforms leading algorithms on this problem.

2 Problem Description

2.1 An overview of the Tartarus Problem

Within the Tartarus problem, a robotic agent is placed in an environment that consists of a 6x6 square grid (akin to a checkers board, see Figure 1). The agent occupies one square, while also on the board are 6 blocks each of which occupy one square. The object of the exercise is for the agent to push the blocks to the edge of the board, scoring 1 point for each block moved to an edge or two points for each block pushed into a corner. The maximum score then is 10. Only one block may be pushed at one time. Each time the agent moves forward, rotates or pushes a block forward it uses one unit of energy.

The agent's sensors can only detect the contents of the 8 squares directly surrounding the agent's position.

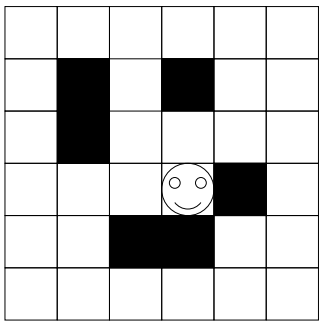
The objective of the agent is to maximize the average score over 100 randomly generated boards.

2.2 Board Initialisation

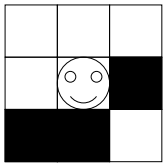
The board is initialised by placing all 6 blocks in random squares, and then placing the agent in a random square facing a random direction. Neither the blocks nor the agent will be initially placed adjacent to the edge. A configuration of 4 blocks placed together cannot be moved by the agent (because it can only move one block at a time). Therefore the board is never initialised with four blocks arranged in a square.

2.3 Sensors

The agents' sensors are capable of sensing the contents of the eight squares adjacent to the agents' current position. The sensors can detect whether each square is empty, contains a block, or constitutes part of an edge. The agent cannot sense its orientation or its



Tartarus board



View of the agent

Figure 1: Example Tartarus board

position on the grid.

2.4 Energy Levels

Within the Tartarus problem, there is no time limit, but the agent has only a limited amount of energy. The agent has an initial energy level of 80 units, each move forward or rotation costs the agent 1 unit of energy. Once all the energy has been used, the agent can no longer move and the attempt to solve the problem ceases.

3 Previous Work

Previous techniques applied to the Tartarus Problem include genetic algorithms, neural networks, finite state machines and genetic programming.

Teller[Teller, 1994] used genetic programming with indexed memory to achieve an average score of 4.5.
In [Balakrishnan and Honavar, 1996], neural networks have been utilised with a maximum score of only 4.5.
The highest score achieved so far has been by [Ashlock and Joenks, 1998], whose GP-based algorithm averaged a score of 8.2. Earlier GP based work by Ashlock and McRoberts[Ashlock and McRoberts, 1997] achieved a score of 8.15.
The most recent research has been undertaken by [Ashlock and Freeman, 2000] who utilised a GA to evolve a finite state machine. The finite state machine interprets the results of the sensors and at each change in state can issue a command to the agent. The average final score achieved by Ashlock and Freeman was 7.11.
All of the research outlined above utilised some form of internal state or memory within the agent to allow the agent to learn about the environment. Solutions that haven't utilised some form of internal state within the agent have not achieved an average score of greater than 2.
By examining previous research, it may be concluded that the agent needs to be equipped with the ability to hold an internal state within some form of 'memory'.

4 Formulating the Solution

4.1 Human Attempts to Solve the Problem

The authors initially carried out an informal experiment using human agents (i.e. a human controlling the agent by manually issuing commands). One agent was asked to solve the problem while only being allowed to view the inputs from the eight sensors. The second agent was allowed paper and pencil to draw a map of the environment as they explored it. Each agent attempted to solve the problem 10 times. The experiment revealed that even with the processing power of a human brain, the efficiency of the solutions increased dramatically when the agent was allowed to collate the information gathered through its sensors in the form of a map. Without a map, the human agent averaged a score of 7.2, but with the energy levels reduced to 0 in every case. By allowing the human agent to build a map, the average score rises to 9.1 with more energy left.

The authors' perceived reason for the human agents improved performance when drawing a map, was the ability to use the information in the map to pre-plan sequences of moves before issuing commands to the agent. Cognitive psychologists have estimated human short-term memory only to capable of containing 7±2 'chunks' of information. The human agent working without the map may have been unable to recall the previous values of the sensors, and build a 'memory map' of the area.

