Space Searching Algorithms Used by Fungi

Elitsa Asenova
McGill University, Department of Electrical & Computer Engineering
Montreal, Canada, H3A 0C3
elitsa.asenova@mail.mcgill.ca

Eileen Fu
McGill University, Department of Electrical & Computer Engineering
Montreal, Canada, H3A 0C3
tian.s.fu@mail.mcgill.ca

Dan V. Nicolau Jr.
Queensland University of Technology
School of Mathematical Sciences
Queensland 4000 Australia
dan.nicolau@qut.edu.au

Hsin-Yu Lin
McGill University
Department of Bioengineering
Montreal, Quebec, Canada, H3A 0C3
hsin-yu.lin@mail.mcgill.ca

Dan V. Nicolau*
McGill University
Department of Bioengineering
Montreal, Quebec, Canada, H3A 0C3
dan.nicolau@mcgill.ca

ABSTRACT
Experimental studies have shown that fungi use a natural program for searching the space available in micro-confined networks, e.g., mazes. This natural program, which comprises two subroutines, i.e., collision-induced branching and directional memory, has been shown to be efficient compared with the suppressing one, or both subroutines. The present contribution compares the performance of the fungal natural program against several standard space searching algorithms. It was found that the fungal natural algorithm consistently outperforms Depth-First-Search (DFS) algorithm, and although it is inferior to *informed* algorithms, such as A*, this under-performance does not increase importantly with the increase of the size of the maze. These findings encourage a systematic effort to harvest the natural space searching algorithms used by microorganisms, which, if efficient, can be reverse-engineered for graph and tree search strategies.

Categories and Subject Descriptors
I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search – *Graph and tree search strategies.*

General Terms
Algorithms, Performance, Experimentation.

Keywords
Maze searching, natural algorithms, biomimetics, microfluidics.

1. INTRODUCTION
Biological entities have evolved highly efficient strategies for space searching, which are essential to their survival[1]. In many instances, non-human biological algorithms appear to be superior to those used by humans[2], which opens up the opportunity for 'reverse engineering' of those algorithms for practical applications[3]. Presently, however, the bio-inspired algorithms are just "inspired" by, rather than "reverse-engineered" from, natural algorithms "developed" by biological entities.

Previously, it was observed[4,5] that fungi behave very differently in micro-confined spaces, and that they use specific programs for searching space available for growth. While different species present different variants of this fungal program, its framework is common and it consists of the interplay of two 'sub-routines': collision-induced branching, and directional memory. These studies also demonstrated that the natural program comprising the two 'sub-routines' is markedly superior to variants where one of these is, or both are suppressed.

The present contribution aims to extend the analysis of the efficiency of the natural space searching program used by fungi, by benchmarking it against classical space searching algorithms.

2. METHODS
2.1 Fungal Space Search Algorithms
Previous works examined the growth of two species of filamentous fungi, i.e., *Pycnoporus cinnabarinus*[4] and *Neurospora crassa*,[5,6] inside a confined maze-like microfluidics structure (Fig1). The fabrication of the mazes, described elsewhere,[4] consisted of (i) patterning of a silicon mold using standard photolithography, followed by deep reactive ion etching; (ii) making a negative relief poly(dimethylsiloxane) (PDMS) stamp, by casting and curing the degassed PDMS prepolymer and curing agent mixture, (iii) rendering the hydrophobic PDMS surface hydrophilic by plasma treatment; (iv) sealing the PDMS onto a flat base glass layer. The enclosed structure had lateral openings, allowing the introduction of inoculation and media.

2.1.1 Corner-induced Branching
The fungus grows until it reaches a wall. If the angle of attack is shallow, the fungus will slide along the wall. However, if the tip of the fungal hypha, i.e., the filamentous extension by which fungi grow, reaches a corner, or a geometry that does not allow an exit, e.g., due to directional memory (see further), then the hypha will branch. Consequently, the fungus has two branching mechanisms: a low frequency one, which is equivalent to the one used when growing in open spaces; and a high frequency one, triggered by the collision with difficult geometries, which occurs often in confined spaces, e.g., mazes. The position of the branching is species-specific, e.g., *P cinnabarinus*[4] branches away from the leading tip, *N. crassa*,[5,6] branches at the leading tip.

