HYPERMARKET – an evolutionary paths planner

Anna Kunicka[1] and Halina Kwasnicka[1]

[1] Wroclaw University of Technology, Institute of Applied Informatics, Wroclaw, Poland,
email: aniakunicka@o2.pl, halina.kwasnicka@pwr.wroc.pl

Abstract. The paper presents a new method for path planning. As a real task we assume path planning during shopping in hypermarkets. The method uses a hierarchical approach, where the first algorithm at the low level is responsible for searching a feasible shortest path between two points in the hypermarket, but the second one, at the high level, is developed for searching the best ordering of purchases and the most suitable cash desk. The main goal is to find the shortest path from an entry, via all assumed purchases to the cash desk. In our study, we developed genetic algorithms at the both levels. However, we tested other approaches, namely Ant Colony Optimization and Dijkstra's algorithm.

1 Introduction

The path-planning problem is a typical problem that mostly occurs in issues concerning mobile robots, games, etc. It consists in finding the path that is collision-free and satisfies certain optimization criteria (for example time, length) in an environment that includes obstacles [8]. There are many approaches that lead to finding a solution; one of them is based on genetic algorithms. An example of such an approach can be found in [6, 8], where an adaptive Evolutionary Planer/Navigator (EP/N) is presented. EP/N finds the near-optimality paths for mobile robots. Improvement of this method is proposed in [7] where the authors reduced the size of the search space. The modified EP/N takes advantage of heuristic knowledge gained through the robot's reaction to the environment.

A genetic algorithm that tries to find the feasible path in an environment in which the robot moves is presented in [4, 5]. It uses simple genotype structure that contains only the essential information for path planning. The authors concentrated mainly on efficient processing. Unfortunately it is not able to find feasible paths for all search spaces.

A genetic algorithm that finds the solution to the problem of planning a robust sub-optimal trajectory in the velocity space of mobile robot is described in [3].

In this paper we present an evolutionary approach for solving a particular instance of path planning problem: we try to develop an efficient evolutionary algorithm for finding the shortest path from an entry, by shelves with required (assumed) purchases to one of the opened cash desks in the hypermarket. We assume that the locations of shelves with needed purchases and cash desks are known. We have designed the hierarchical genetic algorithm (the method called **HGAPath** – **H**ierarchical **G**enetic **A**lgorithm for paths planning) for this task. Experiments using developed computer program HYPERMARKET reveal the usefulness of the **HGAPath** as a paths planner tool.

The paper is structured as follows. Section 2 contains problem formulation. In section three we describe the proposed method together with genetic algorithms at the low and high levels. A huge number of experiments has been performed, but only a few are presented in section four. A short summary ends the paper.

2 Problem definition

Our goal is to find the shortest path during shopping in a hypermarket with an assumed rectangular shape. In the hypermarket there are a number of shelves that are also rectangular. In general, shelves are perceived as obstacles, but we must visit also shelves, where our purchases are located. These are fixed places, which have to be reached to make particular purchases. Additionally, we assume only one entry and a number of cash desks, all represented as points on the considered hypermarket surface. The number of buys is fixed. Fig. 1 presents a schema of hypermarket representation in our study. As a

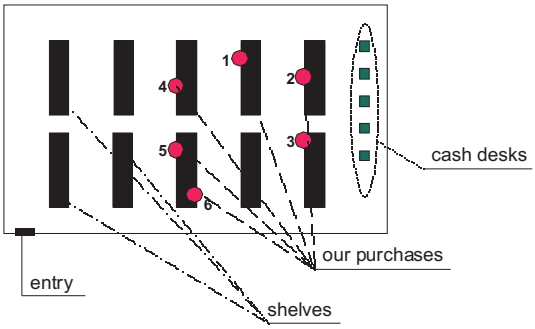

Figure 1. An example of hypermarket.

result, we obtain a list of points that our path has to pass through. The path that we are looking for has to start in the entry point, pass through all points symbolizing purchase, and end in one of points that represent cash desks. The path has to be feasible, what means that none of fragments of the path can lie on any shelve.