4.2 A Description of the Chosen Solution

4.2.1 Overview

The information contained in the agents' sensors may be considered equivalent to the human short-term memory. They are both transient and of low capacity. The informal experiment conducted in section 4.1 and previous research reviewed in section 2 both suggested a requirement for the agent to be given some form of 'long-term' memory. This long-term memory will contain information about the environment, gathered from the short-term memory (sensors) as the agent is moved.

Having established the requirement for short and long-term memories, we now require to process the information stored in the long-term memory to allow the agent to carry out its task. The processor function will be carried out by a Genetic Algorithm (GA). The GA will evolve command sequences consisting of Forward, Left or Right moves to allow the agent to push the blocks discovered so far to the edge of the board. After a set number of evaluations the GA will be halted and the command sequence contained within the best chromosome will be executed by the agent. As soon as the agent discovers a new feature within the landscape, it stops executing the command sequence and the GA is restarted to evolve a new command sequence based on the updated information now contained within the long-term memory.

4.2.2 The Long and Short Term Memories

As has already been described, the short-term memory is the buffer for the eight sensors. Each time the agent moves, the information contained within the sensors will be replaced by values relating to the agents' new position.

The long-term memory is a 11x11 grid. The long-term memory must be bigger than the board, because the agent could initially be placed almost anywhere on the board. The long-term memory is large enough to allow the data sensed from the agents initial position to be placed in the centre and then the map to be built out from this point.

Each of the 121 locations within long-term memory can hold one of five values;

1. Block: This square definitely contains a block
2. Empty: This square is definitely empty
3. Edge: This square is on the edge
4. Probably Empty: This square has not been explored yet, but it is assumed that it is empty
5. Something: The agent has tried to push a block into this square, but couldn't as it is either occupied by another block or it forms part of the edge

As the agent progresses in solving the Tartarus problem, the map contained within long-term memory is built-up. This map is used by the GA fitness function (see section 4.2.4) when evaluating command sequences.

4.2.3 Wall Deduction Heuristics

Because the characteristics of the environment, its size, shape and the number of blocks contained within it are known, the agent may be enhanced with a number of simple heuristics. These heuristics assist the agent when interpreting data contained in short-term memory and then enhancing the map contained in long-term memory.

The deduction of the walls may be assisted by a number of simple rules. If one piece of wall is found, then the entire wall can be deduced. If a wall is found then we can establish the position of the wall running parallel to it.

When a block is discovered at location x, we can deduce that the walls can be no further than 5 squares in any direction, thus the 11x11 grid can be reduced in size. This heuristic has been named 'Smart Wall Deduction' (SWD) by the authors. Further analysis has resulted in the enhancement of SWD not only to use blocks but assume that a wall is never more than 5 squares from any explored square. The modified heuristic has been named Even-Smarter Wall Deduction (ESWD).

Once all 6 blocks have been found, any remaining memory locations marked as 'Something' must hold walls, and vice-versa once the entire wall has been discovered any remaining 'Something's must be blocks. This has been named the '6 block heuristic'.

4.2.4 The Genetic Algorithm

The genetic algorithm is used within the agent to evolve command sequences that may be carried out by the agent. Each chromosome consists of a list of commands in the form:

MMLMMMRM....

The commands are referred to as command sequences, and are interpreted thus:

M - Move forward 1 square

L - Rotate left

R - Rotate right

The length of the chromosomes was altered during the experiments carried out. Initially the chromosome length was set to 80, this being the maximum number of commands that may be carried before the agent runs out of energy.

Table 1: Chromosome Initialisation

Previous Genes	Possible values for current gene
L L	M
R R	M
L	L or M
R	R or M

Table 2: Initial fitness function rewards

Criterion	Reward
A block has just been pushed	3
A previously unknown square explored	2
A block has just been pushed into a wall	7

The GA is initialised with semi-random strings of genes. The authors identified a number of patterns that may occur within the chromosome that would result in the agent wasting energy (e.g. by rotating around in a circle). A simple initialisation scheme has been set up that restricts the choice of gene based on the previous genes (see Table 1). This scheme ensures that the initial population is free from wasteful patterns. Note that no repair occurs after mutation or crossover.

