Wisdom of artificial crowds algorithm for solving NP-hard problems

Roman V. Yampolskiy*

Duthie Center for Engineering, 215,
Speed School of Engineering,
University of Louisville,
Louisville, KY 40292, USA
E-mail: roman.yampolskiy@louisville.edu
*Corresponding author

Ahmed EL-Barkouky

Electrical and Computer Engineering,
Speed School of Engineering,
University of Louisville,
Louisville, KY 40292, USA
E-mail: arelba01@louisville.edu

Abstract: The paper describes a novel algorithm, inspired by the phenomenon of wisdom of crowds, for solving instances of NP-hard problems. The proposed approach achieves superior performance compared to the genetic algorithm-based approach and requires modest computational resources. On average, a 6%–9% improvement in quality of solutions has been observed.

Keywords: knapsack problem; KP; NP-complete; optimisation; travelling salesman problem; TSP; wisdom of artificial crowds; WoAC.

Reference to this paper should be made as follows: Yampolskiy, R.V. and EL-Barkouky, A. (2011) 'Wisdom of artificial crowds algorithm for solving NP-hard problems', *Int. J. Bio-Inspired Computation*, Vol. 3, No. 6, pp.358–369.

Biographical notes: Roman V. Yampolskiy received his PhD in Computer Science and Engineering from the University at Buffalo. He was a recipient of a four-year NSF Fellowship. Before his doctoral studies, he received his BS/MS (High Honours) combined degree in Computer Science from Rochester Institute of Technology. In 2008, he accepted an Assistant Professor position at the Speed School of Engineering, University of Louisville. His main areas of interest are behavioural biometrics, computer forensics, robot authentication and pattern recognition. He is the author of over 50 publications including multiple journal articles and books.

Ahmed EL-Barkouky is currently a PhD student at the ECE Department, University of Louisville, USA. He received his BSc degree from the Electrical Engineering Department, Ainshams University, Egypt in 2002 and his MSc degree from the Engineering Mathematics Department, Ainshams University, Egypt in 2009. His research domain is artificial intelligence and computer vision.

1 Introduction

NP-hard problems are believed to require exponential time for exact solutions (Karp, 1972). Since it is not feasible to practically solve such problems using classical computer architectures, optimal methods have been replaced with approximation algorithms that usually need polynomial time to provide reasonably good solutions (Rabanal et al., 2007).

Heuristic algorithms capable of addressing diverse problems are known as metaheuristics. Such algorithms are computational methods that attempt to find a close approximation to an optimal solution by iteratively trying to improve a candidate answer with regard to a given measure of quality. Metaheuristic algorithms do not make any assumptions about the problem being optimised and are capable of searching very large spaces of potential solutions. Unfortunately, metaheuristic algorithms are unlikely to arrive at an optimal solution for the majority of large real world problems. However, research continues to find asymptotically better metaheuristic algorithms for specific problems.

Most metaheuristic algorithms in optimisation and search have been modelled on processes observed in biological systems: genetic algorithms (GAs) (Goldberg, 1989), genetic programming (GP) (Koza, 1990), cellular automata (CA) (Wolfram, 2002), artificial neural networks (ANN), artificial immune system (AIS) (Farmer et al., 1986). Expanding on this trend of bio-inspired solutions a large number of animal or plant behaviour-based algorithms have been proposed in recent years: ant colony optimisation (ACO) (Dorigo et al., 2006), bee colony optimisation (BCO) (Pham et al., 2006), bacterial foraging optimisation (BFO) (Passino, 2002), glow-worm swarm optimisation (GSO) (Krishnanand and Ghose, 2005), firefly algorithm (FA) (Yang, 2009), cuckoo search (CS) (Yang and Deb, 2009), flocking birds (FB) (Reynolds, 1987), harmony search (HS) (Geem et al., 2001), monkey search (MS) (Mucherino and Seref, 2007) and invasive weed optimisation (IWO) (Mehrabian and Lucas, 2006). In this paper, we propose a novel algorithm modelled on the natural phenomenon known as the wisdom of crowds (WoC) (Surowiecki, 2004).

1.1 Wisdom of crowds

In his 1907 publication in Nature, Francis Galton reports on a crowd at a state fair, which was able to guess the weight of an ox better than any cattle expert (Galton, 1907). Intrigued by this phenomenon James Surowiecki in 2004 publishes: "The Wisdom of Crowds: Why the Many are Smarter than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations" (Surowiecki, 2004). In that book Surowiecki explains that "Under the right circumstances, groups are remarkably intelligent, and are often smarter than the smartest people in them. Groups do not need to be dominated by exceptionally intelligent people in order to be smart. Even if most of the people within a group are not especially well-informed or rational, it can still reach a collectively wise decision" (Surowiecki, 2004). Surowiecki further explains that for a crowd to be wise it has to satisfy four criteria:

- Cognitive diversity – individuals should have private information.
- Independence – opinions of individuals should be autonomously generated.
- Decentralisation – individual should be able to specialise and draw on local knowledge.
- Aggregation – a methodology should be available for arriving at a common answer.

Since the publication of Surowiecki's book, the WoC algorithm has been applied to many important problems both by social scientists (Yi et al., 2010) and computer scientists (Wagner et al., 2010; Mozer et al., 2008; Bai and Krishnamachari, 2010; Moore and Clayton, 2008; Shiratsuchi et al., 2006; Osorio and Whitney, 2005). However, all such research used real human beings either in person or via a network to obtain the crowd effect. In this work we propose a way to generate an artificial crowd of intelligent agents capable of coming up with independent solutions to a complex problem (Ashby and Yampolskiy, 2011).

2 Wisdom of artificial crowds

Wisdom of artificial crowds (WoAC) is a novel swarm-based nature-inspired metaheuristic algorithm for global optimisation (Ashby and Yampolskiy, 2011). WoAC is a post-processing algorithm in which independently-deciding artificial agents aggregate their individual solutions to arrive at an answer which is superior to all solutions present in the population. The algorithm is inspired by the natural phenomenon known as the WoC (Surowiecki, 2004). WoAC is designed to serve as a post-processing step for any swarm-based optimisation algorithm in which a population of intermediate solutions is produced, for example in this paper we will illustrate how WoAC can be applied to a standard GA.

The population of intermediate solutions to a problem is treated as a crowd of intelligent agents. For a specific problem an aggregation method is developed which allows individual solutions present in the population to be combined to produce a superior solution. The approach is somewhat related to ensemble learning (Opitz and Maclin, 1999) methods such as boosting or bootstrap aggregation (Melville and Mooney, 2003, 2004) in the context of classifier fusion in which decisions of independent classifiers are combined to produce a superior meta-algorithm. The main difference is that in ensembles "when combining multiple independent and diverse decisions each of which is at least more accurate than random guessing, random errors cancel each other out, correct decisions are reinforced" (Mooney, 2007), but in WoAC individual agents are not required to be more accurate than random guessing.

3 Solving TSP

Travelling salesman problem (TSP) has attracted a lot of attention over the years (Bellmore and Nemhauser, 1968; Dorigo and Gambardella, 1997; Burkard et al., 1998) because finding optimal paths is a requirement that frequently appears in real world applications and because it is a well defined benchmark problem to test newly developed heuristic approaches (Rabanal et al., 2007). TSP is a combinatorial optimisation problem and could be represented by the following model (Dorigo et al., 2006): $P = (S, \Omega, f)$ in which S is a search space defined over a finite set of discrete decision variables X_i, $i = 1, \ldots, n$; a set of constraints Ω; and an objective function f to be minimised.

TSP is a well known NP-hard problem meaning that an efficient algorithm for solving TSP will be an efficient algorithm for other NP-complete problems. In simple terms the problem could be stated as follows: a salesman is given a list of cities and a cost to travel between each pair of cities (or a list of city locations). The salesman must select a starting city and visit each city exactly once and return to the starting city. His problem is to find the route (also known as a Hamiltonian cycle) that will have the lowest cost. In this paper, we will use TSP as a non-trivial testing ground for our algorithm.

3.1 Dataset

Data for testing of our algorithm has been generated using a piece of software called Concorde (Cook, 2005). Concorde is a C programme written for solving the symmetric TSP and some related network optimisation problems and is freely available for academic use. The programme also allows one to generate new instances of the TSP of any size either with random distribution of nodes, or with predefined coordinates. For problems of moderate size, the software could be used to obtain optimal solutions to specific TSP instances. Below is an example of a Concorde data file with seven cities:

```
NAME: concorde7
TYPE: TSP
COMMENT: Generated by CCutil_writetsplib
COMMENT: Write called for by Concorde GUI
DIMENSION: 7
EDGE_WEIGHT_TYPE: EUC_2D
NODE_COORD_SECTION
1   87.951292 2.658162
2   33.466597 66.682943
3   91.778314 53.807184
4   20.526749 47.633290
5   9.006012 81.185339
6   20.032350 2.761925
7   77.181310 31.922361
```

3.2 Genetic algorithms

Inspired by evolution, GAs constitutes a powerful set of optimisation tools that have demonstrated good performance on a wide variety of problems including some classical NP-complete problems such as the TSP and multiple sequence alignment (MSA) (Yampolskiy, 2010). GAs search the solution space using a simulated 'Darwinian' evolution that favours survival of the fittest individuals. Survival of such population members is ensured by the fact that fitter individuals get a higher chance at reproduction and survive to the next generation in larger numbers (Goldberg, 1989).

GAs have been shown to solve linear and non-linear problems by exploring all regions of the state space and exponentially exploiting promising areas through standard genetic operators, eventually converging populations of candidate solutions to a single global optimum. However, some optimisation problems contain numerous local optima which are difficult to distinguish from the global maximum and therefore result in sub-optimal solutions. As a consequence, several population diversity mechanisms have been proposed to delay or counteract the convergence of the population by maintaining a diverse population of members throughout its search.

A typical GA follows the following steps (Yampolskiy et al., 2004):

1 a population of N possible solutions is created

2 the fitness value of each individual is determined

3 repeat the following steps $N/2$ times to create the next generation

   a choose two parents using tournament selection

   b with probability p_c, crossover the parents to create two children; otherwise simply pass parents to the next generation

   c with probability p_m for each child, mutate that child

   d place the two new children into the next generation.

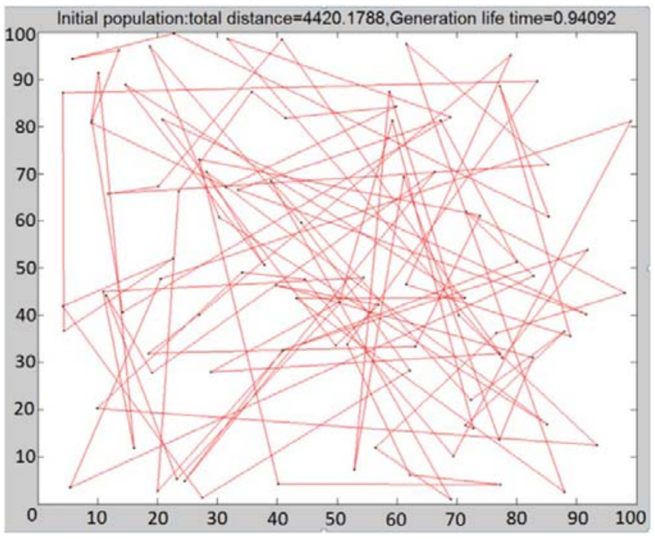
4 repeat new generation creation until a satisfactory solution is found or the search time is exhausted.

3.3 Implemented GA for solving TSP

The GA used in this project has four main operations: initialisation, crossover, mutation and cloning.

To understand the effect of initialisation, Figure 1 shows a totally random route that can represent one of the chromosomes in the initial population. The total distance is very long which suggests that starting from a totally random solution is not the best way to achieve a good result at the end.

Figure 1 A chromosome from a random initial population (see online version for colours)



In order to improve fitness of the starting population, we have undertaken some preprocessing steps. Figures 2 and 3 show two initialisations that use the polar coordinates to arrange the cities in the increasing direction of theta. This allows us to make use of a special property of the polar coordinates, specifically that they span the whole domain without any intersection. Consequently, the chromosome looks well organised but it has no diversity to construct a population of say 50 chromosomes, which means we must have some randomness in the initial population.

Figure 2 Cities arranged according to the value of theta in ascending order from $-\pi$ to π (see online version for colours)

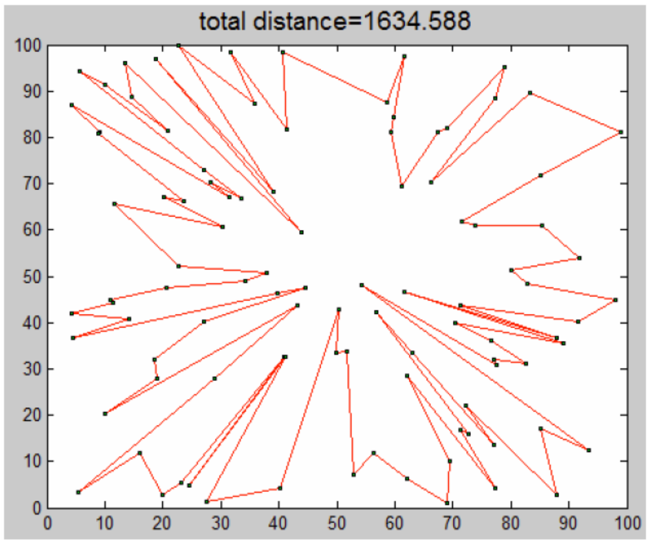
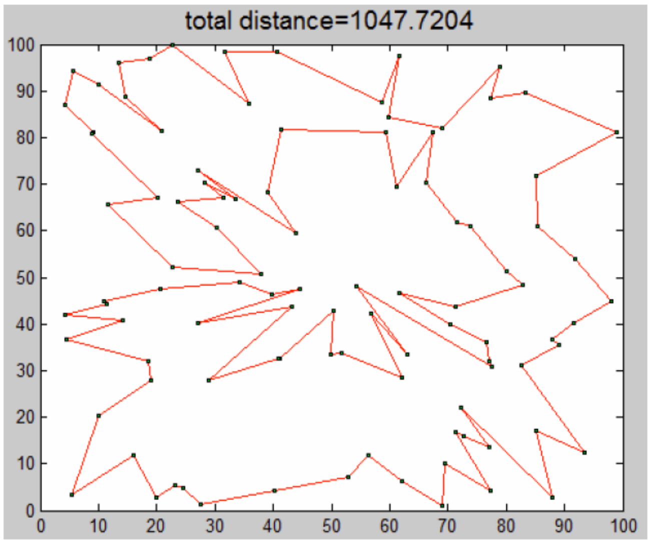


Figure 3 Cities arranged into two groups according to their 'r' values (see online version for colours)