3 Hierarchical Genetic Algorithm

HGAPath is a hierarchical algorithm, it is composed of two levels of algorithms: the low level and the high level (see Fig. 2). The low level algorithm finds the feasible and as short as possible path between any two points situated in the hypermarket, it means, between the location any two purchases. The high level algorithm is responsible for determining the order of taking buys and choosing the cash desk. Thanks to such division we can work these two algorithms separately and the complete algorithm is very flexible. It is worth underlying that we can put at every level not only genetic algorithm, but also any algorithm that is able to solve the same problem. For example, instead of genetic algorithm at the high level we can use **A**nt **C**olony **O**ptimization (ACO) algorithm [2],

at the low level – Dijkstra's algorithm [1]. In the mainstream of our study we use genetic algorithms in both levels.

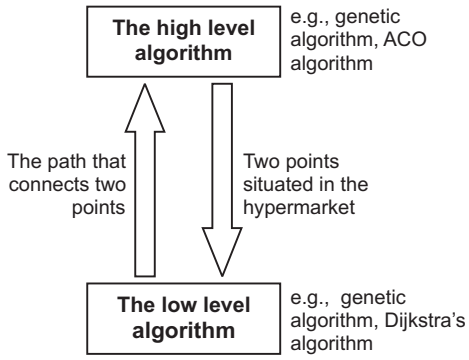

Figure 2. Schema of the **HGAPath**.

3.1 The low level genetic algorithm

The low level genetic algorithm is responsible for finding the feasible and the shortest path between two points situated in the hypermarket.

Chromosomes, fitness function and method of selection. A chromosome represents a path that consists of straight-line segments – a sequence of points. One segment joins adjacent points in a chromosome. All chromosomes (individuals) have the same first point and also the last point. Between these points (i.e., two purchases) we want to find a path in the hypermarket. All individuals in every population are feasible.

Fitness function f is defined as the total length of the path, it has to be minimized. We use the rank-based selection method, it is suitable for minimization problems and does not require fitness scaling. It works in that individuals in the current population are sorted decreasingly by values of fitness function. Each individual is assigned rank that is equal to the position in the new order. The number of descendants of particular individual amounts to integer part of a quotient of doubled rank and a total number of individuals in the population. This method does not provide enough number of individuals, therefore the rest of lacking individuals is taken additionally on the strength of reminder of earlier mentioned quotients. Moreover one extra copy of the best individual goes to the new generation without any changes (so-called *survive the best*).

Initialization. In this subsection we describe how chromosomes are generated. Initially every chromosome consists of two points. the path which we look for has to join these two points. Next, a verification of the chromosome feasibility is made. If yes, the algorithm ends, because the shortest path is found. If chromosome does not represent a feasible path, there is randomly chosen a vertex of shelve from such vertices of shelves, which create a feasible straight-line segment with the first point in the chromosome. A point that is situated near the chosen vertex is inserted into chromosome between the

first and the second point. It is then checked if the whole path represented by the chromosome is feasible. If it is, algorithm ends, if not – there is randomly chosen a vertex from such vertices of shelves, which create a feasible straight-line segment with the second (previously inserted) point in the chromosome. Point situated near the chosen vertex is inserted into chromosome between previously inserted point and the last point of the chromosome. Again the verification is made – the procedure is repeated until the path represented by the chromosome is feasible.

Genetic operators. Developed genetic operators work on feasible chromosomes and create chromosomes that are feasible, too.

Mutations. In the low level genetic algorithm there are eight kinds of mutation. Two remarks are important concerning mutation: (1) all kinds of mutation are made only under assumption that chromosome after mutation remains feasible, (2) the first and the last point in the chromosome cannot be mutated.

- Deletion of the point. It deletes a randomly chosen point from the chromosome.
- Insertion of the point. It inserts a randomly generated point before a randomly chosen point in the chromosome. Of course, before the first point in the chromosome there cannot be inserted any point.
- Displacement of the point. It displaces a randomly chosen point. Values of two coordinates of randomly selected point are changed.
- Change of one coordinate of the point. It changes one coordinate of a randomly chosen point. This operator differs from the previous one, because it makes only small changes in coordinates of the points, while the former makes it much more bigger.
- Deletion of a few points and generation new ones in their place. It deletes a randomly chosen few adjacent points from the chromosome and generates on their place others (not necessarily the same amount).
- Displacement of two adjacent points in the same direction with the same value. It changes a single coordinate of two randomly chosen adjacent points in the chromosome. The randomly chosen coordinate is changed. The same value is added to the coordinate of these two points.
- Displacement of two adjacent points in the same direction with the different value. This operator works like the previous one. There is only one difference - dissimilar values are added to the selected coordinates.
- Displacement of few points. It displaces a few randomly chosen adjacent points in the chromosome. There are randomly chosen new coordinates of these points.