The recombination operator used is standard two-point crossover based on two parents creating one child. The mutation operator selects an individual with probability 0.1, a gene within that individual is then selected for mutation with the probability 0.02. The mutation consists of altering the value of the selected gene to M, L or R randomly.

A steady-state population of 500 is maintained. Selection and replacement of individuals will be facilitated by using a tournament selection operator. A tournament size of 7 was found to give reasonable results.

The fitness function evaluates the chromosome by simulating the execution of the command sequence using a copy of the map contained within long-term memory. The fitness function evaluates each command and rewards it based in the probable position of the agent after the command has been executed criterion as shown in Table 2.

After completing the route the final score (blocks against a wall + blocks in corners) is added to the fitness weighted by a factor of 100. Because the Tartarus problem has to be completed within a finite number of moves, the fitness function only examines those commands that could be executed given the remaining energy level.

5 Experiments

5.1 Experimental setup

Because of the deterministic nature of the GA used within the agent and the wide variety of starting configurations that exist for the Tartarus problem each experiment was carried out 100 times using randomly generated environments.

The software was initially implemented using ANSI standard C++, running on Redhat Linux. To allow for greater flexibility the software was subsequently re-written in Java. Later versions of the software were implemented across a 128 CPU parallel processing network.

5.2 The Initial Version

The initial version used a population size of 100 individuals, a mutation rate of 0.10 and a crossover rate of 0.10. Initially the GA was allowed to run until 1000 tournaments had been completed. Unless it is mentioned, it can be assumed that these basic parameters were used. The initial version incorporated no heuristics, and evaluated as many commands as the current energy level would allow. The average score achieved over 100 boards was 4.38. The distribution of scores was varied, one board scoring 8, four scoring 7 and the remaining 95% achieved scores of 6 or less.

Analysis of boards where the agent achieved a low score showed that a frequent problem was the agent pushes a block while unknown to the agent there's another block or a wall behind this block. In this case, the agent knows there's something behind this block, but it does not know whether this is a block or a piece of wall. Noting this in the long-term memory map would be useful, because the agent would be less likely to try and push this block. In the fitness evaluation (see Section 4.2.4), no points are gained for trying to push a block while knowing this is not possible. In order to be able to note down such information in long-term memory, the data type 'Something' (see Section 4.2.2) was added, allowing the average score over 100 boards to rise to 6.09.

With the addition of the initial SWD heuristic (as described in section 4.2.3) the average score was further increased to 6.21.

It was felt that the GA was running for too brief a period, and because there is no time constraint on the Tartarus problem, the authors allowed the GA to run

Table 3: Average scores over 100 boards using advanced edge detection, the six block heuristic and forcing a restart after hitting a known wall

ESWD	6-block	Restart After Wall	score
0	0	1	7.52
0	0	0	7.39
0	1	1	7.60
0	1	0	7.32
1	0	1	7.41
1	0	0	7.44
1	1	1	7.50
1	1	0	7.40

for 10,000 tournaments. To avoid premature convergence the population size was increased to 500. This modification caused the system to slow down, but the average score increased to 7.95. In the case of two boards the system managed to solve the Tartarus problem completely by achieving the maximum score possible (10).

5.3 Advanced Heuristics

Further analysis showed that the GA sometimes produced a command sequence that forced the agent to move forward into a wall. In our implementation, driving the agent into a wall halts execution of the command sequence and starts a new GA to evolve a new sequence. It was decided that although this move might appear to be illogical, the restarts might be unnecessary. The remainder of the command sequence may contain commands to solve the problem, and although energy might be wasted walking into a wall, a high overall score might be achieved. The effect of switching forcing restarts is shown in Table 3 (third column).

The '6 Block' heuristic and the ESWD heuristics (see section 4.2.3) have been implemented and the results obtained through their use can be seen in Table 3.

Reference to Table 3 allows us to draw the following conclusions, the best score was achieved using the 6-block heuristic, with the use of ESWD and allowing the GA to restart after the agent hits a wall.

The final scores achieved by the GA with the addition of the heuristics can be seen in Figure 2. The 'bump' at score 4 is accounted for by those instances where the GA has pushed 4 blocks together by accident in the beginning of the run. The largest distribution is at score 8, with a bell-like curve around it.