BICT 2015, December 03-05, New York City, United States
Copyright © 2016 ICST
DOI 10.4108/eai.3-12-2015.2262591

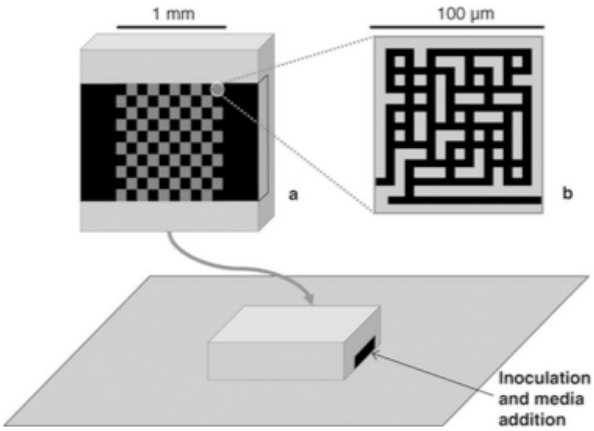

Figure 1. Design of the networked microfluidics chip for the 'harvesting' of the fungal algorithms for space searching.

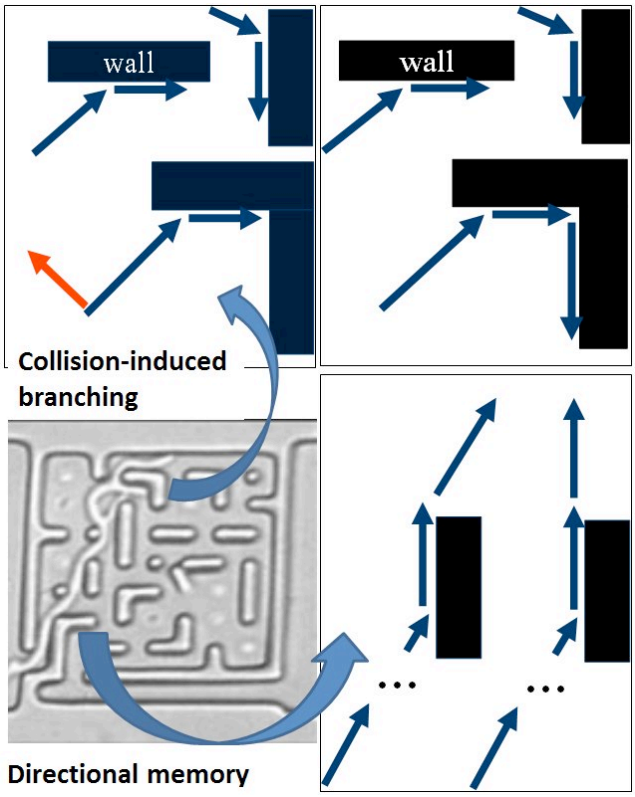

Figure 2. Space searching algorithms used by a fungus (here *P. cinnabarinus*). Top left panel: Collision-induced branching. The hyphae can slide along walls if the angle of attack is shallow. When facing a corner, the hyphae will branch (red arrow), unlike "no collision-induced branching" (top right panel). Bottom right panel: Directional memory. The hyphae slide along the walls, then redirect their growth in the same direction they had initially (left scheme), unlike "no-directional memory" (right). An image of the actual fungus growing in a maze (bottom left panel) shows both the collision-induced branching events (top arrow), and the overall directional memory (bottom arrow).

2.1.2 Directional Memory

Each branch 'remembers' its initial direction of growth, and while it has to negotiate various geometries, whenever the branch has the opportunity to grow in the direction it had initially, it will follow this with a high probability. Moreover, the branch will not grow further if the available space opens in a direction which is more than orthogonal with its initial direction, i.e., angles larger than 90° are not allowed. If no alternative is available, the hypha will stop growing and will often branch.

2.2 Bio-inspired Search Algorithms

2.2.1 Maze Generation

Since maze-solving can be generalized to graph searching, the test mazes have been generated by a graph-based algorithm (Fig. 3). Specifically, the graph was implemented in the form of a "*w*-by-*h*" grid of cells, with w and h representing the width and height of the maze. Each cell represents a square on the grid whereas the black lines indicate the walls (Fig 3B). If each cell is the vertex of a graph, then a cell may be connected to another cell if there are no walls between them (Fig 3C). Each cell can have from 0 to 4 walls; 0 walls means the cell is a 4-way intersection, and 4 walls means the cell is completely closed off.