Crossover. Crossover involves two chromosomes. A test is made as to whether paths represented by these two chromosomes meet in some place. If there does not exist an intersection point – chromosomes are not changed. If there exists such point – the first descendant is created in that way, that the first part of points from the first chromosome (before the intersection point) become the first part of points of the descendant. Next, to the descendant is added the intersection point and then the last part of points (after the intersection point) from the second chromosome. The second descendant is created in the same way, but its first part comes from the second chromosome and the second part – from the first parent. Descendants replace parents in the population. If there is more than one intersection point, only the first one becomes the crossover point.

3.2 The high level genetic algorithm

As it was mentioned, this algorithm is responsible for determining the order of taking purchases and choosing the cash desk. It uses the results provided by the low level algorithm.

Chromosomes, initialization, fitness function and method of selection. Every purchase has a unique number. Similarly, all cash desks are also numbered. A chromosome is a sequence of the numbers of purchases (a permutation) and, as a last number, it contains a number of cash desk, where the path ends. The length of chromosome is constant and is equal to number of buys incremented by one. The exemplary chromosome, which codes a path that passes by six buys in a hypermarket containing eight cash desks, could be a sequence (bold number is a number of cash desk): (2 5 3 4 1 6 **5**).

An initial population is randomly generated. First, there is generated a random permutation of numbers of purchases and then a random number of cash desk.

Fitness function is defined as a total length of the path that is coded by the particular chromosome. To calculate it we have to know the length of every path that joins adjacent points in the chromosome. In this place we have to make use of results generated by the low level algorithm. Fitness function, of course, has to be minimized. We use a tournament selection (three individuals take part in a tournament). Furthermore, the best individual in the current population goes to the next population without any change.

Genetic operators. Now we introduce genetic operators that we used in the high level genetic algorithm.

Mutations. In the high level genetic algorithm there are four kinds of mutation.

- Change of a cash desk. It changes a current number of a cash desk. The new number of cash desk is chosen randomly.
- Exchange of two randomly chosen buys. It exchanges two randomly chosen buys in the chromosome. For example, the individual (2 *5* 3 *4* 1 6 **5**) could be modified into (2 *4* 3 *5* 1 6 **5**).
- Insertion of a randomly chosen purchase in a random place. It inserts a randomly chosen purchase into a random place in the chromosome. Of course, this purchase cannot be inserted at the last place in the chromosome, because this place is reserved for the number of a cash desk. For example, the individual (3 6 1 2 *5* 4 **7**) could be modified into (3 *5* 6 1 2 4 **7**). The buy number 5 was inserted at the second position in the chromosome.
- Inversion. Two points in the chromosome are randomly selected and the part of chromosome between these points is inverted. The last number in the chromosome (cash desk) is not a subject of inversion. For example, the individual (3 | 5 4 1 6 2 | **2**) could be modified into (3 2 6 1 4 5 **2**).

Crossover. Crossover of two chromosomes is analogous to **PMX** (**P**artially **M**atching **C**rossover) defined to solve the traveling salesman problem [6]. The last number in the chromosome does not take part in the crossover. For example, two chromosomes:

```
(2 | 3  1  5 | 6  4  2)
(3 | 6  2  4 | 1  5  6)
```

could be transformed into:

```
(3   6  2  4   1  5  2)
(6   3  1  5   2  4  6)
```

4 Simulation study of the proposed method (HGAPath)

To verify proposed approach, the computer program named HYPERMARKET was worked out. Using this program a number of simulations have been conducted. At the beginning, the first phase of study was done – the two genetic algorithms, proposed at the high and the low levels, were studied separately. These experiments allow to state usefulness of developed algorithms and to tune all needed parameters, as probabilities of mutations and crossovers and a size of population. All presented in the paper experiments were performed with using values of parameters found in the first phase.

Below we present an example of the results obtained during study of the low level genetic algorithm. Next subsection shows the results of the **HGAPath**.

4.1 Testing the low level genetic algorithm

Providing short paths between particular two purchases is crucial for the high level algorithm, therefore the quality of the low level algorithm is very important.

Fig. 3 shows paths found by the algorithm in the untypical hypermarkets. Algorithm copes with this task very well. It found optimal paths in relatively short time. It makes it mainly thanks to the method of initialization of chromosomes – it works well for such kinds of problem. The experiments show us that the algorithm is able to find paths between two points on the surface with obstacles; the hypermarket with shelves is only an example of such a task. Figures 4a), 4b) and 4c) present paths found in hypermarket that has higher amount of shelves than hypermarkets before. The best individual after 10 generations is shown in Fig. 4a), in Fig. 4b) – after 61 generations, and in Fig. 4c) – after 144 generations. As we can see, at the beginning the algorithm found path that was not near the shortest path between points. But thanks to diversity of genetic operators it was able to find the optimal path (Fig. 4c)).

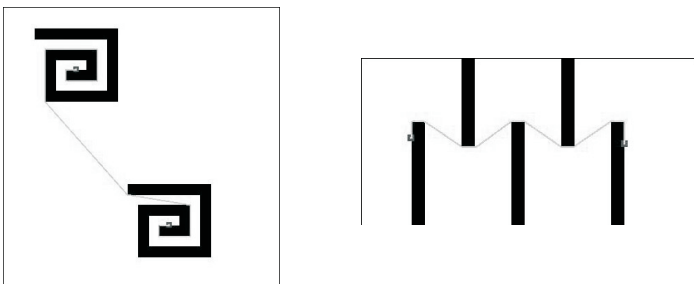

Figure 3. Paths between two points found in untypical hypermarkets.

We also tested Dijkstra's algorithm, but the results were not so good as we expected. Dijkstra's algorithm worked at a graph that was constructed in that a node was created for every point in a hypermarket, which did not lie at any shelve. Arc connected such pairs of nodes that represented points situated directly next to each other. Weights of

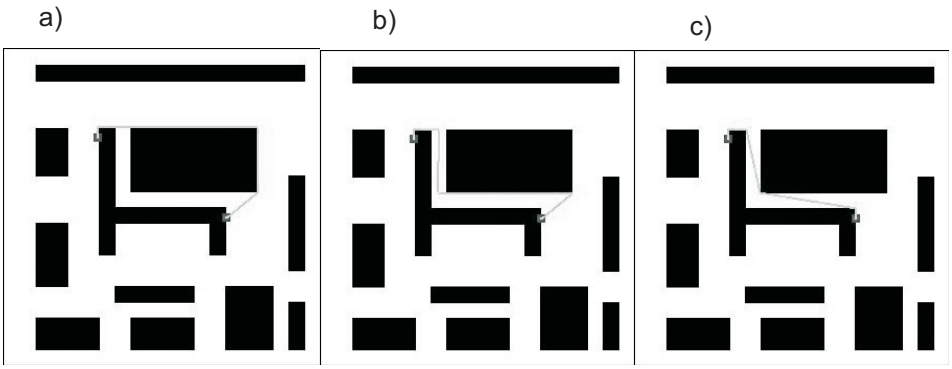

Figure 4. Paths between two points – the best individual after 10 (a), 61 (b), and 144 (c) generations.

arcs amounted to the distance between points that was represented by the nodes. Such construction of the graph is very convenient (it is not necessary to hold whole structure of the graph in memory and it is very easy to create such graph), but sometimes it makes finding the shortest path between two points impossible. There is taken into consideration only direct neighborhood of points and paths that this algorithm finds are "angular" (see Fig. 5). As we can see in Table 1, genetic algorithm is generally better then Dijkstra's algorithm (it finds shorter paths in shorter time) but there are instances when Dijkstra's algorithm works better. Such situations take place when hypermarket has big number of shelves and there are not many free space between them.

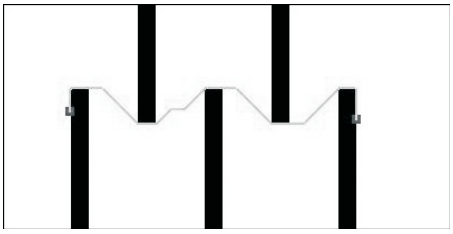

Figure 5. Example of the path between two purchases found by Dijkstra's algorithm.