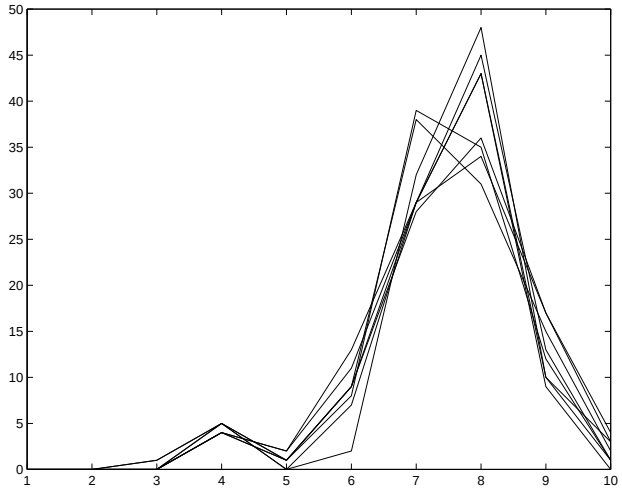


Figure 2: Score distributions for Table 3

5.4 Combining the GA with brute-force

When there is only a small amount of energy left, it is quicker for the system to perform an exhaustive search using every possible command sequence, rather than running the GA again. When the number of amount of remaining energy drops to below a given threshold, the system employees exhaustive search to finish the problem.

In Section 5.5 the exact number of legal strings is calculated for each length. If the number of strings examined by the GA (= #tournaments + population size) is more than the total number of legal strings, exhaustive search will take place.

The GA has always been allowed, so far to produce command strings that if fully executed would use up all the agents' remaining energy. It was felt that some improvement might be forthcoming if the GA was only allowed to produce small strings. This will not only concentrates the evolution into a smaller search space, but also reduces that amount of energy lost.

Table 4 shows the results when examining chromosome lengths between 7 and 20. The GA in figure 12 is also using the brute force method for calculating the final strings.

By only looking ahead a small number of moves (about 12) the scores rise up to 8.77. The reason for this improvement may be attributed to the fact that the GA almost never executes the last moves in the command sequence, while they do count in the fitness calulation.

Whilst starting to solve the problem, new information concerning the landscape will be frequently be found,

Table 4: Results for reducing the number of moves for the GA to look ahead. In column three, the average number of times the GA is run per board is shown. The average number of evaluations per board is the number of strings considered per board (= #runs * (populationsize + #tournaments)). The average number of actions per board is the number of actions (M,L,R) considered by the agent (=#evals * chromosomelength).

Len of chromo	Avg. score	#runs of GA	evals/ board	actions/ board
7	8.42	20.60	30900	216300
8	8.69	19.14	28710	229680
9	8.67	18.48	27720	249480
10	8.66	18.41	27615	276150
11	8.63	18.04	27016	297176
12	8.77	17.33	25995	320040
13	8.67	17.32	25980	337740
14	8.73	16.64	24960	359100
15	8.60	17.23	25845	387675
20	8.67	16.63	24945	498900

after only a few commands have been executed. It is wasteful and even misleading to include the later steps in the fitness function.

There should be an optimum number of moves to look ahead when evolving a command sequence. Too few moves will prevent the GA evolving a meaningful sequence, but too many moves are misleading.

The execution time of a board is typically between 3 and 5 minutes. Note that our system was not optimized for speed, that it was written in Java and ran on a fairly slow processor (Pentium 200 MHz).

5.5 Method for Calculating the Exact Number of Allowed Strings for a Given Length

There is a large number of inefficient command sequences, such as an LR sequence where the R reverses the effect of the L without any side-effect. All strings with LR, RL, LLL or RR in it (RR is equivalent to LL, thus redundant) are therefore not considered when doing an exhaustive search.

The number of 'legal' strings can be calculated as follows. After an M, what can follow is M, LLM, LM or RM. The rewrite rules are given in Figure 3.

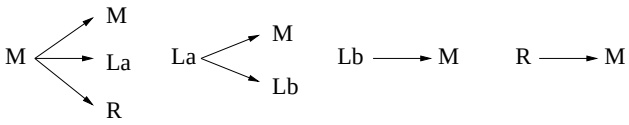


Figure 3: Legal strings

$La(x)$, $Lb(x)$, $R(x)$ and $M(x)$, i.e. the number of Las, Lbs, Rs and Ms at level x in the tree are calculated as follows:

$$\begin{aligned} La(x) &= M(x-1) \\ Lb(x) &= M(x-1) \\ R(x) &= M(x-1) \\ M(x) &= M(x-1) + La(x-1) + \\ &\quad + Lb(x-1) + R(x-1) \end{aligned}$$

with $M(0) = 1; La(0) = Lb(0) = R(0) = 0$. Level $x = 0$ is artificial, but with this initial setting all legal strings of length 1 and higher are correct.