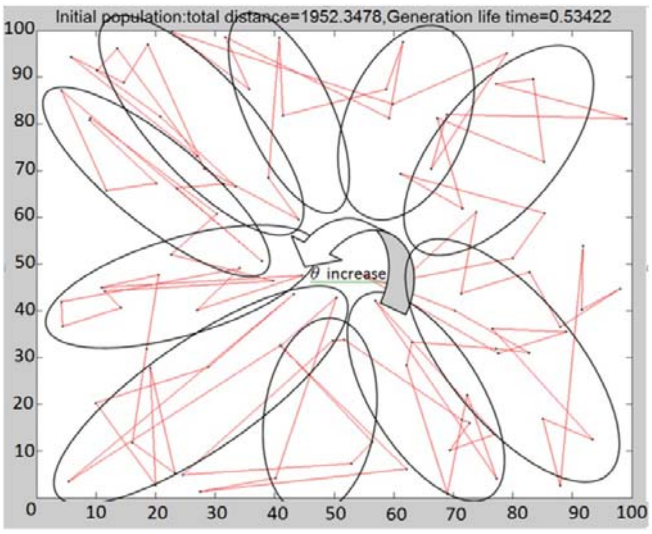


To use polar coordinates we take the following steps:

- shift Cartesian coordinates for all cities such that the origin is in the middle
- transform all cities coordinates from Cartesian to polar
- arrange cities according to the value of theta in ascending order from $-\pi$ to π
- Figure 2 shows the result of the previous steps
- if we split the cities into two groups according to their '*r*' values then arrange cities in each group according to theta we will achieve the result in Figure 3.

This allows us to initialise the search in a way that is less random but still contains enough diversity in the initial population. This makes the chromosomes able to produce diverse children that have better characteristics. To do this, we make use of the polar coordinates by arranging the cities into ten regions each of 36°. Inside each region, the cities order is random. So we keep randomness, but in an organised way. This ensures that the cities in each group are close to each other but inside any group, they are connected randomly. This is illustrated in Figure 4.

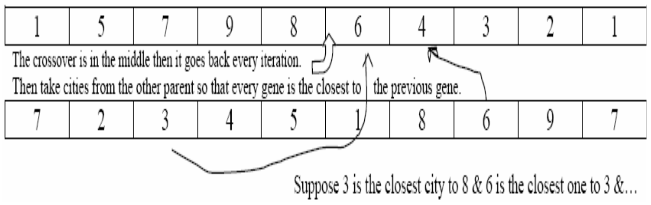
Figure 4 A chromosome arranged in ten groups in the direction of ascending increase of theta but inside each group connected randomly (see online version for colours)



The crossover is done using two different operators.

The first one is a one-point crossover that changes the position of the crossover point every iteration starting from being in the middle of the chromosome and going back to the beginning of the chromosome. Then it takes from the other parent the remaining cities in order, starting from the nearest city to the last city, before crossover and choosing every time the city closest to the previous one.

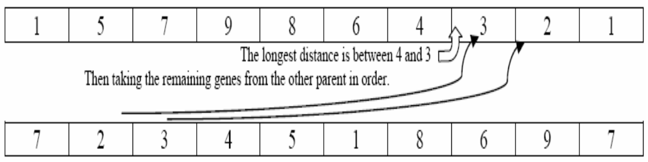
Figure 5 Random crossover point


1 5 7 9 8 6 4 3 2 1
The crossover is in the middle then it goes back every iteration.
Then take cities from the other parent so that every gene is the closest to the previous gene.
7 2 3 4 5 1 8 6 9 7
Suppose 3 is the closest city to 8 & 6 is the closest one to 3 &…

The second one is also a one-point crossover but it selects the longest distance between two cities in the chromosome as a breaking point and then it takes the rest of the chromosome from the other parent in the order of appearance.

Both crossover operators produce two children from every two parents resulting in the same number of chromosomes in each population. To ensure that the algorithm progresses towards a better solution, the best ten parents are cloned into the new population instead of the ten worst children. For the mutation operation we swap two cities in such a way as to remove one path crossing (intersection). The mutation operator was designed to remove one intersection at a time but it does this only if it will enhance the result by making the final path shorter. That is why at the end, some intersections will remain, since when the mutation function tried to remove them the resulting distance was longer.

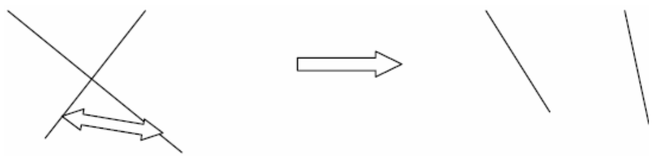
Figure 6 Crossover at the longest distance between two cities


1 5 7 9 8 6 4 3 2 1
The longest distance is between 4 and 3
Then taking the remaining genes from the other parent in order.
7 2 3 4 5 1 8 6 9 7

3.4 Post processing

A function was designed to check every two segments in a chromosome. If they are intersected, it removes the intersection by swapping two points as shown in Figure 7. This function was added as an optional post processing step that might enhance the solution. The function does not check if the resulting solution is better or not. It simply removes any intersections from the resulting solution.

Figure 7 Mutation operation removes intersections



3.5 WoAC aggregation method

Building on the work of Yi et al. (2010) who used a group of volunteers to solve instances of TSP and aggregated their answers, we have developed an automatic aggregation method which takes individual solutions and produces a common solution which reflects frequent local structures of individual answers. The approach is based on the belief that good local connections between nodes will tend to be present in many solutions, while bad local structures will be relatively rare. After constructing an agreement matrix, Yi et al. (2010) applied a non-linear monotonic transformation function in order to transform agreements between answers into costs. They focused on the function:

$$c_{ij} = 1 - I_{a_{ij}}^{-1}(b_1, b_2), \tag{1}$$

where

$$I_{a_{ij}}^{-1}(b_1, b_2) \tag{2}$$

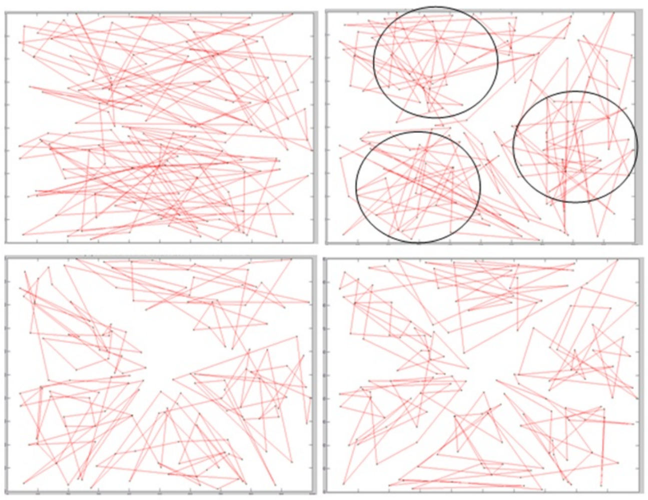
is the inverse regularised beta function with parameters b_1 and b_2 both taking a value of at least 1 Yi et al. (2010).

In our implementation of the aggregation function, we continue working with agreements between local components of the solutions.

- *initialising* the crowds must be done in such a way that they have diversity
- *aggregating* population of solutions to produce a better solution.

To achieve diversity in the crowds we use polar coordinates in initialising the GA used for making the crowds. To illustrate that, Figure 8 shows four different initialisations. The cities are arranged in ascending order of theta with the origin in the middle of the figure, then divided into 2, 3, 5 and 6 groups. After that, the GA is applied to each group five times to initiate 20 different solutions. These solutions will be the crowds and in this way they will contain the required diversity.