The maze is initialized to be a 2D array of disconnected vertices, visually represented as a rectangular grid. Two randomized options are provided for maze generation, each following a separate algorithm for removing walls in the grid, allowing for two "styles" of mazes be created for testing and simulation. The first algorithm implements a queue-based, Depth-First-Search (DFS) methodology[7], which is a first-in-first-out graph-searching algorithm. Starting from the entrance cell, a random wall leading to an adjacent cell is removed, and that cell is marked as "visited". The algorithm then proceeds to remove a random wall in the newly- visited adjacent cell, and the next, until the user-supplied exit point is found and all cells have been visited. This results in a maze containing many long corridors, and few branches.

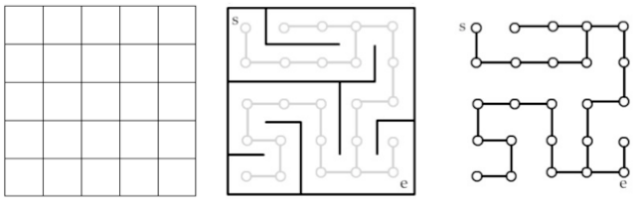

Figure 3. Evolution of the construction of a maze.

The second option implements Kruskal's algorithm[8]. In this case, the initial maze of closed cells can be seen as a grid of disjoint sets, each containing only one vertex. A separate array containing all the walls is also created. Then, at random, walls are chosen out of this array; if the wall divides two cells that belong to separate sets, the wall is removed and the two cells are joined into one set. This is repeated until all cells belong in a single set. In contrast to DFS, Kruskal's algorithm results in a maze that branches frequently in all directions, and contains few long paths.

2.2.2 Search Algorithm

The natural search routine (Bio-Inspired Algorithm, BIA) was implemented in Java, using the two fungal sub-routines, i.e., collision-induced branching, and directional memory, with the fundamentals described in section 2.1.

2.2.2.1 Corner-induced Branching

From an algorithm perspective, instead of using recursion or a stack, cells were stored in a data structure that allows direct access, in this case, an array. This way, when a dead end is reached, the next cell to visit is determined algorithmically rather than using a last-in-first-out method.

2.2.2.2 Directional Memory

Directional memory is easily implemented in Java by including the starting angle of each branch object as a class constant, and the current angle of the branch as a class variable. A simulated fungal branch will travel at its starting angle as far as possible, whenever possible, and only changes its current angle when no paths are available in its starting direction.

3. RESULTS AND DISCUSSION

3.1 Space searching and solving mazes

Solving mazes is a difficult algorithmic exercise, which is why mazes are used to estimate the optimality of the behavioral response, or intelligence, of many higher organisms including ants, bees, mice, rats, octopi, and humans,[9] as well as artificial intelligence-enabled robots.[10]

The efficiency of space searching algorithms depends greatly on the geometry of the space and specifically confinement properties. At one end of the scale, empty space without obstacles cannot be explored any better than by using a diffusion, or diffusion-like approach, e.g. a Levy flight. Depending on how the nutrients (or other resources of interest) are distributed in such a space, it appears generally that Levy flight processes are both what biological systems use and what is actually mathematically optimal.

At the other end of the scale, the space search problem in a maze, a highly constrained geometry, reduces specifically to the problem of graph connectedness. Because the maze is a graph, and it is required for an exit to be found, this translates into asking a computational system, e.g., a fungus, to find if the entry and exit of the maze/graph are connected (and if so, how), or not. This problem, of graph connectedness, is known to be in computational class P and can be solved in a number of ways, but most commonly this is done using "breadth-first search", first proposed in the 1950s. Its time complexity is O(V+E) where V and E are the number of edges and vertices, respectively. In general we think of graphs (and mazes) as being specified by the number of vertices and in the case of a graph were most vertices are locally connected together, the time to establish the path from entry to exit would be a fixed, low (say, 1-3) power of E.

3.2 Fungal 'intelligence'

Microfluidics technology has allowed the miniaturization of maze structures, which have been used to test the maze-solving ability of both abiotic[11] and biotic[12] agents and to modulate and observe the collective behavior of bacteria.[13,14]

An interesting aspect of the fungal search algorithms is that they do not require nutrient-related clues regarding the geometry of the environment. Previous studies have documented maze-solving by placement of nutrients at the exits,[12] or quorum-related signaling,[15] but the study of fungal space searching suppressed nutrient gradients. In this context, the observed response is consistent with the observation that natural fungal habitats are nutritionally heterogeneous and require hyphae to efficiently continue colony extension in the absence of chemotactic cues.

The ubiquity of fungi in microconfined mazelike habitats suggests that they may be efficient solving agents of geometrical problems. Although this ability has been assessed[4] versus variants missing one, or the other sub-routine, or both, the performance of the overall fungal space search program versus standard path search algorithms was not previously examined.

3.3 Assessing the Bio-Inspired Algorithm

To examine the Bio-Inspired Algorithm (BIA), we tested its completeness, reachable state space and optimality under both non-randomized and randomized mazes, generated by DFS (which resulted in almost no dead ends), and by Kruskal's algorithm (leading to multiple dead ends), respectively.

3.3.1 Completeness

Starting from a root node (beginning of maze), this test aims to find to what extent BIA finds the leaf node (end of maze). The tests were run on different maze sizes and by placing the starting and ending vertices at various positions. For the maze sizes up to 50x50, BIA solved every maze with different starting and ending positions. Upon encountering a dead end, the algorithm goes back to a previously branching point and resumes from there. While this approach of the natural program appears not to be the best solution, the algorithm will always find the exit on finite mazes. In this regard, it must be noted that in many biological instances, robustness of the behavior is more important than efficiency.

3.3.2 Reachable State Space

Mazes can serve as a background to state-space searching because this is composed of an environment (the maze) that is divided into equally sized units (states). For this test, we defined a start state (beginning of the maze) and final state (end of maze) and counted the number of covered nodes when the final state was reached. In Fig. 4, it can be seen that when the maze size grows, the portion of reachable state space remains constant (around 2/3 is explored).

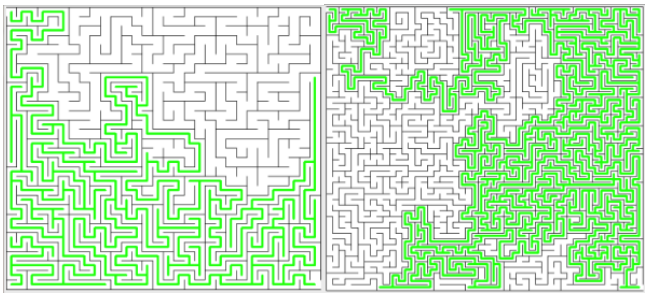

Figure 4. Examples of the coverage of the mazes when explored by BIA, for 20x20 (left), and 50x50 (right) mazes.

3.4 Comparison with Other Maze Solving Algorithms

Noting that microorganisms, e.g., fungi, have complex, and different from computer algorithms, 'objective function,' against which they optimize their behavior, it is of critical importance for "mathematical biomimetics", i.e., study of natural algorithms in the view of their reverse engineering, to benchmark these natural algorithms against a standard one with similar scope.

3.4.1 *Reachable State Space*

This test was applied to examine the amount of memory necessary. The quantification method for reachable state space was mentioned in section *3.1.2*. Because this is an uninformed search (the search is the same regardless of the context), we observed that BIA is less efficient than and an informed search, e.g., A*.[16] However, BIA is consistently better than DFS, i.e. the covered area while the final state was reached for BIA will be larger than informed search but smaller than DFS. While the use of A*, which as an informed search algorithm, may be 'unfair', this conservative benchmarking is justified for a comprehensive comparison. As shown by Fig. 5, DFS takes the most space when performing maze search, while BIA is ~20% more efficient in larger mazes (30x30 and up). A*, being an informed search, is, as expected, much more efficient and therefore much more compact.

Also, it is important to note the different between randomized and nonrandomized mazes. In the non-randomized maze, BIA and DFS are more similar in performance due to the limited number of dead ends (Fig 5A). With randomized mazes where they are multiple dead ends, the difference between BIA and DFS is visible even with smaller mazes (Fig 5B).