5.6 A comparison to a non-evolutionary heuristic

Given the success of the exhaustive search in enhancing the GA, a full comparison of solving the Tartarus problem by replacing the GA with exhaustive search has been carried out. All the heuristics used to produce the data shown in Table 4 are still in use. The only difference is that instead of using an GA to evolve the command sequence using mutation and crossover, every possible command sequence generated using the rules in Section 5.5 is evaluated and the best taken as the command sequence. The maximum score presented in Table 5 (8.81) is slightly greater than that presented in Table 4 (8.77). An exhaustive search will usually always outperform an Genetic Algorithm, given the non-deterministic nature of the GA. Note though that for shorter lengths, the GA outperforms the exhaustive search, which is most likely due to the greater number of restarts of the GA. What is significant is the number of evaluations required per board, the exhaustive search evaluates 70% more command sequences for an overall gain of 0.5%. A comparison of the exhaustive search (look ahead length 14) and the GA (look ahaead length 12) may be seen in Figure 4. The exhaustive search method is especially good at scoring the maximum 10 points, while the GA score distribution peaks between 8 and 9. This would suggest that the exhaustive search is better at finding solutions to complete the problem that the GA, due to the exhaustive search always finding the optimal partial solution for the current board state. The exhaustive search heuristic performs best with a look

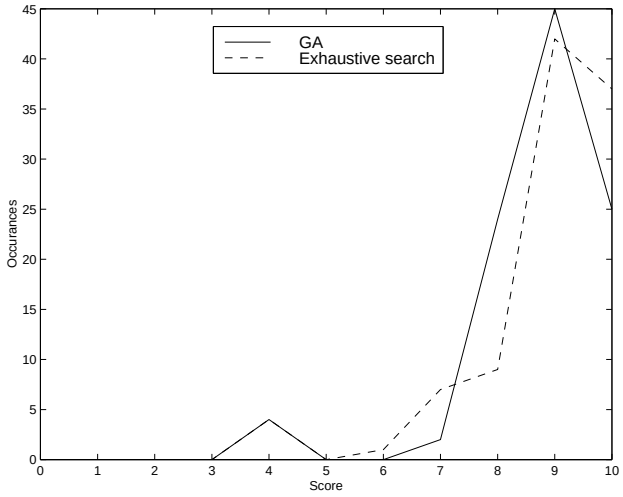


Figure 4: Score distributions for the GA with chromosome length 12 and the exhaustive heuristic with chromosome length 14

ahead of 14. This may be partly due to the fact that we have 80 energy points. If we assume that the algorithm produces stings of length l and restarts n times. The best performance will be recieved in situations where $n * l$ is equal to the energy level (ie all the moves in the final string can be executed). If we examine the nl relationship below we can deduce that a length of 14 with 6 restarts allows 10 out of 14 moves in the final string to be evaluated. Looking forward to the results in Table 5, we can see that indeed $l = 15$ performs worse than both $l = 16$ and $l = 14$.

```
6*12 = 72  7*12 = 84
6*13 = 78  7*13 = 91
5*14 = 70  6*14 = 84
5*15 = 75  6*15 = 90
4*16 = 64  5*16 = 80
```

Further research is needed to determine the exact relationship between chromosome length and the final result. A major problem is the unpredictablity of the number of runs of the GA. The number of runs is determined by the nature of the landscape that the agent is operating in.

5.7 Upscaling properties

In this section we will investigate how well our approach scales up to larger boards with more blocks. Following [Teller, 1994] we will use the following formulas for the number of pieces and the initial amount

Table 5: Results for reducing the number of moves for the exhaustive heuristic to look ahead. The number of valid command sequences is calculated as in Section 5.5. The last two columns are similar to those in Table 4.

Len	Avg. score	# runs	#valid com seq	evals/ board	actions/ board
1	0.84	80	3	240	240
2	0.96	40	6	240	480
3	3.44	27	13	352	1056
4	6.98	20	28	560	2240
5	7.39	16	60	960	4800
6	7.01	14	129	1806	10836
7	8.19	12	277	3324	23268
8	8.64	10	595	5950	47600
9	8.40	9	1278	11502	103518
10	8.65	8	2745	21960	219600
11	8.79	8	5896	47168	518848
12	8.81	7	12664	88684	1064208
13	8.76	7	27201	190407	2475291
14	8.91	6	58425	350550	4907700
15	8.57	6	125491	752946	11294190
16	8.78	5	269542	1437710	23003360

of energy:

$$\begin{aligned} Pieces &= 1/3 * (N-2)^2 \\ Energy &= 2(N^2 + 2N - 3) - 10 \end{aligned}$$

N is the width (and height) of the board. The -10 in the latter formula is somewhat artificial, but for reasons of comparability we will use it.

The results with chromosome length 12 are given in Table 6. Clearly the scores do not scale up terribly well. The reason for this is the (very) limited amount of initial energy, which makes initial exploration infeasible.

If we allow an initial energy of N^3, as argued in [Balakrishnan and Honavar, 1996], and make two more modifications, results are much better (see Table 7). Note that with larger boards, initial situations may occur that are partly unsolvable, e.g.

```
XX
X X
 XX
```

The modifications are:

- Make explorePoints a decreasing function of time.

Table 6: Results with chromosome length 12 for larger boards

N	Pieces	Energy	Max score	Average score
6	6	80	10	8.77
7	9	110	13	10.96
8	12	144	16	13.01
9	17	182	21	15.78
10	22	224	26	17.82

Table 7: Results with chromosome length 12 for larger boards with energy=N^3 and square penalty

N	Pieces	Energy	Max. score	Score	Energy used
6	6	216	10	9.23	113.38
7	9	343	13	12.17	146.20
8	12	512	16	14.95	206.52
9	17	729	21	19.55	290.75
10	22	1000	26	23.06	419.26

After some tuning we used the following formula:

$$ep = 2 + 10 \cdot e^{-4 * \frac{initialEnergy - energy}{initialEnergy}}$$

- Introduce a penalty for pushing a block into a known four block square. We used a very strong one: fitness = 0 if this happens.

6 Conclusions and future research

The authors have presented a novel approach to the Tartarus Problem. We have achieved the highest score in literature for the Tartarus Problem. An average score of 8.91 has been achieved by the exhaustive search heuristic with the fitness function introduced in this work.

The use of GA combined with the long-term memory gave an average result of 4.5, equivalent to that achieved using parse trees[Teller, 1994] and neural networks[Balakrishnan and Honavar, 1996]. The addition of heuristics to assist with the building of the long-term memory map such as smart wall deduction and the 6-block heuristic improved results. The most significant improvement, scoring 8.77, was achieved by the reduction in the length of the command sequence (chromosome).

Given the relative inefficiency of the exhaustive search, the hybrid GA approach developed by the authors would appear to be the most effective solution to the Tartarus problem yet published.

When allowed more initial energy, the agent scores close to optimal on all boards, even of larger sizes.

The basic agent developed here is now competent at solving the Tartarus problem. Future research may look at the possibilities of carrying out more complex tasks in similar environments. Although the fitness function and some of the heuristics used are specific to this problem, it remains to be seen whether the approach taken can be reapplied elsewhere.

Acknowledgements

We would like to thank Ernesto Costa and Jason Maassen for their helpful contributions to our work. This work was based on an problem set at the CoIL Summer School 2000. We would like to thank Adrian Trenaman for setting the problem.

References

[Ashlock and Freeman, 2000] Dan Ashlock and Jennifer Freeman. A pure finite state baseline for tartarus. In *CEC 2000*, volume 2, pages 1223–1230, 2000.

[Ashlock and Joenks, 1998] Dan Ashlock and Mark Joenks. ISAc lists, A different representation for program induction. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 3–10, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[Ashlock and McRoberts, 1997] Dan Ashlock and McRoberts. A gp-automata reprise of astro teller's bulldozer experiment. Technical Report AM97-17, ISU Mathematics, 1997.

[Balakrishnan and Honavar, 1996] Karthik Balakrishnan and Vasant Honavar. On sensor evolution in robotics. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 455–460, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[Teller, 1994] Astro Teller. The evolution of mental models. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 9, pages 199–219. MIT Press, 1994.