Building Simple Non-identical Organic Structures with Dispersive Flies Optimisation and A* Path-finding

Michael King [1] and **Mohammad Majid Al-Rifaie** [2]

Abstract. The use of natural based algorithms within films, television programs and games is becoming increasingly prominent and keeping the consumers interested and immersed in the world they are being exposed to is becoming a pre-requisite. This paper proposes a new method of generating unique structures based upon a simple hybrid model through the use of A* Path-finding and a swarm intelligence algorithm, Dispersive Flies Optimisation. Every time a structure is built, this hybrid method generates a unique form which is non-identical to structures created in different runs. Within the game industry, these structures could potentially assist providing users with unique environments presenting different feels, despite staying loyal to designers' overall structure.

1 INTRODUCTION

The use of algorithms which take inspiration from the natural world grows in popularity[3] as the technology and hardware have upgraded to such a degree where complex algorithms are now accessibly by many. This results in games becoming more realistic, thus producing more natural environments which aid in immersing the user into the world that they are being shown[4]. To build on the existing techniques, new and innovative algorithms are required to match the expectations of the user and to ensure that more powerful hardware is being used to its full potential.

This paper proposes a novel hybrid algorithm to help aid in generating naturally looking, "organic" structures. The two adapted algorithms used in this work will be described in the next section, followed by details in which the hybrid algorithm is tasked to create the "organic" non-identical structures. The rationale behind using swarm intelligence techniques in such work lies behind their non-deterministic behaviour, which mounts to the creation of different structures with every individual run of the system. The choice of Dispersive Flies Optimisation (DFO) as one such technique is made due to the fact that DFO, in addition to its simple and minimalistic structure [2], does not suffer from having several tunable parameters, therefore paving the way to focus on other areas of interest. Additionally the outperformance of DFO in comparison with other well-known optimiser offers an additional reason to utilise DFO (more details are provided later in this work). Therefore, the this swarm intelligence technique is coupled with the well established A* technique which is adapted for the purpose of this work. To the best of the author's knowledge, these two algorithms have not been explored together before.

[1] Corresponding author. Goldsmiths University of London, United Kingdom, email: mking038@gold.ac.uk

[2] Goldsmiths University of London, United Kingdom, email: m.majid@gold.ac.uk

2 BACKGROUND

In this section, the two algorithms used in this paper – Dispersive Flies Optimisation and A* path-finding – are briefly explained. The use of Dispersive Flies Optimisation within a game/film environment has not been done before due to the relatively new introduction of the algorithm. Whilst A* Path-finding is a commonly used path-finding algorithm in various domains including games, however to the best of the authors' knowledge, it has not been adapted in the way described in this paper.

2.1 DISPERSIVE FLIES OPTIMISATION

Dispersive Flies Optimisation [1] (DFO) is population based, global optimiser which has been introduced in 2014. It takes inspiration from the observation of flies and their behaviour when swarming around a food source. In summary, DFO is a simple numerical optimiser over continuous search spaces. Despite the algorithm's simplicity with only two tunable parameters, it is shown that DFO outperforms the standard versions of the well-known Particle Swarm Optimisation [8], Genetic Algorithm (GA) [6] as well as Differential Evolution (DE) algorithms [9] on an extended set of benchmarks over three performance measures of error, efficiency and reliability. It is shown that DFO is more efficient in 84.62% and more reliable in 90% of the 28 standard optimisation benchmarks used; furthermore, when there exists a statistically significant difference, DFO converges to better solutions in 71.05% of problem set. In addition to theoretical research on this algorithm, DFO has recently been applied to medical imaging, the study of aesthetic evaluation of digital images, as well as its use in philosophical discussion in the area of computational creativity.

To describe the algorithm, it is important to know that the swarming behaviour of the flies in DFO consists of two tightly connected mechanisms, one is the formation of the swarms and the other is its breaking or weakening. The algorithm and the mathematical formulation of the update equations are introduced below.

The position vectors of the population are defined as:

$$\vec{x}_i^t = \left[x_{i1}^t, x_{i2}^t, ..., x_{iD}^t\right], \qquad i = 1, 2, ..., NP \tag{1}$$

where t is the current time step, D is the dimension of the problem space and NP is the number of flies (population size).