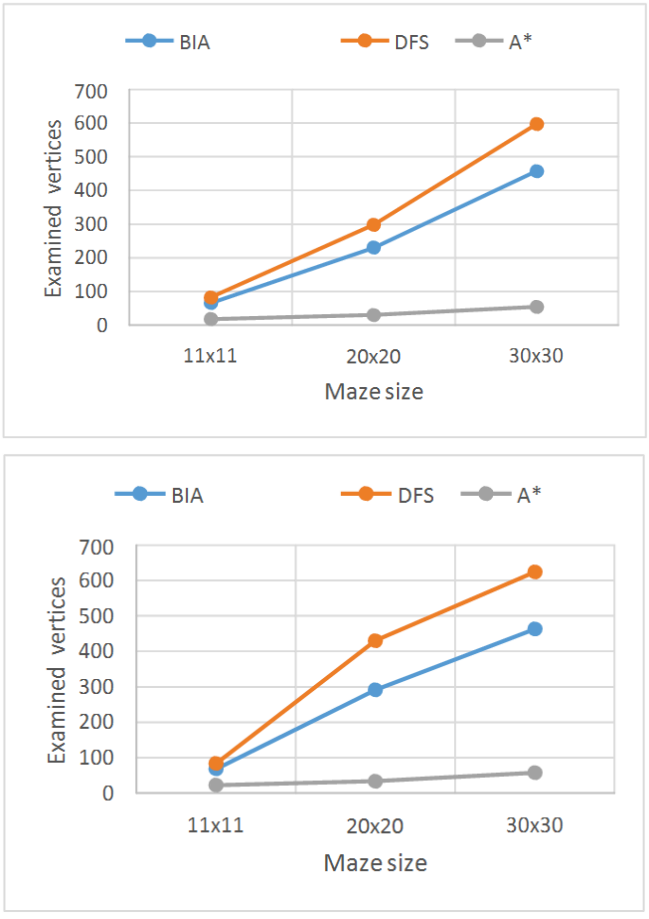

Figure 5. Reachable space, for (A, top) non-randomized, and (B, bottom) randomized space, as a result of the exploration of mazes with various sizes by BIA, DFS and A* algorithms, respectively.

3.4.2 *Running Time*

The comparison of the performance of the natural (BIA) algorithm with standard ones has been extended and deepened, by benchmarking the experimental running time on a computer for uninformed algorithms, i.e., BIA, and DFS, and for informed maze search algorithms, i.e., Best First Search[17], Jump Point Search,[18] and Dijkstra[19]. The tests have been run on a Dell Inspiron i3. The mazes used for tests were non-randomized.

Fig 6 presents the computational performance of various maze-solving algorithms. For large mazes, the execution time increases significantly for uninformed algorithms, i.e., BIA and DFS. For uninformed algorithms, DFS performs slightly better in smaller maze size (up to 30x35). However, when the maze size keeps growing, BIA performs much better than DFS.

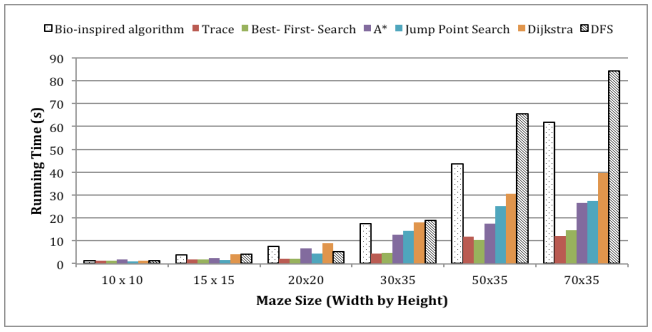

Figure 6. Running time for various maze solving algorithms, both uninformed (BIA, DFS) and informed (A*, Best First Search, Jump Point Search, Dijkstra).

Again, a more 'correct' comparison, i.e., between the uninformed algorithms DFS and BIA, demonstrates the robustness of the natural algorithm (Fig 7). Interestingly, in smaller mazes (up to 30x30), DFS performs slightly better than BIA, but at mazes with sizes larger than 40x40, BIA performs 20-40% better (tested until 70x70).

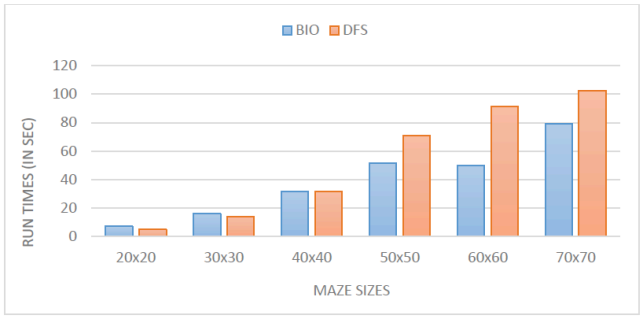

Figure 7. Running time for uninformed maze solving algorithms, i.e., BIA and DFS for mazes of various sizes.

4. CONCLUSION

We compared the performance of the natural program for space search used by fungi, as documented by previous experimental studies, against several standard space searching algorithms, both uninformed of the maze structure, i.e., Depth-First-Search (DFS) algorithm, and informed algorithms, such as A*, Best First Search, Jump Point Search and Dijkstra. It was found that the fungal natural algorithm consistently outperforms the DFS algorithm, and although it is inferior to informed algorithms, such as A*, this under-performance does not increase importantly with the increase of the size of the maze. These findings encourage a systematic effort to harvest the natural space searching algorithms used by microorganisms, which, if efficient, can be reverse-engineered for graph and tree search strategies. Another direction

of research could be the optimization of the natural space searching algorithm, e.g., via evolutionary computing.

5. ACKNOWLEDGMENTS

Financially supported by the European Union Seventh Framework Programme (FP7/2007-2011) under grant agreement number 613044 (ABACUS); and Defense Advanced Research Projects Agency (DARPA) under grant agreement N66001-03-1-8913.

6. REFERENCES

[1] Cho, H. et al. Self-organization in high-density bacterial colonies: Efficient crowd control. PLoS Biology 5, 2614-2623, doi:10.1371/journal.pbio.0050302 (2007).

[2] Helbing, D. Traffic and related self-driven many-particle systems. Reviews of Modern Physics 73, 1067-1141, doi:10.1103/RevModPhys.73.1067 (2001).

[3] Binitha S, S. S., S. A Survey of Bio inspired Optimization Algorithms. International Journal of Soft Computing and Engineering 2, 137-151 (2012).

[4] Hanson, K. L. et al. Fungi use efficient algorithms for the exploration of microfluidic networks. Small 2, 1212-1220, doi:10.1002/smll.200600105 (2006).

[5] Held, M., Edwards, C. & Nicolau, D. V. Probing the growth dynamics of Neurospora crassa with microfluidic structures. Fungal Biology 115, 493-505, doi:10.1016/j.funbio.2011.02.003 (2011).

[6] Held, M., Lee, A. P., Edwards, C. & Nicolau, D. V. Microfluidics structures for probing the dynamic behaviour of filamentous fungi. Microelectronic Engineering 87, 786-789, doi:10.1016/j.mee.2009.11.096 (2010).

[7] Korach, E. & Ostfeld, Z. Recognition of DFS trees: sequential and parallel algorithms with refined verifications. Discrete Mathematics 114, 305-327, doi:10.1016/0012-365X(93)90375-4 (1993).

[8] Kruskal, J. B. On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical Society 7, 48–50 (1956).

.

[9] Wasserman, E. A. & Zentall, T. R. Comparative Cognition: Experimental Explorations of Animal Intelligence. (2012).

[10] Nelson, A. L., Grant, E., Galeotti, J. M. & Rhody, S. Maze exploration behaviors using an integrated evolutionary robotics environment. Robotics and Autonomous Systems 46, 159-173, doi:10.1016/j.robot.2003.11.002 (2004).

[11] Fuerstman, M. J. et al. Solving mazes using microfluidic networks. Langmuir 19, 4714-4722, doi:10.1021/la030054x (2003).

[12] Nakagaki, T., Yamada, H. & Tóth, Á. Maze-solving by an amoeboid organism. Nature 407, 470, doi:10.1038/35035159 (2000).

[13] Park, S. et al. Enhanced Caenorhabditis elegans locomotion in a structured microfluidic environment. PLoS ONE 3, doi:10.1371/journal.pone.0002550 (2008).

[14] Park, S. et al. Influence of topology on bacterial social interaction. Proceedings of the National Academy of Sciences of the United States of America 100, 13910-13915, doi:10.1073/pnas.1935975100 (2003).

[15] Park, S. et al. Motion to form a quorum. Science 301, 188, doi:10.1126/science.1079805 (2003).

[16] Hart, P. E., Nilsson, N. J. & Raphael, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics 4, 100-107, doi:10.1109/TSSC.1968.300136 (1968).

[17] Pipe, A. G., Fogarty, T. C. & Winfield, A. in IEEE Conference on Evolutionary Computation - Proceedings. 485-489.

[18] Harabor, D. & Grastien, A. in Proceedings International Conference on Automated Planning and Scheduling, ICAPS.January edn 128-135.

[19] Dijkstra, E. W. A note on two problems in connexion with graphs. Numerische Mathematik 1, 269-271 (1959).