Figure 8 Four different initialisations of the GA (see online version for colours)



To aggregate these solutions we do the following:

- Prepare a list containing in every row a city and the two cities connected to it. For example if the third row contains 65 and 34 that means that the third city is connected to city 65 and 34.
- If 90% of the crowd agreed on a link between two cities then we will keep this link.
- Finally, we execute a greedy algorithm which removes all of the remaining intersections as a post processing step.

The results of the aggregation process are illustrated in Figures 9 and 10. Figure 9 shows the common segments that were repeated in 90% of the solutions. Figure 10 shows the final result after connecting these segments and removing intersections.

Figure 9 Segments common to 90% of crowd members (see online version for colours)

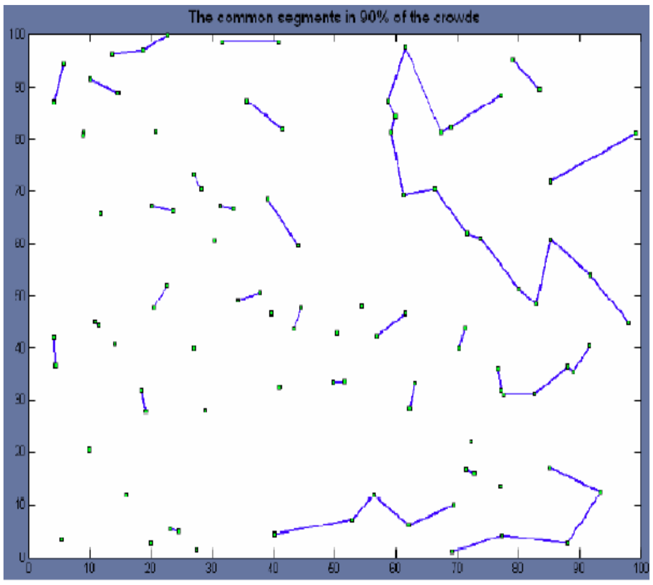
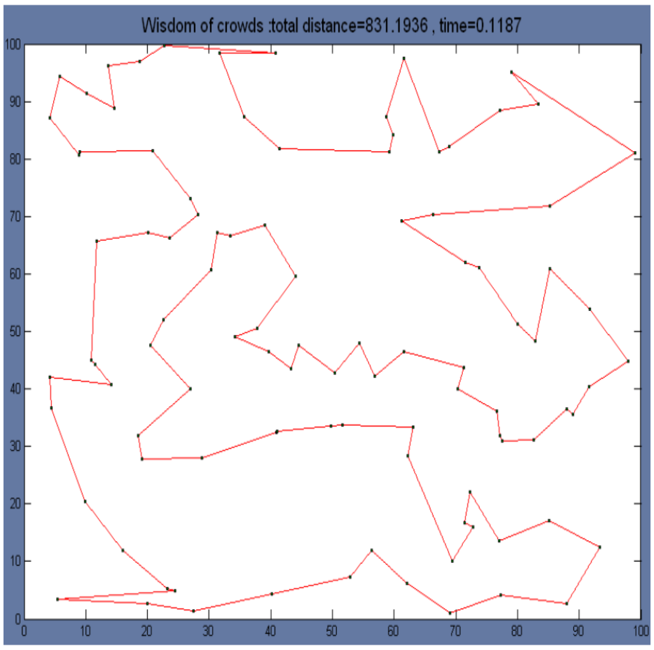


Figure 10 Solution based on aggregation of segments and intersection removal (see online version for colours)



3.6 *TSP-experimental results*

The developed software was tested on problems with 100, 150 and 200 cities. Table 1 shows achieved results for the GA and for WoAC. Both minimum and average performance is reported for the GA. WoAC algorithm outperformed GA on all problems on average by between 6% and 9%.

Table 1 Performance of GA vs. WoAC

# of Cities	*GA min*	*GA average*	*WoAC*	*% Improved*
100	894.1571	914.1940	831.1936	9%
150	1,077.9	1,093.1	1,025.4681	6%
200	1,233.7	1,256.7	1,178.4072	6%

Figure 11 Performance of GA (top) and WoAC on a 100 city problem (see online version for colours)

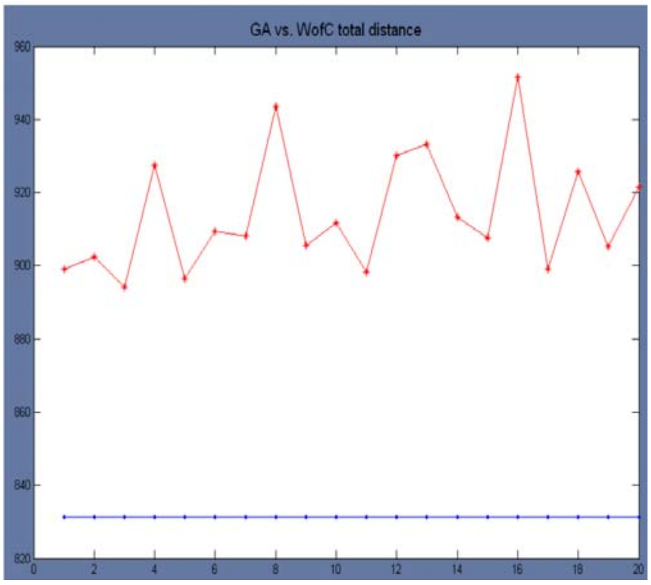
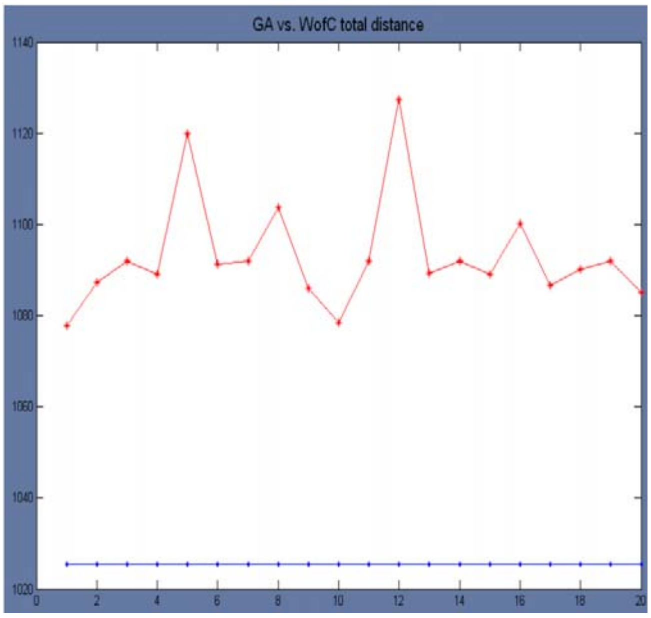
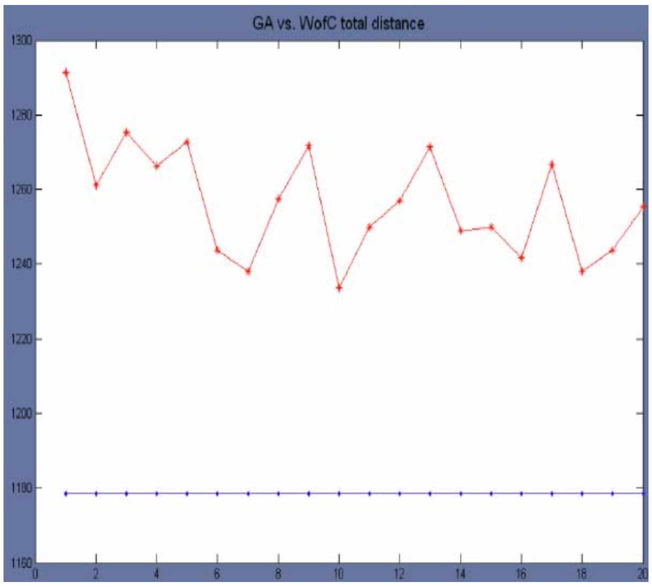


Figure 12 Performance of GA (top) and WoAC on a 150 city problem (see online version for colours)



Figures 11 to 13 display in red the distance of each of the GA solutions while the blue line is the solution after applying the WoAC represented as a horizontal line. It is clear that it enhanced the GA solutions and produced a solution that is better than all the 20 solutions of the GA. This was achieved up to the case of 200 cities.

Figure 13 Performance of GA (top) and WoAC on a 200 city problem (see online version for colours)



4 Solving the KP

Knapsack problem (KP) is a classical NP-complete problem in the field of combinatorial optimisation (Guo et al., 2010). This problem has very important applications in financial and industrial domains, in combinatorics, complexity theory, cryptography and applied mathematics. KP can be used to model resource distribution, investment decision-making, items shipment, budget control and project selection, and it often represents a part of a larger problems.

In KP, a set of items each with a weight and a value is given and the objective is to determine which items to include in a collection so that the total weight is less than a given limit and the total value is as large as possible. The problem derives its name from the situation of someone who is constrained by a fixed-size knapsack and needs to fill it with the most useful items. The decision form of the KP is the question "can a value of at least V be achieved without exceeding the weight W?"

Mathematically the problem consists of a knapsack that has positive integer weight (capacity) *W*. There are *N* distinct items that may potentially be placed in the knapsack. Item *i* has a positive integer weight W_i and positive integer value V_i. In addition, there are C_i copies of item *i* available, where quantity C_i is a positive integer satisfying $1 \le C_i \le \infty$.

Let X_i determines how many copies of item *i* are to be placed into the knapsack. The goal is to maximise:

$$\sum_{i=1}^{N} v_i x_i \tag{3}$$

Subject to the constraint

$$\sum_{i=1}^{N} w_i x_i \le w \tag{4}$$

If one or more of the c_i is infinite, the KP is unbounded; otherwise, the KP is bounded. For the bounded KP $x_i \in \{0, 1, \dots, C_i\}$. The 0–1 KP is a special case where $x_i \in \{0, 1\}$ (Hristakeva and Shrestha, 2004). In this paper, the work is done on the bounded 0–1 KP, where we cannot have more than one copy of an item in the knapsack.

To illustrate the problem in more details let's consider the case of three items A, B and C with weights 5, 9 and 15 and values 4, 6 and 8 respectively. The knapsack capacity is 20. Because this is a small problem we can brute force the solution by checking all the possible solutions as shown in the Table 2.

Table 2 Simple case of the 0-1 KP

A B C	*Weight*	*Value*
0 0 0	0	0
0 0 1	15	8
0 1 0	9	6
0 1 1	9 + 15 = 24 > 20 rejected	6 + 8 = 14
1 0 0	5	4
1 0 1	5 + 15 = 20	4 + 8 = 12 best
1 1 0	5 + 9 = 14	4 + 6 = 10
1 1 1	5 + 9 + 15 = 29 > 20 rejected	4 + 6 + 8 = 18

It is clear that if we have a 100 items, we should consider 2^{100} cases and calculate the weight for all of them then calculate the value for those satisfying the constraint and finally select the highest value. We can see from this discussion that brute forcing is not suitable for such a problem so in this paper we will consider finding an approximate solution by using a GA and improving results via a novel postprocessing algorithm we call the WoAC.

4.1 Prior work

There are two kinds of algorithms for KP. The first kind consists of precise algorithms such as dynamic programming, backtracking, and branch and bound, and the other kind is the set of approximate algorithms including greedy method and Lagrange method. The time complexity for solving the KP increases rapidly as the problem scale grows. Specifically, the time complexity is $O(2^n)$ in the worst case. So, it is important to design an effective approximation algorithm for solving the KP (Qiao et al., 2008).

Several works in literature used GAs combined with other methods to solve the 0-1 KP. In Zhao et al. (2009), a

GA based on Greedy strategy is introduced. It begins with analysis of three kinds of commonly used greedy algorithms to solve 0–1 KP. Combining this with the basic principles of GAs, the achieved improvement lies in the establishment of the original population using greedy strategy. In Guo et al. (2010), a solution of the problem by Chaotic GA is presented. It introduces Chaos idea into GA, adding the disturbance to help find better solutions compared to the traditional GA. The work in Zhao et al. (2008) combines multi-agent theory and master-slave model parallel GA (MSM-PGA) together into one union. This union solves the 0–1 KP via coordination between many Agents inside the union.

GA is not the only approximation approach for solving the KP. In Zhang and Wei (2008), particle swarm optimisation (PSO) algorithm is used. This is a bio-inspired optimisation algorithm based on group intelligence. In Qiao et al. (2008), the authors combine the mobile agent technology with the traditional parallel algorithm which enables changing the parallel process handled in a parallel computer to the process performed by several ordinary computers, and by doing so avoid the restrictions of the limited computational resources. In Jun and Jian (2009), a discrete binary version of differential evolution (DBDE) was employed, where each component of a mutated vector component changes with the differential probability and will take on a zero or one value.

A schema-guiding evolutionary algorithm (SGEA) is proposed in Liu and Liu (2009). It improves the diversity of the population and the local and global search power. The work in Martello et al. (1999) presents a combination of dynamic programming and strong bounds, in addition valid inequalities are generated and surrogate relaxed, and a new initial core problem is adopted.

In this paper, a GA is developed and the results are enhanced using the postprocessing algorithm we call the WoAC. The original WoC concept was introduced by James Surowiecki in 2004 (Surowiecki, 2004). It highlights the aggregation of information in groups, resulting in decisions that are often better than could have been made by any single member of the group (Narasimhan et al., 2010; Osorio and Whitney, 2005; Kostakos, 2009).

4.2 The 0–1 KP using GA and WoAC

To solve the KP using GA the chromosome length is set equal to the number of items and each gene will represent one item and take the value 0 if we will not put that item in the knapsack or 1 if we will put it. The population starts randomly with chromosomes that do not necessarily satisfy the capacity constraint.

Two types of crossover are tested. The one-point crossover selects randomly a point and does the crossover after that point and the two-point crossover selects two points randomly and does the crossover between them as follows:

Parent 1: **0 1 0** | **1 1 1 0 0**   **0 1 0** | **1 1 1** | **0 0**

Parent 2: *1 0 1* | *1 0 1 1 0*   *1 0 1* | *1 0 1* | *1 0*

Child 1: **0 1 0** | *1 0 1 1 0*   **0 1 0** | *1 0 1* | **0 0**

Child 2: *1 0 1* | **1 1 1 0 0**   *1 0 1* | **1 1 1** | *1 0*

One point   Two-point

To make this clear one of the parents is represented in the *bold* font and the other is in *italic* and the children after the crossover can be seen by tracking the bold and italic genes.

The mutation is done on a small percentage of the children to ensure that the algorithm does not get stuck at a local maximum point. The mutation simply picks randomly a chromosome then picks randomly a gene in that chromosome and reverses its value as follows:

1 0 1 0 1 ***1*** 0 1 $\rightarrow$ mutation $\rightarrow$ 1 0 1 0 1 ***0*** 0 1

The cloning is used to ensure that every new generation has the best value which is equal to the best in the previous generation if not higher. This is done through keeping the best 10% of the parents. The ratio will vary because we insert these best parents in place of the children that exceed the capacity, so if in a generation: less than 10% exceeds the capacity this means that this generation has good children and will keep them so fewer parents are cloned. The minimum cloning is to keep the best parent, and so to ensure that the best value in the new generation will not be less than the previous generation.

The WoAC is used after that to refine the results. To initiate a crowd that has diversity of opinions, we run the GA 100 times for the one-point crossover and another 100 for the two-point crossover. In this way we have 200 solutions that came from two different ‘cultures’. The method used to aggregate opinions can be summarised as follows: If 80% of the crowd set an item to zero, the item is not included. If 55% of the crowd set an item to one it is included in the knapsack. Doing this will lead to total weight less than the allowed capacity, so we use a greedy algorithm to fill the rest of the knapsack with higher value items.

An important thing to note is that different percentages were used in the case of aggregating opinions in the zero and one cases. It was found that accepting the crowds opinion in the case of zero (not taking the item) should be more accurate than the case of one (taking the item). The disadvantage in using the WoC in this way is that it took a lot of time to initiate the crowds, but this is important to ensure diversity of opinions.

4.3 Experimental results

4.3.1 Data

Generating instants of the 0–1 KP to test the algorithm was not a hard task since we simply need N items with different weights and values. First, a vector of length N with values taken randomly between *1* and *1,000* is created to represent the weights of the N items arranged in ascending order.

Then, the values are also arranged in ascending order from *1* to *200* which means items with higher weight will have higher values. The capacity of the knapsack is considered as *1/4* of the sum of all of the weights rounded to the nearest thousand by the floor function. The problem generated in this way will consist of four variables:

- N: the number of the items (will be the length of the chromosome)
- weights: a $1 \times N$ vector arranged in ascending order represents weights of items
- values: a $1 \times N$ vector represents value of items
- capacity: 1/4 of the sum of all of the weights rounded to the nearest thousand.

For example an instance with *nine* items of weights and values will be:

Weights = [50 200 357 411 473 556 670 910 950]

Values = [20 40 60 80 100 120 140 160 180]

The solution of the problem takes the form of a vector of length N that has a value of 1 if this item is taken into the knapsack or 0 if it is not taken, for example:

Solution = [1 0 0 1 0 1 0 1 1] with total value = 56

4.3.2 Results

The code was written using MATLAB 7.8.0 (R2009a) and was tested on a PC with a processor Intel(R) Pentium(R) 4 CPU 3.00 GHz and installed memory (RAM) 4.00 GB (3.25 GB usable). It was tested on the previously illustrated problem with number of items $N = 100$. To visualise the results a 10×10 matrices are plotted which illustrate the items with the value of the item written inside each cell and its weight written under the cell. If we will put an item in the knapsack then its colour is green and if we will not take it, its colour is red. In this way we can see how the GA evolves from one generation to another.

In Figure 14, we can see the best chromosome in the initial population which had a total value of 2,362 and utilised 12,862 unit of weight out of the 13,000 possible. Figures 15 and 16 show the best chromosome in the population after 2000 generations using one-point crossover and two-point crossover respectively. For the one-point crossover the value was 2,576 and in the two-point crossover it was 2,556. In both cases the whole weight of 13,000 was utilised. Figure 17 shows the result of applying the WoAC to 200 chromosomes obtained from running the GA 200 times half of them with one-point crossover and the other half with two-point crossover. The total value increased to 2,602 achieving 1% increase over the one-point crossover and 1.8% increase over the two-point crossover. To illustrate more, Figure 17 shows green boxes with red frame which denote items that were not taken in GA but the WoAC decided to take them and the red boxes with green frames representing the items the WoAC removed from the knapsack.

It is clear from the results that the GA was suitable to the problem because its implementation was very simple which makes a population of size 100 processed in just 0.005 seconds. This enabled doing a large number of generations. The evolution curves for 2,000 generation and 20,000 generation are displayed in Figures 18 and 19 respectively. The WoAC was suitable for postprocessing and allowed us to obtain better results compared to running GA for 20,000 generations. The disadvantage of the WoAC approach is that it takes a lot of time to produce the initial crowd, specifically we need to run 2,000 generations 200 times.

Figure 14 Initial population (see online version for colours)

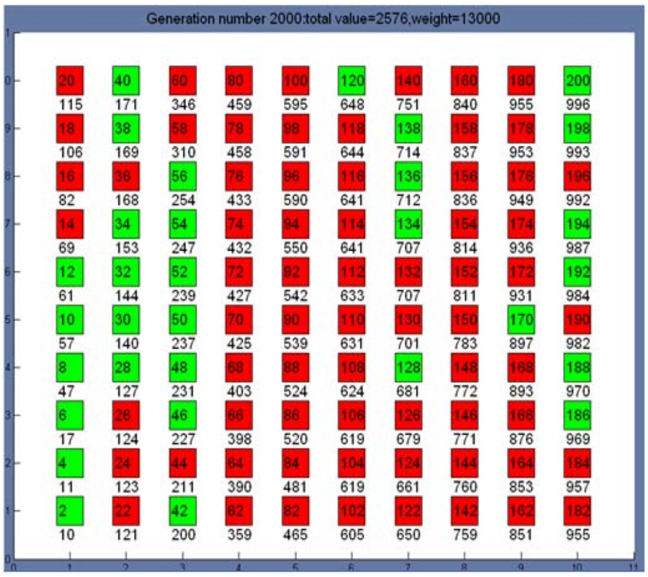


Figure 15 After 2,000 generation of GA using one-point crossover (see online version for colours)

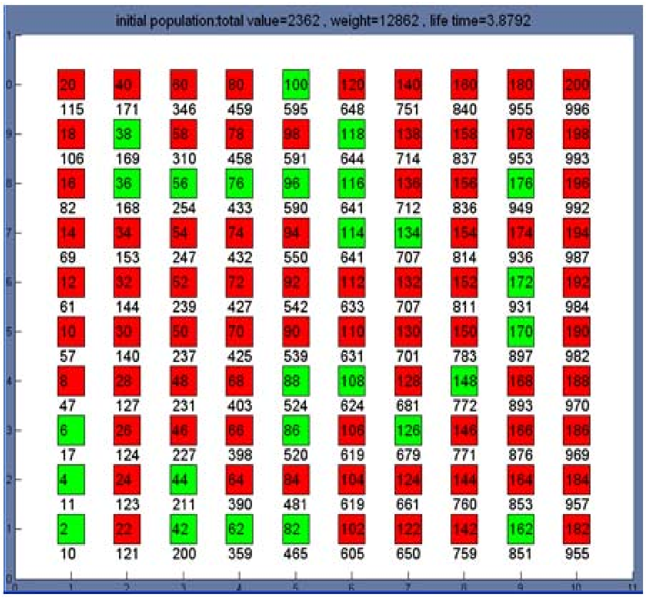


Figure 16 After 2,000 generations of two-point crossover (see online version for colours)

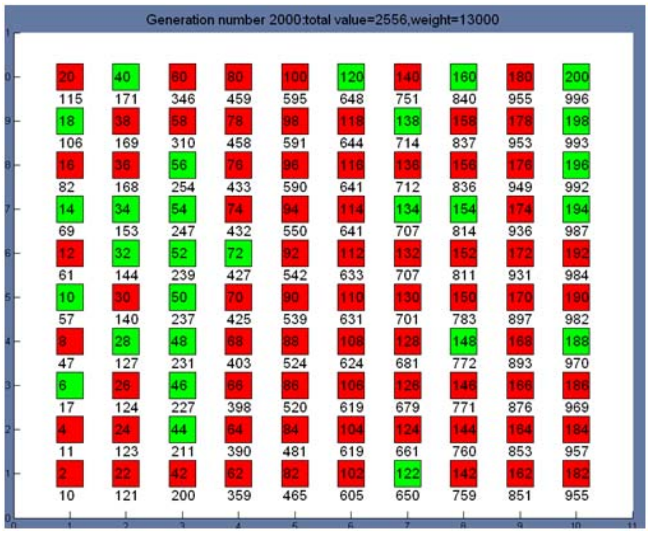


Figure 17 Applying WoAC (see online version for colours)

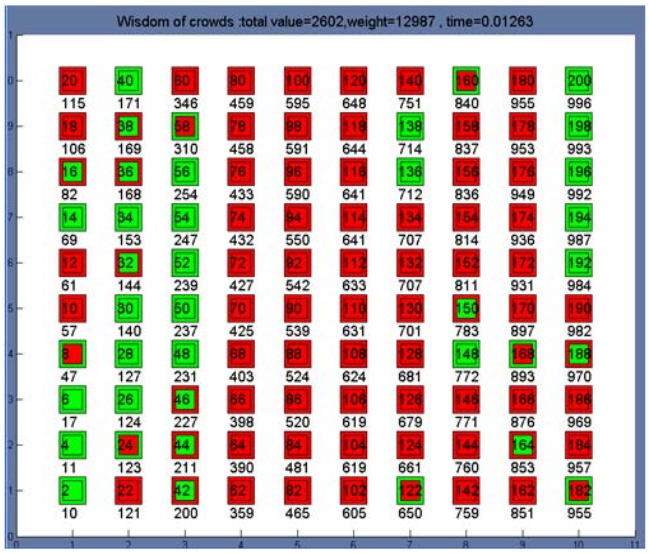


Figure 18 Evolution curve for 2,000 generations of GA only (see online version for colours)

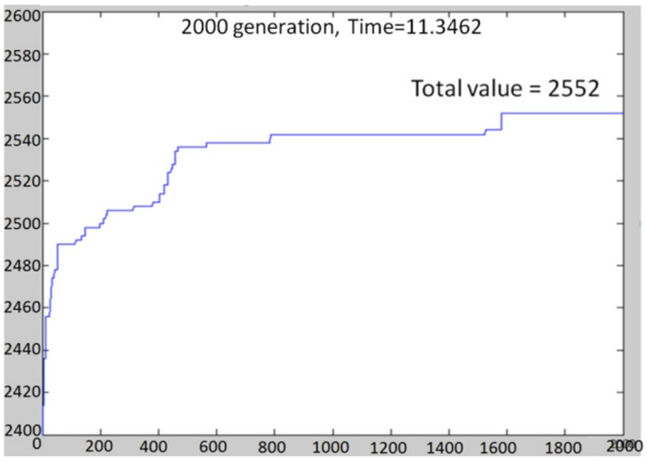
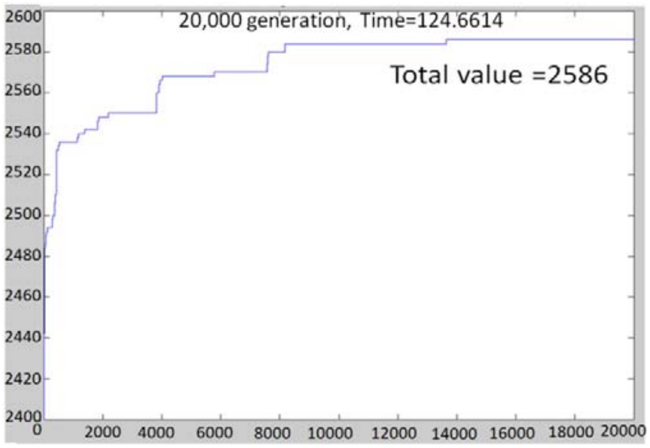


Figure 19 Evolution curve for 20,000 generations of GA only (see online version for colours)



5 Conclusions

We have presented a novel swarm-based nature-inspired metaheuristic algorithm for global optimisation. In many cases WoAC outperformed even the best solutions produced by the GA. As the datasets increase in size, the GA performs worse, but this allows more room for improvement for WoAC. WoAC is a postprocessing algorithm with running time in milliseconds which is negligible in comparison to the algorithm it attempts to improve, genetic search. While, WoAC does not always produce a superior solution, in cases where it fails it can be simply ignored since the GA itself provides a better solution in such cases. Consequently, WoAC is computationally efficient and can only improve the quality of solutions, never hurting the overall outcome.

In the future, we plan on conducting additional experiments aimed at improving overall performance of the WoAC algorithm. In particular we are going to investigate how WoAC could be combined with non-GA, swarm-based approaches such as ACO (Dorigo et al., 2006), BCO (Pham et al., 2006), (BFO) (Passino, 2002), or (GSO) (Krishnanand and Ghose, 2005). Special attention should be given to investigating better aggregation rules and optimal ways of achieving diversity in the populations. An important question to ask, deals with an optimal percentage of the population to be used in the crowd. In other words, should the whole population be used or is it better to select a sub-group of 'experts'.

References

Ashby, L.H. and Yampolskiy, R.V. (2011) 'Genetic algorithm and wisdom of artificial crowds algorithm applied to light up', *16th International Conference on Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational & Serious Games*, Louisville, KY, USA, 27–30 July.

Bai, F. and Krishnamachari, B. (2010) 'Exploiting the wisdom of the crowd: localized, distributed information-centric VANETs', *Communications Magazine, IEEE*, May, Vol. 48, No. 5.

Bellmore, M. and Nemhauser, G.L. (1968) 'The traveling salesman problem: a survey', *Operations Research*, May–June, Vol. 16, No. 3, pp.538–558.

Burkard, R.E., Deineko, V.G., Dal, R.V., Veen, J.A.A.V.D. and Woeginger, G.J. (1998) 'Well-solvable special cases of the traveling salesman problem: a survey', *SIAM Review*, Vol. 40, No. 3, pp.496–546.

Cook, W. (2005) 'Concorde TSP solver', available at: http://www.tsp.gatech.edu/concorde/index.html (accessed on 4 December 2010).

Dorigo, M. and Gambardella, L.M. (1997) 'Ant colonies for the traveling salesman problem', *Biosystems*, July, Vol. 43, No. 2, pp.73–81.

Dorigo, M., Birattari, M. and Stutzle, T. (2006) 'Ant colony optimization: artificial ants as a computational intelligence technique', *IEEE Computational Intelligence Magazine*, November, Vol. 1, No. 4, pp.28–39.

Farmer, J.D., Packard, N. and Perelson, A. (1986) 'The immune system, adaptation and machine learning', *Physica D*, Vol. 2, Nos. 1–3, pp.187–204.

Galton, F. (1907) 'Vox Populi', *Nature*, Vol. 75, No. 1949, pp.450–451.

Geem, Z.W., Kim, J.H. and Loganathan, G.V. (2001) 'A new heuristic optimization algorithm: harmony search', *Simulation*, February, Vol. 76, No. 2, pp.60–68.

Goldberg, D.E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Pub. Co., Boston, MA, USA.

Guo, X-H., He, D-X. and Liu, G-Q. (2010) 'An algorithm based on chaotic genetic algorithm for 0-1 knapsack problem', *International Conference on Biomedical Engineering and Computer Science (ICBECS)*, Wuhan, China, 23–25 April.

Hristakeva, M. and Shrestha, D. (2004) 'Solving the 0-1 knapsack problem with genetic algorithms', *Science & Math Undergraduate Research Symposium*, Simpson College, Indianola, Iowa.

Jun, S. and Jian, L. (2009) 'Solving 0-1 knapsack problems via a hybrid differential evolution', *Third International Symposium on Intelligent Information Technology Application (IITA 2009)*, Nanchang, China, 21–22 November.

Karp, R.M. (1972) 'Reducibility among combinatorial problems', in Miller, R.E. and Thatcher, J.W. (Eds.): *Complexity of Computer Computations*, Plenum, New York.

Kostakos, V. (2009) 'Is the crowd's wisdom biased? A quantitative analysis of three online communities international conference on computational science and engineering', *CSE '09*, Vancouver, Canada, 29–31 August.

Koza, J.R. (1990) 'Genetic programming: a paradigm for genetically breeding populations of computer programs to solve problems', Technical Report No. STAN-CS-90-1314, Stanford University. Stanford, California.

Krishnanand, K.N. and Ghose, D. (2005) 'Detection of multiple source locations using a glow-worm metaphor with applications to collective robotics', *IEEE Swarm Intelligence Symposium (SIS'05)*, Pasadena, California, 8–10 June.

Liu, Y. and Liu, C. (2009) 'A schema-guiding evolutionary algorithm for 0-1 knapsack problem', *International Association of Computer Science and Information Technology – Spring Conference (IACSITSC '09)*, Singapore, Singapore, 17–20 April.

Martello, S., Pisinger, D. and Toth, P. (1999) 'Dynamic programming and strong bounds for the 0-1 knapsack problem', *Management Science*, March, Vol. 45, No. 3.

Mehrabian, A.R. and Lucas, C. (2006) 'A novel numerical optimization algorithm inspired from weed colonization', *Ecological Informatics*, December, Vol. 1, No. 4, pp.355–366.

Melville, P. and Mooney, R.J. (2003) 'Constructing diverse classifier ensembles using artificial training examples', *18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, Acapulco, Mexico, August.

Melville, P. and Mooney, R.J. (2004) 'Diverse ensembles for active learning', *21st International Conference on Machine Learning (ICML'04)*. Banff, Canada, July.

Mooney, R.J. (2007) 'Machine learning: ensembles', available at http://www.cs.utexas.edu/~mooney/cs391L/slides/ensembles.ppt (accessed on 8 January 2011).

Moore, T. and Clayton, R. (2008) 'Evaluating the wisdom of crowds in assessing phishing websites', *Lecture Notes in Computer Science*, Vol. 5143, pp.16–30.

Mozer, M.C., Pashler, H. and Homaei, H. (2008) 'Optimal predictions in everyday cognition: the wisdom of individuals or crowds?', *Cognitive Science*, October, Vol. 32, No. 7, pp.1133–1147.

Mucherino, A. and Seref, O. (2007) 'Monkey search: a novel metaheuristic search for global optimization', *AIP Conference on Data Mining, Systems Analysis and Optimization in Biomedicine*, Gainesville, FL, 28–30 March.

Narasimhan, N., Wodka, J., Wong, P. and Vasudevan, V. (2010) 'TV answers – using the wisdom of crowds to facilitate searches with rich media context', *7th IEEE Consumer Communications and Networking Conference (CCNC)*, Las Vegas, Nevada, USA, 9–12 January.

Opitz, D. and Maclin, R. (1999) 'Popular ensemble methods: an empirical study', *Journal of Artificial Intelligence Research*, Vol. 11, pp.169–198.

Osorio, F.C.C. and Whitney, J. (2005) 'Trust, the "wisdom of crowds", and societal norms: the creation, maintenance, and reasoning about trust in peer networks', *Workshop of the 1st International Conference on Security and Privacy for Emerging Areas in Communication Networks*, 5–9 September.

Passino, K.M. (2002) 'Biomimicry of bacterial foraging for distributed optimization and control', *Control Systems Magazine, IEEE*, June, Vol. 22, No. 3, pp.52–67.

Pham, D.T., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S. and Zaidi, M. (2006) 'The bees algorithm – a novel tool for complex optimisation problems', *Virtual International Conference on Intelligent Production Machines and Systems (IPROMS'06),* Web Based, 13–14 July.

Qiao, S., Wang, S., Lin, Y. and Zhao, L. (2008) 'A distributed algorithm for 0-1 knapsack problem based on mobile agent', *Eighth International Conference on Intelligent Systems Design and Applications (ISDA'08)*, Kaohsiung City, Taiwan, 26–28 November.

Rabanal, P., Rodriguez, I. and Rubio, F. (2007) 'Using river formation dynamics to design heuristic algorithms', *Lecture Notes in Computer Science*, Vol. 4618, pp.163–177.

Reynolds, C.W. (1987) 'Flocks, herds, and schools: a distributed behavioral model', *14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'87)*, Anaheim, CA, 27–31 July.

Shiratsuchi, K., Yoshii, S. and Furukawa, M. (2006) 'Finding unknown interests utilizing the wisdom of crowds in a social bookmark service', *IEEE/WIC/ACM International Conference on Intelligence and Intelligent Agent Technology*, Hong Kong, December.

Surowiecki, J. (2004) *The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations*, Little, Brown.

Wagner, C., Schneider, C., Zhao, S. and Chen, H. (2010) 'The wisdom of reluctant crowds', *43rd Hawaii International Conference on System Sciences (HICSS'10)*, Honolulu, HI, 5–8 January.

Wolfram, S. (2002) *A New Kind of Science*, 14 May, Wolfram Media, Inc., Canada.

Yampolskiy, R., Anderson, P., Arney, J., Misic, V. and Clarke, T. (2004) 'Printer model integrating genetic algorithm for improvement of halftone patterns', *Western New York Image Processing Workshop (WNYIPW) – IEEE Signal Processing Society*, Rochester, NY, 24 September.

Yampolskiy, R.V. (2010) 'Application of bio-inspired algorithm to the problem of integer factorisation', *International Journal of Bio-Inspired Computation (IJBIC)*, Vol. 2, No. 2, pp.115–123.

Yang, X.S. (2009) 'Firefly algorithms for multimodal optimization', *Lecture Notes in Computer Sciences*, Vol. 5792, pp.169–178.

Yang, X-S. and Deb, S. (2009) 'Cuckoo search via Levy flights', *World Congress on Nature and Biologically Inspired Computing (NaBIC'09)*, Coimbatore, India, 9–11 December.

Yi, S.K.M., Steyvers, M., Lee, M.D. and Dry, M. (2010) 'Wisdom of crowds in minimum spanning tree problems', *32nd Annual Conference of the Cognitive Science Society*, Austin, TX.

Yi, S.K.M., Steyvers, M., Lee, M.D. and Dry, M. (2010) 'Wisdom of the crowds in traveling salesman problems', available at: http://www.socsci.uci.edu/~mdlee/YiEtAl2010.pdf (accessed on 7 January 2011).

Zhang, G-L. and Wei, Y. (2008) 'An improved particle swarm optimization algorithm for solving 0-1 knapsack problem', *International Conference on Machine Learning and Cybernetics*, Kunming, China, 12–15 July.

Zhao, J.F., Huang, T.L., Pang, F. and Liu, Y.J. (2009) 'Genetic algorithm based on greedy strategy in the 0-1 knapsack problem', *3rd International Conference on Genetic and Evolutionary Computing (WGEC '09)*, Guilin, China, 14–17 October.

Zhao, T., Yang, L. and Man, Z. (2008) 'A MSM-PGA based on multi-agent for solving 0-1 knapsack problem', *International Conference on Computer Science and Information Technology (ICCSIT '08)*, Singapore, 29 August – 2 September.