In the first generation, when $t = 0$, the i^{th} vector's d^{th} component is initialised as:

$$x_{id}^0 = x_{\min,d} + r\left(x_{\max,d} - x_{\min,d}\right) \tag{2}$$

where r is a random number drawn from a uniform distribution on the unit interval $U(0, 1)$; $x_{\min}$ and $x_{\max}$ are the lower and upper initialisation bounds of the d^{th} dimension, respectively. Therefore, a

population of flies are randomly initialised with a position for each flies in the search space.

On each iteration, the components of the position vectors are independently updated, taking into account the component's value, the corresponding value of the best neighbouring fly (consider ring topology) with the best fitness, and the value of the best fly in the whole swarm:

$$x_{id}^{t} = x_{nb_i,d}^{t-1} + \mathrm{U}(0,1) \times (x_{sb,d}^{t-1} - x_{id}^{t-1}) \quad (3)$$

where $x_{nb_i,d}^{t-1}$ is the value of the *n*eighbour's *b*est fly of $\vec{x_i}$ in the d^{th} dimension at time step $t-1$; $x_{sb,d}^{t-1}$ is the value of the *s*warm's *b*est fly in the d^{th} dimension at time step $t-1$; and $\mathrm{U}(0,1)$ is the uniform distribution between 0 and 1.

The algorithm is characterised by two main components: a dynamic rule for updating flies position (assisted by a social neighbouring network that informs this update), and communication of the results of the best found fly to other flies.

As stated earlier, the swarm is disturbed for various reasons; one of the positive impacts of such disturbances is the displacement of the disturbed flies which may lead to discovering a better position. To consider this eventuality, an element of stochasticity is introduced to the update process. Based on this, individual components of flies' position vectors are reset if the random number, r, generated from a uniform distribution on the unit interval $U(0,1)$ is less than the *disturbance threshold* or *dt*. This guarantees a proportionate disturbance to the otherwise permanent stagnation over a likely local minima. Algorithm 1 summarises the DFO algorithm.

Algorithm 1 Dispersive Flies Optimisation

while Function Evalutions $<$ Evaluations Allowed **do**
  for $i = 1 \rightarrow NP$ **do**
    $\vec{x}_i$.fitness $\leftarrow f(\vec{x}_i)$
  end for
  $sb \leftarrow \{sb,\ \forall\ f(\vec{x}_{sb}) = \min(f(\vec{x}_1), f(\vec{x}_2), ..., f(\vec{x}_{NP}))\}$
  for $i = 1 \rightarrow NP$ **do**
    $nb_i \leftarrow \left\{nb_i,\ \forall\ f(\vec{x}_{nb_i}) = \min\left(f(\vec{x}_{\mathrm{left}_i}), f(\vec{x}_{\mathrm{right}_i})\right)\right\}$*
  end for
  for $i = 1 \rightarrow NP$ **do**
    for $d = 1 \rightarrow \mathrm{D}$ **do**
      $\tau_d \leftarrow x_{nb_i,d}^{t-1} + \mathrm{U}(0,1) \times (x_{sb,d}^{t-1} - x_{id}^{t-1})$
      if $(r < dt)$ **then**
        $\tau_d \leftarrow x_{\min,d} + r\,(x_{\max,d} - x_{\min,d})$
      end if
    end for
    $\vec{x}_i \leftarrow \vec{\tau}$
  end for
end while

* $\vec{x}_{\mathrm{left}_i} = \vec{x}_{i-1}$ and $\vec{x}_{\mathrm{right}_i} = \vec{x}_{i+1}$

This algorithm allows the agents to search the space they are within whilst maintaining a natural ascetic.

2.2 A* PATH-FINDING

A* path-finding algorithm, first introduced by Hart in 1968 [7], is a well known algorithm that is renowned for being quicker than most other path-finding algorithms when put into a situation in which the origin and target location are already defined on a graph that is completely known to the agent [11].

A* Path-finding algorithm, which is an extension of Dijkstra's 1959 algorithm [5], works by first building a graph of nodes and weighted edges. The nodes all represent a location within the search space and the weighted edges connect these nodes together with a given weight for how much it costs to go from one node to the other. The agent can then traverse this graph by searching all the nodes that have an edge to the current node that the agent is located. It scans through the graph to find the target node and will create the path that is the shortest (path with the least amount of weight). It scans through the graph by having a heuristic function which returns a cost value that is then added to an overall count on how expensive it is to get to that node. The algorithm then uses a priority queue to keep the lowest weight path on top which ensures that the amount of time the algorithm takes to scan a path for the target is reduced, thus saving precious processing time.

3 ADAPTED A* AND DFO

This section summarises the process through which the two aforementioned algorithms are adapted in order to obtain the functionality required for the purpose of this work.

Minor changes are introduced to DFO before applying the algorithm to the problem in hand. DFO requires a fitness function which would allow each individual fly to evaluate the suitability of its current position. For this work, the fitness function is the distance from the food object thus giving a relatively simple but effective fitness yet ensuring that the problem is kept as a minimisation problem. The only other modification made is for the search phase in which the Y co-ordinate was clamped to the diameter of the flies to prevent early building in sub-optimal locations. In other words, the flies are grounded during the search/convergence phase to avoid multi-layered structures from being built before the actual building phase commences. The purpose for this modification is due to the fact that this system is intended for multiple targets. It should also be noted that this is all within a 3D environment which is governed by "physics" so each fly has a 'body' which can collide with the world and other flies (this process is illustrated in the figures provided later in the paper). The relevance of this modification will be highlighted further in the next section.

Further changes are applied to A* path-finding algorithm in order to utilise its capabilities for the work presented. In one such changes, the nodes are made to be the agents that can also traverse the graph. The edges connecting the nodes are also set to be dynamic as they were added and removed dependent on whether the two relevant agents were touching or not. This allows a dynamic graph of nodes and edges that agents could refer to when needed. This modification is crucial if agents are required to climb a structure as they can identify a path to the other agent i.e. the target agent they want to get to by accessing the graph. To facilitate this kind of mapping, the cost between two nodes is increased dependent on how many edges those nodes have in total, this stops a path being made in the centre of the structure.

Acknowledging the No Free Lunch Theorem [10], and the fact that A* algorithm is not suited for all situations, it is decided that the A* path-finding algorithm is better placed in dealing with the scenario of this work; this is due to the agents' global knowledge of the entire graph whilst also always having a known start and an end goal.

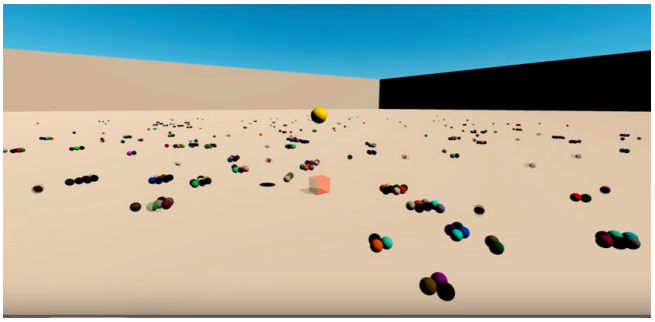
The collaboration between these two algorithms ensures creating a structure that both touches the ground and leads to the target. As mentioned before, by first clamping the Y co-ordinate within the search phase it is ensured that the agents do not attempt to begin

building in a location that is not the best (this is more relevant when the swarm has multiple food sources to choose from). This allows the swarm to then focus on the absolute best base position for the structure (the best position is determined by storing the swarm's best fitness and the relevant position during the search phase). Once the search phase is over the clamp on the Y co-ordinate is removed. The swarm then begins to build upwards towards the designated food source. At regular intervals, A* is then used to create a path from the best agent to the ground (the best agent is the agent with the lowest fitness within the swarm which is different to the best position as the best position is a fixed location that is found from the search phase where as the best agent is constantly changing during the building phase). The agents along this path are first given the target of their current position and then set to "inactive" meaning they would not be seeking a new target. This ensures that the structure maintains stability[3].

4 EXPERIMENTS AND RESULTS

In the experiment reported in this work, one food source (seen as the yellow sphere in Figs. 2, and 4) in is created and agents are set to get to. The food source is located at position $(0, 5, 0)$ or 5 units above the ground. All agents are scattered (initialised) across the search space randomly. Ring topology is used for this work and the population size is set to 1,000. The search phase lasts for 5000 frames whilst evaluation and update happens every 2 seconds.

Figure 1. Forming small communities

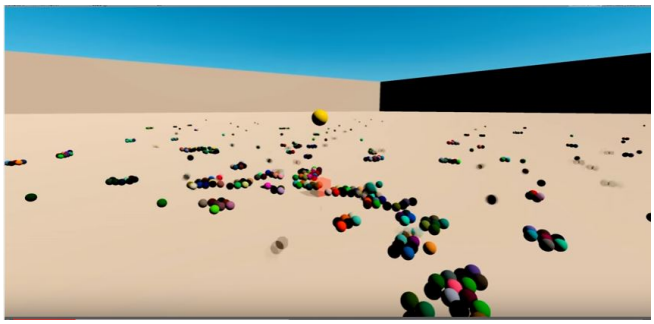


Once started, the agents first drop to the floor and begin to move around to their first target. Over time, it is observed that small communities of agents form in different areas (see Fig. 1). Over time these communities slowly move towards the optimal location where the base is highlighted a transparent box as shown in Figs. 2(a,b). As anticipated, it can be observed that communities which are not on the optimal location shrink over time and merge with communities possessing best fitness. Although the agents have clamping on their target Y co-ordinate they still manage to get on top of a group of other agents. This is due to the way in which each agent moves around. As they hit another agent their own velocity launches them upwards, as they come back down they land on top of a group. When this happens the agent get dispersed away from the group and is moved away to search another area. By the end of the search phase there is a large community surrounding the best position whilst there are small communities/individual agents not connected to this due to the dispersion aspect of DFO (see Fig. 2(c)).

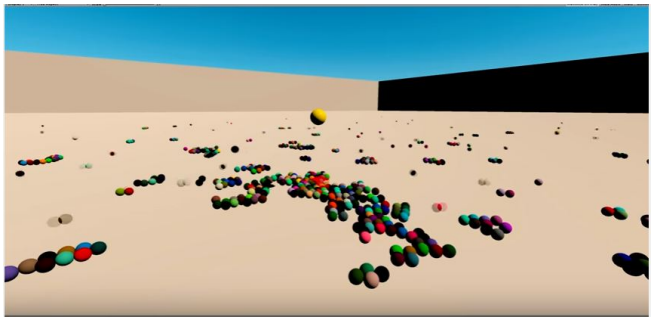
Following the search phase, the building phase commences. The agents immediately climb on top of one another and every time the

[3] In this work, the algorithms were developed in Unity and written in C Sharp.