4.2 Simulation study of the whole proposed method

In this section we shortly present how the whole hierarchical genetic algorithm works. Two examples are presented, the first with regular arranged shaves (plain hypermarket), and the second one, that seems to be more difficult.

Experiment with the plain hypermarket. Fig. 6a) shows the best individual after the second generation. It is easily seen that this path is far from the optimal one. The path after 59 generations (Fig. 6b)) is evidently shorter then the previous one. Further evolution leads to the better solution – see Fig. 7. Our method found the optimal path

Table 1. Paths found by algorithms tested at the low level (Genetic and Dijkstra's algorithms) for different hypermarkets.

Hypermarket	Genetic Algorithm			Dijkstra's Algorithm	
	Generation Number	Length	Time [min:s]	Length	Time [min:s]
Plain hypermarket with 4 shelves	3	236,4802	00:0.1812	238,1232	01:03.4062
	36	232,9650	00:03.3328		
	156	232,1086	00:15.1953		
Hypermarket as in Fig. 3 (on the left)	10	717,0866	00:18.4296	683,2325	33:01.0625
	50	683,2647	01:30.9843		
	330	670,9768	10:03.8312		
Hypermarket as in Fig. 3 (on the right)	8	357,0657	00:01.4500	357,8477	04:38.9843
	35	350,6917	00:07.3250		
	449	343,4961	01:41.7984		
Hypermarket as in Fig. 4	10	269,5939	00:14.1890	193,4264	01:59.1718
	72	230,8128	01:58.5625		
	250	182,9485	06:37.1125		
More complex hypermarket with 19 shelves	10	471,4351	00:24.3546	368,3624	10:33.1875
	220	402,6661	10:57.0218		
	399	401,3044	20:01.1812		

in the relatively short time. But, we should admit that the studied hypermarket is not complex and the number of buys is not high.

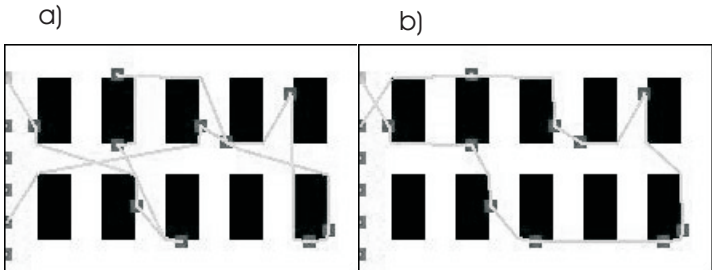

Figure 6. The plain hypermarket – the best individual after 2 (a) and 59 (b) generations.

Experiment with the more difficult hypermarket. We have defined a number of more complicated tasks for the proposed method, but here we show only one example. The hypermarket shown in Fig. 8 is more complex than the previous one, and the number of purchases is higher as well (it amounts to 15). The solution space is larger and the method requires more time for solving such problems. For the task presented in Fig. 8, the hybrid genetic algorithm found solution during 960 generation, what is not very long evolution. The near optimal solution is presented in Fig. 9a), but this path can be shortest, as in Fig. 9b).

We tested also Ant Colony Optimization approach. Table 2 contains results generated by the tested algorithms: genetic algorithm, Elitist Ant System [2] and Rank-Based Ant System [2]. Rank-Based Ant System turned out to be slightly better than genetic algorithm and Elitist Ant System, but generally all algorithms found paths with comparable

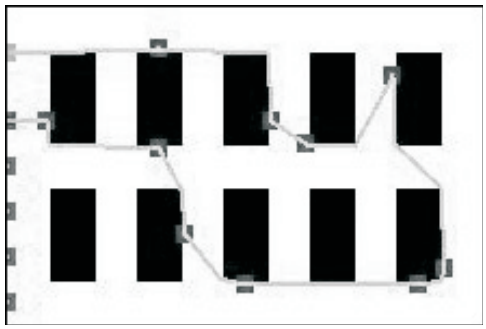

Figure 7. The plain hypermarket – the best individual after 140 generations.

lengths.

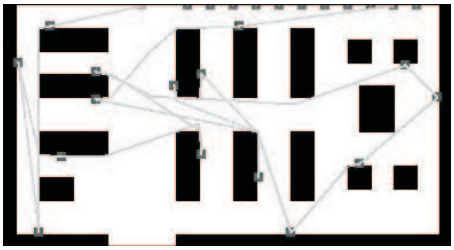

Figure 8. The best individual after third generation.

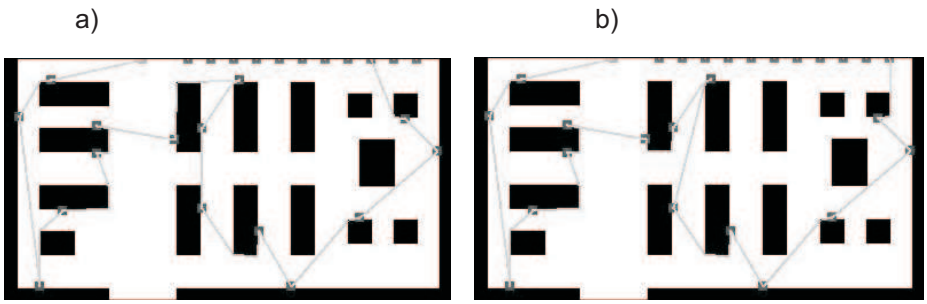

Figure 9. The best individual after 950 (a) and 960 (b) generations.

5 Summary

In the paper we present a general approach for path planning problems. As an example we have used atypical problem - searching for the shortest path from entry to cash desk making by the way all needed shopping. We assume that shaves are obstacles, but our purchases placed on the shelves constitute desired points for the searched paths.

Table 2. Paths found by algorithms tested at the high level for different hypermarkets (GN: generation number, IN: iteration number, L: length, T: time).

Hypermarket	Genetic Algorithm			Elitist Ant System			Rank-Based Ant System		
	GN	L	T[s]	IN	L	T[s]	IN	L	T[s]
Plain hypermarket as in Fig. 6	1	924	0.000	1	645	0.005	1	601	0.006
	44	627	0.019	54	574	0.095	40	588	0.127
	242	574	0.100	56	578	0.096	54	574	0.169
More complex hypermarket as in Fig. 8	1	2308	0.000	1	1459	0.005	1	1242	0.014
	150	1295	0.095	60	1216	0.180	20	1218	0.161
	831	1216	0.505	165	1206	0.483	44	1206	0.334

We have proposed a method consisting of two hierarchical algorithms. As the low level algorithm (searching the shortest path between any two points in the hypermarket) and the high level algorithm (searching the optimal permutation of purchases and an adequate cash desk) we use genetic algorithms with developed special genetic operators and a method of generation of initial population. This method and genetic operators assure feasible solutions. But in general, both genetic algorithms (at low and high levels) can be substituted by any other algorithm. We have tested Dijkstra's algorithm at the low level and Ant Colony Optimization at the high level.

In the future we plan to develop our method by tuning other algorithms and testing the method using a number of different path planning tasks.

Bibliography

[1] Dijkstra, E. W. A Note on Two Problems in Connection with Graphs. *Numerische Math* 1: 269-271, 1959.

[2] Dorigo, M., Stutzle, T. *Ant Colony Optimization*. MIT Press, 2004.

[3] Gallardo, D., Colomina, O., Florez, F., Rizo, R. A Genetic Algorithm for Robust Motion Planning. *Proceedings of IEA-AIE'98*, vol. 1416, 1998.

[4] Geisler, T., Manikas, T. W. Autonomous robot navigation system using a novel value encoded genetic algorithm. *Proceedings 45th IEEE Int. Midwest Symp. on Circuits and Systems* 3: 45-48, 2002.

[5] Hermanu, A., Manikas, T. W., Ashenayi, K., Wainwright, R.L. Autonomous robot navigation using a genetic algorithm with an efficient genotype structure. Intelligent Engineering Systems Through Artificial Neural Networks: Smart Engineering Systems Design: Neural Networks, Fuzzy Logic, Evolutionary Programming, Complex Systems and Artificial Life, 319-324, 2004.

[6] Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1996.

[7] Wang, Y., Mulvaney, D., Sillitoe, I. Robot Navigation by Genetic Algorithms. 2nd ESC Division mini-conference, 2005.

[8] Xiao, J., Michalewicz, Z., Zhang, L., Trojanowski, K. Adaptive Evolutionary Planner/Navigator for Mobile Robots. *IEEE Transactions of Evolutionary Computation* 1: 18-28, 1997.