Parallelisation of Probabilistic Sequential Search Algorithms

Prasanna Jog
Dirk Van Gucht

Indiana University
Computer Science Department
Bloomington, IN 47405
jog@iuvax.cs.indiana.edu
vgucht@iuvax.cs.indiana.edu

Abstract

This paper explores ways of parallelising probabilistic sequential search algorithms that use a local improvement operator to generate iteratively a new candidate solution from previous candidate solution. We study the trade-off between the processors working in isolation and communicating with each other, in terms of required effort and performance achieved.

1. Introduction

This paper deals with the parallelisation of probabilistic sequential search algorithms which generate a sequence of candidate solutions (structures), each derived from the previous one by the use of a probabilistic operator. The type of operators under consideration are those that make small local changes that improve the structure. As an example, consider the probabilistic sequential search algorithm of Lin and Kernighan (the r-opt strategy) [1] used to find an approximate solution to the Traveling Salesman Problem (TSP). The TSP can be stated as follows: Given N cities, if a salesman starting from his home city is to visit each city exactly once and then return home, find the order of visits (the tour) such that the total distance traveled is minimum. In the general r-opt strategy, the operator replaces r edges in the current tour for r edges not in this tour if the resulting tour has a tour length less than the length of the previous tour. For example, the 2-opt strategy randomly selects two edges (i_1, j_1) and (i_2, j_2) from a tour (see Figure 1) and checks if

$$ED(i_1, j_1) + ED(i_2, j_2) > ED(i_1, j_2) + ED(i_2, j_1)$$

(ED stands for Euclidean distance). If this is the case, the tour is replaced by removing the edges (i_1, j_1) and (i_2, j_2) and replacing them with the edges (i_1, j_2) and (i_2, j_1) (see Figure 2).

One way of parallelising a probabilistic sequential search algorithm is to split the problem into n subproblems and let each processor work on one subproblem. Such a division is, in general, not possible. For instance, in a TSP, it is unlikely that we could make different processors attempt edge interchanges simultaneously on the same tour and hope to obtain

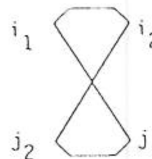

Figure 1. Tour with edges (i_1, j_1) and (i_2, j_2).

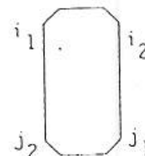

Figure 2. Tour with edges (i_1, j_2) and (i_2, j_1).

a legal tour. In other words, to achieve such parallelisation one has to add conflict resolution techniques, usually resulting in a degradation of the performance of the algorithm.

Another way of parallelising is to let all n processors run independently and take the best available solution at the end. We will call this the *independent strategy*. A potential problem with this strategy is that as the processors run independently, some of them may get caught in a local minima or may search sub-optimal regions of the search space, wasting valuable resource power. Intuitively, it seems likely that we might do better if we let the processors work inde-

pendently for some time, then exchange information about "good" candidate solutions, again work for a while, exchange new information and so on. We call such a method an *interdependent strategy* and call the time of the processing in between two information exchanges a *generation*†. Clearly, there are many ways of exchanging information about good candidate solutions. A straightforward strategy is to overwrite after each generation a certain number of "bad" candidate solutions by good candidate solutions. More sophisticated strategies could involve exchanging structural parts of good candidate solutions. Examples of such strategies are cross-over operators found in *Genetic Algorithms* (GA) [3].

In the rest of this paper, we give evidence that interdependent strategies are usually better than the independent strategy when parallelising probabilistic sequential search algorithms which use local improvement operators. In fact, we suggest that a good technique of parallelisation is to use an interdependent strategy where information exchange is done on a fairly regular basis. In Section 2, we illustrate this approach by parallelising a simple problem, the Classical Occupancy Problem [4]. Section 3 covers the parallelisation of a more complicated search algorithm: the 2-opt strategy of Lin and Kernighan for the TSP mentioned above. In Section 4, we describe experiments with a genetic algorithm for the TSP. We show that genetic algorithms can be viewed as parallel search algorithms that implement an interesting kind of interdependent strategy to achieve good, robust performance. The standard selection procedure of the GA can be viewed as a mechanism for achieving information exchange and the local improvement operator can be viewed as a recombination operator of the GA. Finally, Section 5 offers a discussion of the ideas and results given in this paper.

2. A Toy Problem: The Classical Occupancy Problem

Consider the classical occupancy problem: Given a structure of N empty cells, shoot points randomly at the cells (with a probability of $1/N$ of hitting any given cell) until all cells are filled. The time for solving this problem is the number of shots required to fill all N cells. This problem has been studied extensively by Kolchin et.al. [5]. We present here some experimental results which illustrate the advantages of an interdependent strategy over the independent strategy.

The sequential algorithm involves starting with the initial structure of N empty cells and repeatedly generating a random number between 1 and N, this is called a *trial*. If the cell was empty before, it is now assumed to be full and if not, the trial has been unsuccessful.

In the independent case, this sequential algorithm will run separately on all n processors. In this case, the time required to solve the problem is the number of shots required by the processor that finished first. In the interdependent case the algorithm would look as follows:

```
assign to each processor the empty structure;
full ← 0;
generation ← 0;
while full < N do
begin
  generation ← generation + 1;
  each processor generates a number between 1 and N;
  if (at least one processor is successful) then
  begin
    randomly choose one among the successful processors;
    distribute its structure to all other processors;
    full ← full + 1;
  end;
end;
```

We now consider results of simulation on the classical occupancy problem with $N = 100$. First we fixed the number of processors and varied the number of trials per generation (tpg). As indicated by the above algorithm, after each generation the best structure was redistributed (copied) to all the processors. With the number of processors $n = 10$, we found that $tpg = 1$ resulted in the minimum number of generations. It should be noted, however, that a higher number of trials per generation also did well compared to the independent case. For $N = 100$ and $n = 10$ processors, the independent case required on average (taken over 500 experiments) 370 generations until completion. (Note that 100 is optimal). In the interdependent case, for $tpg = 1$ the processors required on average 128 generations, for $tpg = 4$ the processors required 165 generations and for $tpg = 7$ the processors required 190 generations.

Next we fixed the trials per generation and varied the number of processors. As the number of processors increased the generations required in the independent case reduced, but at a slower rate than the generations required for the interdependent case. For example, with 1 trial per generation and $N = 100$, it took 3 processors on average 221 generations to finish in the interdependent case and on average 427 generations in the independent case. With 5 processors the interdependent strategy required 166 generations while the independent case required 397 generations. With 10 processors the generations required were 128 and 370 respectively. (As the number of processors tends to infinity both strategies will require N generations). Our experiments suggest that doing less number of trials per generation and increasing the number of processors is better.

But this was a simple problem. In this problem, we know the (optimal) solution and among processors that have a successful trial, there is no one best structure, since they all have the same amount of empty cells. That is, they are all equally good. For this reason, we decided not to try a strategy of taking the k best candidate solutions with $k > 1$, although a slightly different performance may be expected. We now consider the parallelisation of a more complex problem and compare the performance of various interdependent strategies with each other and the independent strategy.

3. Experiments with a Search Algorithm for the Traveling Salesman Problem

† Note that if a generation involves infinitely many trials (attempts at local improvements), the interdependent strategy reduces to the independent strategy.

The domain of this experiment is the Traveling Salesman Problem (TSP). We use the operator of Lin and Kernighan (the r-opt strategy), described earlier as an example of a probabilistic operator that makes small local changes to produce new structures from old ones. The larger the value of r, the more likely it is that the final solution (when no more exchanges are possible) is optimal. However, r is usually chosen to be 2 because the possible edge interchanges is of the order of $\binom{N}{r} \times r!$†. In each generation, each processor performs a certain number of applications of the 2-opt strategy on the structure it currently holds in its local memory and after each generation the best structure was redistributed (copied) to all the processors. The domain for experiments described here is the lattice of 100 points spread over (0,0), (0,9), (9,0), (9,9) as shown in Figure 3.

Figure 3. Lattice of 100 cities

Clearly, the optimal tour length is 100‡. We will consider the total effort, given by the number of generations times the number of trials per generation, required to get to within 10% of optimal performance as the number of trials per generation (*tpg*) is varied. Notice that we do not take into account the overhead involved in redistributing the structures after each generation. The algorithm is as follows:

```
for various values of tpg do
begin
  generation ← 0
  generate a structure randomly;
  copy it to all n processors;
while (bestperformance ≥ (1.10 * optimumvalue)) do
  begin
    generation ← generation + 1;
    each processor attempts tpg local improvements;
    find the structure with the best performance;
    distribute this structure to all other processors;
  end
end
```

† It should be noted that Lin and Kernighan propose a more powerful strategy where r is varied dynamically, but the simple 2-opt strategy also gives good results and is quite efficient [2].

‡ It should be noted that similar results can be obtained for other TSPs.

The actual algorithm also keeps track of the number of successful local improvements (i.e., applications of the 2-opt operator that make the tour length smaller) and the number of experiments that got aborted (i.e. those experiments wherein the performance did not reach below 10% of optimal performance even after many generations). Figure 4 shows the graph for 10 processors. Similar graphs were obtained for different number of processors. This shows that if we have fewer trials per generation then the total effort required to get a relatively good performance is less. (As mentioned before, when *tpg* tends to infinity we have the independent strategy, and that clearly requires a lot more effort).

But not all trials involve successful local improvements. Those that do not perform any local improvement will take very little processor time because the tour does not need to be rearranged. Therefore, to get an idea of total time required, we plotted in Figure 5 the number of successful local improvements done on average. Again, we see that doing fewer trials per generation reduces the total time required.

There is a danger in doing too few trials per generation however. As Table I shows, out of 50 experiments for each value of *tpg*, some get aborted for low values of *tpg*. This happens when the algorithm gets caught in a local optimum, which occurs because the algorithm described above has no means of maintaining diversity of structures for low values of *tpg*. Note that for higher values of *tpg*, that is strategies closer to the independent strategy, no experiments get aborted, showing the robustness of the operator and the algorithm being used.

To avoid getting caught in a local optimum we decided to change the interdependent strategy slightly to increase the diversity of examined structures. Instead of taking the one best structure we decided to experiment with the redistribution of the k best structures after each generation. (Again as k tends to n we have the independent case). In the experiments with 10 processors we found that with $k = 4$ only 1 or 2 experiments get aborted (out of 50) for lower values of *tpg*. We may conclude that increasing k (i.e. maintaining diversity) makes the algorithm more robust.

There is another price we pay for exchanging information too quickly. Every time we exchange information a copy time is involved and this copy time increases as the number of trials per generation decreases. But it will decrease as k increases (for a fixed value of *tpg*)†. Thus, our experiments show that one obtains a good interdependent strategy by keeping *tpg* as low as possible to decrease the number of total trials and to use a large enough k to make the algorithm avoid getting trapped into a local minima as well as to reduce copy time.

4. Experiments using a Genetic Algorithm

Genetic Algorithms (GA) [3], introduced by Holland, have been applied with good success to function optimisation problems involving complex functions [6] as well as on some combinatorial optimisation problems [7,8,9,10,11,12,13]. A typical GA maintains a population of structures after each.

† It should be noted that on a parallel machine the copy time can be reduced.

generation, which consists of applying local improvement operator a certain number of times (trials) to each structure, uses a selection mechanism to produce a new population of structures for the next generation. (Thus, a trial involves a probabilistic operation that makes a small local change to the structure). The selection mechanism assigns a strength of performance measure to each structure. This measure is usually the ratio of the performance of the structure to the average performance of the population. The number of occurences of this structure in the next generation is proportional to this performance measure ‡.

A genetic algorithm can be viewed as a parallel search algorithm of the type we have been discussing in the previous sections. Whereas a sequential algorithm (using the local improvement operator) operates on one structure a GA operates on a population of n structures. If one imagines that each structure is worked on by a separate processor, the selection mechanism can now be viewed as an interdependent strategy. The n processors run separately for a while (a generation) and then exchange information (about performance) resulting in processors getting a (possibly) different structure for the next generation. This strategy is a more sophisticated version of our earlier idea which consisted of taking the k best structures from the current population.

We used a version of the genetic algorithm (GA) that uses the 2-opt local improvement operator to solve the TSP. We ran the GA for 50,000 trials, varying tpg. The population size, or alternatively the number of processors, is 50.

On the lattice of 100 points (optimum length is 100) the GA did not do well for small values of tpg (5 and 10) but did almost equally well on higher values of tpg. Table II shows the effort required for the GA to get to within 10% of the optimal performance. It should be noted that these values are close to the values obtained in our earlier interdependent strategy of taking the one best (Table I).

Next we chose the domain to be 100 points uniformly distributed between (0,0) (0,1) (1,0) (1,1). Figure 6 shows the performance versus tpg curve. The best performance is seen at about $tpg = 500$ and it worsens a little after that. Therefore using the selection mechanism of a GA as strategy of interdependency, and keeping tpg relatively small yields a fast and robust algorithm with good performance.

5. Conclusion

We have given evidence that probabilistic sequential search algorithms which operate by performing a local change on a structure to generate a new structure, can be parallelised by doing a reasonably large amount of local improvements for each structure per generation and then exchanging information about "good" structures. Doing too few trials per generation, however, may not yield good performance values as premature convergence may occur. On the other hand, we also showed that taking one best structure and redistributing it over the other processors is too simplistic a strategy since it also causes premature convergence; hence a few best should be selected for redistribution. In fact, it turned out that the more sophisticated interdependent strategy, the selection procedure of a genetic algorithm, gave the resulted in the most robust strategy. It should be noted that we have ignored copy time in the presentation of the results. We believe however, that even if we take this overhead into account similar results concerning the independent versus interdependent strategy may be obtained.

‡ Most GAs also use a crossover operator that takes two structures and interchanges parts of them to produce two new structures. In this paper we ignore such operators.

Acknowledgements

We wish to thank the referees for helpful comments which helped in clarifying some of the ideas and results of the paper.

References

1. S. Lin and B.W. Kernighan, "An Effective Heuristic Algorithm for the Traveling Salesman Problem", *Operations Research* 1972, pp. 498-516.
2. E. L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys (ED), *The Traveling Salesman Problem*, John Wiley and Sons Ltd (1985)
3. J.Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.
4. N.L. Johnson, S. Kotz, *Urn Models and their Applications, Wiley and Sons*, 1977
5. V.F.Kolchin, B.A.Sevastyanov, V.P.Chistyakov, *Random Allocations*, V. H. Winston and Sons, 1978.
6. K. A. Dejong, "Adaptive system design: a genetic approach", *IEEE Trans. on System, Man and Cybernetics, Vol SMC-10(9)*, pp 556-574 (Sept 1980).
7. J.J. Grefenstette, R. Gopal, B.J. Rosmaita and D. