

Comparison between Genetic Algorithm and Genetic Programming Performance for Photomosaic Generation

Shahrul Badariah Mat Sah¹, Vic Ciesielski¹, Daryl D'Souza¹,
and Marsha Berry²

¹ School of Computer Science and Information Technology

² School of Creative Media,

RMIT University, GPO Box 2476V Melbourne Victoria 3001, Australia

smatsah@cs.rmit.edu.au,

{vic.ciesielski,daryl.dsouza,marsha.berry}@rmit.edu.au

Abstract. Photomosaics are a new form of art in which smaller digital images (known as tiles) are used to construct larger images. Photomosaic generation not only creates interest in the digital arts area but has also attracted interest in the area of evolutionary computing. The photomosaic generation process may be viewed as an arrangement optimisation problem, for a given set of tiles and suitable target to be solved using evolutionary computing. In this paper we assess two methods used to represent photomosaics, genetic algorithms (GAs) and genetic programming (GP), in terms of their flexibility and efficiency. Our results show that although both approaches sometimes use the same computational effort, GP is capable of generating finer photomosaics in fewer generations. In conclusion, we found that the GP representation is richer than the GA representation and offers additional flexibility for future photomosaics generation.

Keywords: Photomosaic, Genetic Programming (GP), Genetic Algorithm (GA).

1 Introduction

Photomosaics are a new form of digital mosaic composed of a tessellation of thumbnail pictures known as tiles. When viewed from afar, the subject becomes evident as we perceive the association of the tiles rather than the individual tiles. When viewed close up, the subject is invisible as the details of each tile emerge. An example of a photomosaic appears in Figure 1.

In a common approach photomosaics are generated by distributing one or more copies of tiles from among a set of small, carefully selected image tiles, across a two-dimensional gridded canvas. Besides being artistically interesting, a photomosaic may be viewed as a solution to a combinatorial optimisation problem based on the examples given by Mitchell et al [3]. Since photomosaic generation involves a tile collection and a set of fixed locations on a gridded

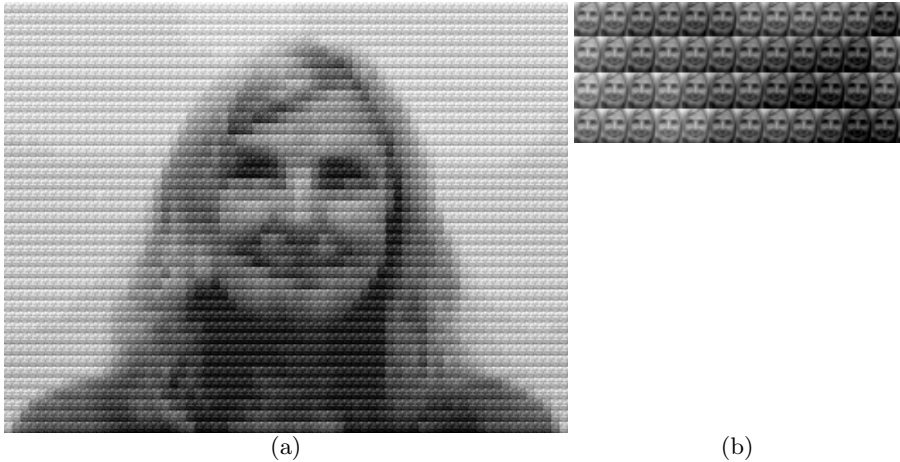


Fig. 1. Example of (a) a photomosaic and (b) a close-up of the tiles

canvas, it may be likened to the problem of attaining an optimal distribution of limited available resources, a problem well-suited to resolution via evolutionary computing. The use of evolutionary computing also allows parallel search for many possible solutions in a single run. Photomosaics are of interest to both artists and computer scientists, and indeed, our work in photomosaic generation has involved ongoing collaboration with artists. From our experiences of working with artists, two important aims of photomosaic generation have been established. First it is necessary to be able to arrive at the final photomosaic as soon as possible, that is, with the smallest number of generations. Secondly, it is desirable to have a flexible development environment, in order to be able to easily implement new creative suggestions from the artists, for example, a new tile arrangement strategy. We anticipated that genetic algorithms (GAs) will be more suited to the first goal, as we expect that GAs use fewer evaluations to generate better photomosaics, while genetic programming (GP) will be better suited to the second goal. As such, the aim of this paper is to compare the GA and GP representations of photomosaic problem. A more generic treatment of GA and GP problem representations appear, respectively, in [3] and [2]. Furthermore, some experiments were carried out to seek answers for improved representations, in terms of the following criteria.

- Number of generations needed to generate the best photomosaic.
- Development flexibility for artistic variation implementation.

The remainder of this paper is structured as follows. Section 2 discusses previous research related to photomosaic generation and to comparative studies between GA and GP. In Section 3 we describe the overall strategy to generate a photomosaic and present our formula to determine the fitness of a solution. Section 4 provides the experimental backdrop to the assessment of the differences between our GA and GP implementations for photomosaic generation.

The results of these experiments are presented in Section 5. Finally, Section 6 summarises our findings and provides pointers to future research direction in the area of photomosaic generation.

2 Related Work

Research in computer-generated photomosaics was pioneered by Silvers [5] in 1997. Silvers' approach relied on a large image repository in order to maximise the probability of finding matches between a given target image and a set of tiles. Silvers' pioneering work attracted further interest in photomosaic generation, and ranging from performance improvements to variations in problem-solving strategy. We briefly review the most relevant of such work.

Finkelstein and Range [6] applied wavelets-based image matching between potential tiles and target image in order to speed up the generation process. Di Blasi et al. focused on technique to speed up the tile search process by using Antipole Tree structure [7]. Images from a database were first classified using the tree structure before any photomosaic was constructed.

Other work has focused on creating forms of photomosaics using different tile placement strategy. The Jigsaw Image Mosaic (JIM), created by Kim and Pelacini [8], is able to create arbitrary-shaped mosaics using arbitrary shaped tiles. The use of stacking layers of tiles for photomosaic generation was introduced by Park [9]. However, in both of these studies the use of exhaustive search was employed, to generate photomosaics.

More recently work in photomosaics has explored the use of evolutionary algorithms in the generation process. Ciesielski et al. [4] investigated the use of GA to produce frames for animated photomosaics. Small sets of generic images and miniature portraits have been used in the photomosaic generation. The work of Wijesinghe et al. [10] explored the use of random tile placement using genetic programming GP. Working with small set of tiles, this study allowed rotation and overlapping between tiles to create the photomosaics.

In this paper we present results of a comparative study of GA and GP for photomosaic generation. Comparative studies between evolutionary algorithms is not a new idea, with several having been completed for different problems to better understand the use of evolutionary algorithms in arriving at good solutions. In a comparative study by Sinclair and Shami [12], in the area of software agents, the comparison is made using almost similar problem formulations, with one function discarded from the GP approach for the ease of implementation and some modifications to the range of arguments in the GP program. Meanwhile the GA implementation is a simplified version of an existing GA-based work of Maskell and Wilby(mentioned in [12]). Although the study served as preliminary work for a larger project to evolve complex software agents for telecommunication network, the results of the study provided some insight into performance and its dependency on having a focused set of functions for particular problems, regardless of the GA or GP implementation. However, the concluding decision to proceed with GP implementation was based on the fewer number of generations

from the experiments. Another comparative study between GA and GP, in the area of robot control [13], it was discovered that using different approaches, the problem of finding the best motions for a robot to score a goal required different representations and there are some trade-offs in each implementation. Based on these previous studies, it has been shown that the implementation of a problem in GA and GP is not directly comparable. Hence, it is of interest to us to assess the performance of GA and GP to find a better representation for photomosaic generation.

3 Problem Formulation

To generate a photomosaic in each of the GA and GP approaches, we start with a blank canvas, the size of the target image. The canvas is logically viewed as comprising a grid of width K pixels and height of L pixels. For tiles each of width k pixels and height l pixels, the canvas will accommodate $K/k = n$ tiles in each row, and $L/l = m$ tiles in each column. Tiles to be placed on the canvas are randomly selected from a tile collection. The size of each tile in this collection is set sufficiently small so as to ensure that, together with other tiles of the same size, it forms a clear photomosaic, whilst at the same time large enough to show the tile details when viewed close up. The total number of tiles for the given target is $m \times n = C$. The range of integers $[0, C - 1]$ is used to index the cell locations on the gridded canvas. As such, cells are filled in sequential order starting from the top left corner to bottom right corner of the grid, row by row.

Fig 2 provides a simple example of the use of the canvas space. The canvas in the example can accommodate a total of 6 tiles, that is, $n = 2$ (the number of tiles in each column), and $m = 3$ (the number of tiles in each row). The tile positions are numbered 0 to 5, and taken row by row. In our implementation a tile can be reused, as in the case of cell locations 0 and 2. In order to fill the cells, tile selection is made from tile collection, implemented as a list. The tile selection is done randomly using the indices of the tile collection. In the example provided, tiles numbered 15 and 10 represent the 15th and the 10th tiles in the tile collection. In this study, we are more interested in computational analysis than the artistic output. Therefore, a small-sized target image and a small tile collection (i.e. 16 flat-shaded potential tiles) were chosen. The size of the canvas and the target image were set at 120×100 pixels and the size of each tile was set at 5×5 pixels. The grid can fit $n = 24$ tiles in each row and $m = 20$ tiles in each column.

3.1 Fitness Evaluation

The fitness of an individual (photomosaic) is calculated as the *sum of pixels differences* between the candidate photomosaic and the target image with i and j referring to a pixel's position, and is given by the following formulation.

$$\sum_{i=1}^K \sum_{j=1}^L |target(i, j) - individual(i, j)|. \quad (1)$$

Here, $target(i, j)$ is the pixel value of i^{th} location on K width and j^{th} location on L height of the target image, and likewise for $individual(i, j)$ on a generated photomosaic. This measure is further *normalised* to scale to the range of $]0, 1]$ by dividing the final value with the total number of pixels on the target.

4 Experiments: GA and GP Approaches

We present our implementations for the GA and GP to generate photomosaics, in the following subsections. The common parameters of evolutionary algorithms such as population size and number of generations are defined to be equal. However, the GA and GP programs use different mutation rate, suitable for each approach. Details of parameters for both GA and GP implementations are listed in Table 1. To support our discussion Fig 2 presents a simplified example of a target image, an evolved photomosaic and a table of selected tiles. The table also contains the position of each selected tile on the grid. Additional parameters for each implementation are explained in the designated subsections.

4.1 GA-Based Program

The GA chromosome is a concatenation of C tiles in a single list, which in this case is 480 tiles. A chromosome is generated by placing the first selected tile from the tile collection in the first cell of the chromosome, the second selected tile is placed in the second cell of the chromosome, and so on. The sequence of selected tiles in the chromosome correlates to the tile positions on the grid. As the grid is filled sequentially, the first tile from the chromosome be placed in the first cell of the gridded canvas, the second selected tile is placed in the second cell of the gridded canvas, and so on.

A chromosome of the initial population is generated by randomly selecting tiles from the tile collection, and placing the tiles, as they are selected, onto the canvas as described above. We implemented the GA program using SGA-C [11]. GA parameters used are presented in Table 1. In addition, a maximum number of 9 mutations were allowed for a single chromosome. Based on the fitness value, a new best photomosaic is identified from each evolutionary run and written out (saved) as an image file. Fig 2 shows an example of a GA chromosome for the given target image.

4.2 GP-Based Program

To represent the photomosaic problem as a GP parse tree, we provide one function and one terminal as the building blocks. The function *TileJoint* allows up to three terminals to be linked together in the tree structure and does not contribute to the fitness calculation. Each terminal, that is, *Tile*, provides the

Table 1. GA and GP configuration

Parameter	GA Value	GP Value
Population Size	200	200
Max Generations	1,000	1,000
Crossover Rate	0.70	0.70
Mutation Rate	0.0001	0.25
Elitism Rate	0.05	0.05
Crossover	2-point crossover between integer boundaries	standard 1-point crossover between randomly selected subtrees
Mutation	Randomly selected tile	Randomly generated sub-tree
Max depth	-	7
Min depth	-	2
Terminal	-	Tile
Function	-	TileJoint
Target Size	120 x 100 pixels	120 x 100 pixels
Tile Size	5 x 5 pixels	5 x 5 pixels
Selection	Proportional to fitness	Proportional to fitness
Replacement	Generational replacement	Generational replacement
Termination	Number of generations	Number of generations

index of the selected tile from the tile collection. Unlike GA, GP constructs parse trees of differing height for different individuals. However, for the photomosaic problem, only the first C tiles ($C = 480$ in our experiment), traversed from a parse tree will be used to construct the phenotype which is a photomosaic. If a parse tree has fewer than C terminals, the fitness will still be calculated but the individual will eventually be discarded in the evolutionary process, as the fitness value is worse. As the fitness evaluation is based on equation 1, a value of 1 indicates that the generated photomosaic bears no resemblance to the target image. Therefore a fitness value *closer to 1* is considered as *the worst* which is also been implemented in the GA approach. The GP program was developed using our own RMITGP package. Table 1 presents the parameters used in the GP program. Fig. 2 also shows an example a GP parse tree(chromosome). For clarity, only two tiles are connected to a TileJoint(TJ) function.

5 Results

In our experiment 5 runs were carried out for each of the GA and GP approach. In the context of photomosaic problems the best fitness value remains unknown, since the value of pixels in a photomosaic is not an exact duplicate of the target and a perfect matching between the target image and the generated photomosaic will never occur. Therefore, in this problem, the number of generations is used as the termination condition (as mentioned in Table 1).

Over the course of 1000 generations, the average of 5 runs shows an interesting result. GA exceeds the performance of GP based on the best fitness values

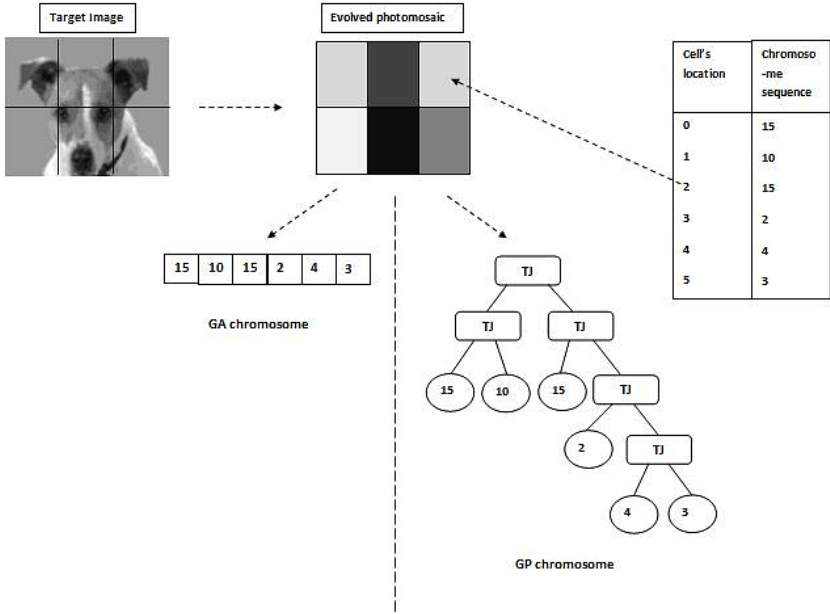


Fig. 2. An example of the GA and GP representations for photomosaic generation

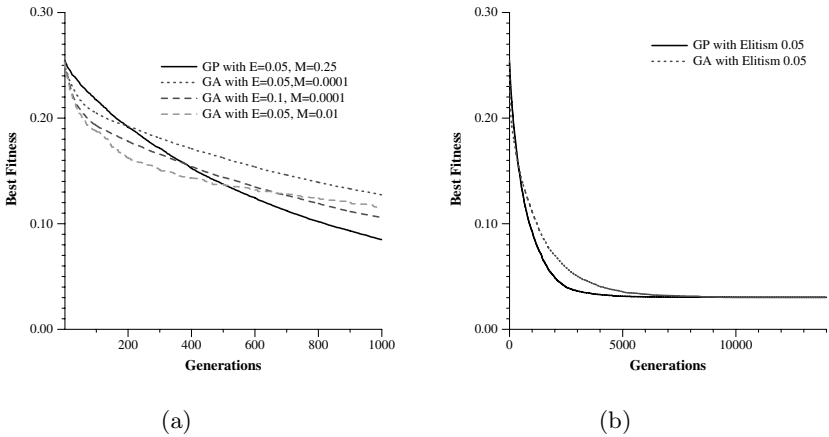


Fig. 3. (a) Results of GP and GA with different elitism rates and mutation rate (E and M represent elitism rate and mutation rate) (b) The performance of GP and GA for 14 000 generations

between generations. However at almost 200 generations, GP starts to converge with GA and thereafter achieves better fitness values than GA.

Based on these preliminary results, we conducted further experiments with the GA program. Using an elitism rate of 0.1 with other parameters remain the same,

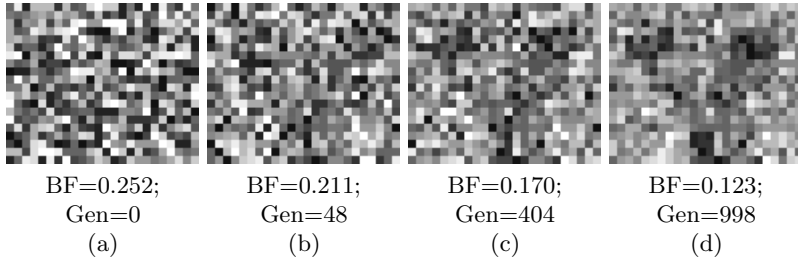


Fig. 4. Sample frames from the GA approach with elitism rate 0.05, BF represents best fitness value while Gen refers to generation

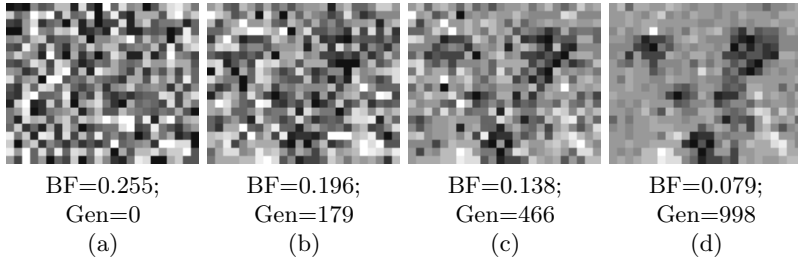


Fig. 5. Sample frames from the GP approach, BF represents best fitness value while Gen refers to generation

we repeated the 5 runs. We found that the new elitism rate further improved GA performance, though again it was eventually outperformed by the GP program. In order to understand more about the GA program, a higher mutation rate (i.e. mutation rate of 0.01) combined with the initial elitism rate of 0.05 have been used in further investigation. Nevertheless, the result did not shows much improvement to the GA performance. Figure 3(a) presents the average best fitness values over 5 runs for the GP program and GA program. The GP program was executed using elitism rate of 0.05 while the GA program uses elitism rates of 0.05, 0.1 and mutation rate of 0.01.

We also looked at the individuals in the GP program and found that in the earlier generations, larger tree size was generated with 1093 nodes consisting of 364 *TileJoint* and 729 *Tile* in the parse tree. However, approaching 1000 generations, we discovered that the size of the parse tree for the best individual was about 730 nodes, comprising 243 *TileJoint* and 487 *Tile* nodes. As the total number of tiles needed is 480, the analysis indicates that the GP program eventually adapts to the desired size of the solution, with a small number of unused tiles for genetic operations.

Finally, for both approaches we investigated the impact of longer runs, specifically to assess convergence. After 14,000 generations, we decided to terminate the runs as the fitness values converged to a stagnant measure, as shown in Figure 3(b). Figure 4 and Figure 5 provide samples of photomosaics generated using GA and GP, respectively.

6 Discussion and Conclusions

Referring to the research questions listed in Section 1, the findings show that, at least for the limited problem size explored, GP is able to produce better photomosaics in fewer generations. Even though the GP and the GA programs eventually converged at almost similar point as shown in Figure 3(b), GP produces better photomosaics than GA. Our initial assumption was that the GA approach would be superior to the GP approach, as the photomosaic generation problem is a fixed-length problem. Our implementation of photomosaic generation adopts a fixed number of cells on a canvas (480 cells in our experiment). Since GA represents a problem in fixed-length chromosomes while in GP the chromosomes are variable-length parse trees, the photomosaic generation problem would appear to be better suited to the GA approach. We expected the GP program to take longer to converge to the result produced by the GA photomosaic program. Our expectation was based on the variation in parse tree height for each individual, as well as the occurrence of an oversupply of tiles in the parse tree, thus lengthening the process of translating genotypes to phenotypes. In reference to Figure 4(d) and Figure 5(d), respectively, the fitness values and the visual qualities of the figures over 1000 generations indicate that the GP produces better photomosaics than the GA program (i.e. closer to target image which can be referred to Fig 2). This might be due to the richer representation brought about by the GP approach. Although only the first 480 terminals are used in a photomosaic construction, the extra terminals in the parse tree are still available for the crossover operation. This could have contributed to the generation of fitter individuals.

On the other hand, for the GA approach, the two-point crossover operator could have been the reason behind its unexpected poor performance. We can only speculate that two-point crossover creates destructive combinations in the evolution process, creating unfit photomosaics. The search space for the GA was about 10^{42} while GP search space was larger with approximately 10^{117} based on the equation from [14]. Despite the search space being massive, GP program was able to narrow down the search to the area of possible best solutions earlier than the GA program.

In terms of development flexibility, to experiment with variations in tile placement strategy in GP (for example, non-grid-based tile placement, in which tiles may be placed anywhere on the canvas) requires an introduction of new terminals, with the extra coding effort localised in the *Draw* function. In contrast, in the GA approach, to implement a new placement strategy requires a major change to the chromosome structure, to include the position information.

As the work presented here is a part of an ongoing study, further experiments and analysis are required to verify the results. Such experiments will involve additional runs and in the context of larger problems of similar nature. However, it is anticipated that future work will be focused on the use of the GP approach for photomosaic generation, given its greater flexibility for a range of tile placement strategies and the representation of the problem.

References

1. Hinterding, R.: Representation, Mutation and Crossover Issues in Evolutionary Computation. In: Proceeding of Congress of 2000 Evolutionary Computation (CEC 2000), vol. 2, pp. 916–923. IEEE Service Center (2000)
2. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Massachusetts (1992)
3. Mitchell, M.: An Introduction to Genetic Algorithms. MIT Press, Massachusetts (1996)
4. Ciesielski, V., Berry, M., Trist, K., D'Souza, D.: Evolution of Animated Photomosaics. In: Giacobini, M., et al. (eds.) EvoWorkshops 2007. LNCS, vol. 4448, pp. 498–507. Springer, Heidelberg (2007)
5. Silvers, R., Hawley, M.: Photomosaic. Henry Holt and Company, Inc., New York (1997)
6. Finkelstein, A., Range, M.: Image Mosaic. In: Hersch, R.D., Andre, J., Brown, H. (eds.) RIDT 1998 and EPub 1998. LNCS, vol. 1375, pp. 11–22. Springer, Heidelberg (1998)
7. Di Blasi, G., Gallo, G., Maria, P.: Smart Ideas for Photomosaic Rendering. In: Proceedings of Eurographics Italian Chapter Conference 2006, Eurographic Association, Catania, Italy (2006)
8. Kim, J., Pellacini, F.: Jigsaw Image Mosaics. ACM Transactions on Graphics (TOG) 21, 657–664 (2006)
9. Park, J.W.: Artistic depiction: Mosaic for Stacktable Objects. In: ACM SIGGRAPH 2004 Sketches SIGGRAPH 2004. ACM, New York (2004)
10. Wijesinghe, G., Mat Sah, S.B., Ciesielski, V.: Grid vs. Arbitrary Placement of Tiles for Generating Animated Photomosaics. In: Proceeding of Congress of 2008 Evolutionary Computation (CEC 2008). IEEE Service Center, Piscataway (2008)
11. Smith, R.E., Goldberg, D.E., Earickson, J.A.: SGA-C: A C-language Implementation of a Simple Genetic Algorithm (1991), <http://citeseer.ist.psu.edu/341381.html>
12. Sinclair, M.C., Shami, S.H.: Evolving simple agents: Comparing genetic algorithm and genetic programming performance. In: IEE Genetic Algorithms in Engineering Systems: Innovations and Applications, pp. 421–426. IEEE Press, New York (1997)
13. Walker, M., Messom, C.H.: A Comparison of Genetic Programming and Genetic Algorithms for Auto-tuning Mobile Robot Motion Control. In: Proceedings of the First IEEE International Workshop on Electronic Design, Test and Applications (DELTA 2002), pp. 507–509. IEEE Press, New York (2002)
14. Ebner, M.: On the search space of genetic programming and its relation to nature's search space. In: Proceedings of the 1999 Congress on Evolutionary Computation, Washington, D.C, July 6-9, vol. 2, pp. 1357–1361. IEEE Press, Los Alamitos (1999)