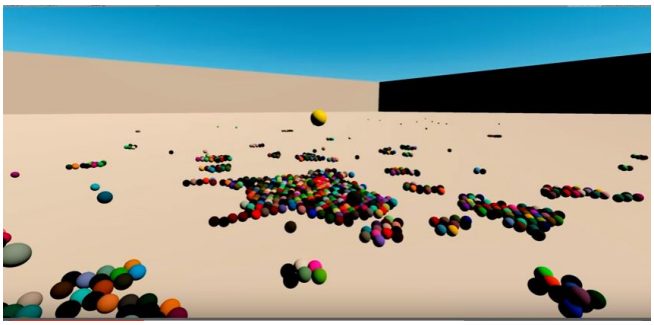
Figure 2. Search and convergence phase



(a) Moving towards the base # 1



(b) Moving towards the base # 2



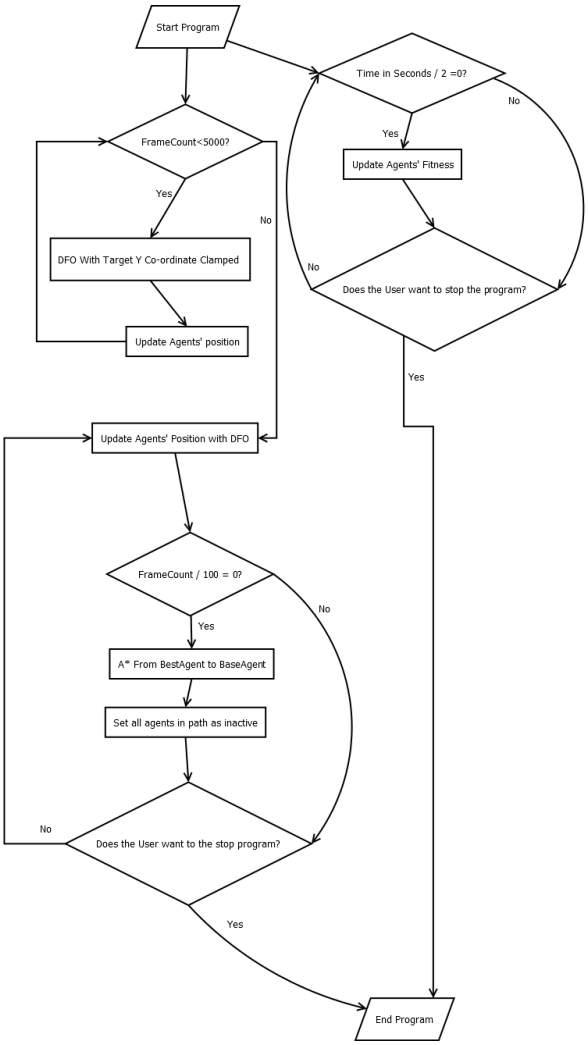
(c) End of the search phase

evaluation and update takes place, they begin to create a mound surrounding the base position. During this time, an A* path find normally takes place and, with the help of a visual guide, agents are selected to become the path to the best agent thus creating a structure.

If the A* algorithm is using the heuristic function which increases the cost of moving from one agent to another depending on the amount of agents in question. Then the path is seen being made from the base of the structure to the outside of the mound and then to the top of the mound around the outside.

Otherwise it would just create a path to the best agent going relatively straight upwards. After a certain height the very stable mound stops growing. Following this point, a more unstable structure of agents jumping around is created. However, this does not stop the A* from finding a path to the ground from the agent. When a path is valid it is seen to generally produce an entirely new branch from the current path. This creates some interesting visual. If the program is allowed to continually run, it is seen to have a large amount of inactive agents (agents who are in the best path at one point) within the

Figure 3. Hybrid A*DFO Flowchart



unstable section of the structure whilst the more stable mound section has a path of agents that tends to increase in thickness as it gets closer to the unstable section. Eventually the entire unstable section becomes inactive.[4].

The process through which the hybrid model functions is shown as a flowchart in Fig. 3, where the two arrows from the 'Start' indicate concurrency.

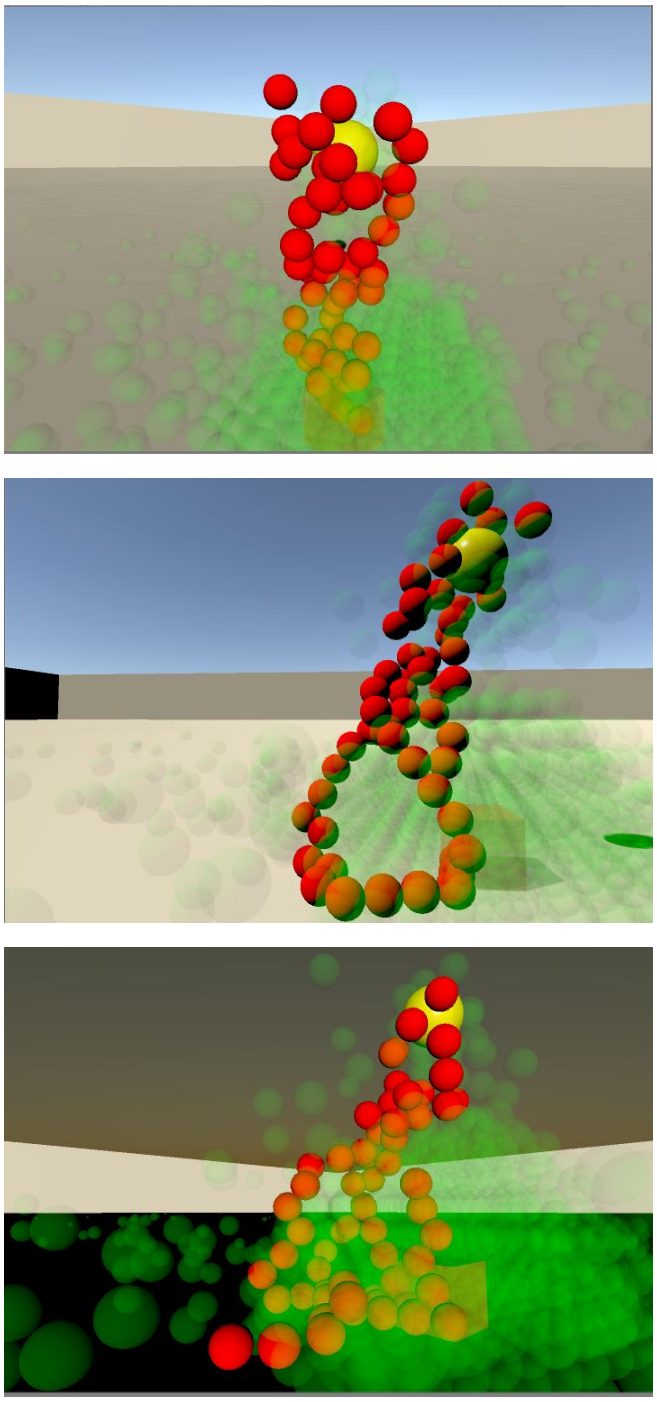
5 CONCLUSION

In conclusion, this work demonstrates the use of A* Path-finding and Dispersive Flies Optimisation in generating "organic" structures which are unique each time the program is run whilst exhibiting a 'natural' feel (see the images in Fig. 4). The natural aesthetics of these structures are shown through the different twists and turns that the path of inactive agents show, much like how plants such as vines grow. While there are many nature-inspired techniques which could be investigated in this work, the simplicity of DFO and its minimal parameter tuning, offered a attractive advantage over other algorithms leading to is deployment for this work. Other features of the algorithm is currently being investigated in context of the presented work.

Amongst the topics for future work is the introduction of multiple food sources in order to cater for the generation of more complex structure. The use of spring joints to join the inactive agents together could also vastly improve the stability of the overall structure but would also increase processing power required.

[4] A video of a sample run of the program can be accessed here: https://youtu.be/FyHFnvGkPRA

Figure 4. Sample unique structures



Another potential scenario for future use of the proposed method is when a user draws structures, shapes or designs in a 3D environment that are fed into the system, which will subsequently "replicate" the user's design in a more "organic and natural" way.

In summary, this work shows the viability of the method for generating unique, "naturally looking" structures. This entails when a user plays a game or watches a film, the environment they see could be formed in front of their eyes; this environment could dynamically change depending on the requirements. Investigating and optimising the speed of this process is a topic of an ongoing research.

Acknowledging the need to apply further changes to "firm up" the overall stability of the structures which would allow a wider range of potential for this method, the presented approach demonstrates its potential. Further experiments will be conducted to add to the complexity of the structures built while maintaining the simple principles behind the process of generating the structures.

REFERENCES

[1] Mohammad Majid Al-Rifaie, 'Dispersive flies optimisation', in *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*, pp. 529–538. IEEE, (2014).

[2] Mohammad Majid al-Rifaie, 'Perceived simplicity and complexity in nature', in *AISB 2017: Computational Architectures for Animal Cognition*, University of Bath, Bath, U.K., (2017). Accepted.

[3] Bernard Chazelle, 'Natural algorithms and influence systems', *Communications of the ACM*, **55**(12), 101–110, (2012).

[4] Kevin Cheng and Paul A Cairns, 'Behaviour, realism and immersion in games', in *CHI'05 extended abstracts on Human factors in computing systems*, pp. 1272–1275. ACM, (2005).

[5] Edsger W Dijkstra, 'A note on two problems in connexion with graphs', *Numerische mathematik*, **1**(1), 269–271, (1959).

[6] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1989.

[7] Peter E Hart, Nils J Nilsson, and Bertram Raphael, 'A formal basis for the heuristic determination of minimum cost paths', *IEEE transactions on Systems Science and Cybernetics*, **4**(2), 100–107, (1968).

[8] J. Kennedy and R. C. Eberhart, 'Particle swarm optimization', in *Proceedings of the IEEE International Conference on Neural Networks*, volume IV, pp. 1942–1948, Piscataway, NJ, (1995). IEEE Service Center.

[9] Rainer Storn and Kenneth Price, 'Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces', (1995). TR-95-012, [online]. Available: http://www.icsi.berkeley.edu/ storn/litera.html.

[10] David H Wolpert and William G Macready, 'No free lunch theorems for optimization', *IEEE transactions on evolutionary computation*, **1**(1), 67–82, (1997).

[11] W Zeng and RL Church, 'Finding shortest paths on real road networks: the case for a', *International journal of geographical information science*, **23**(4), 531–543, (2009).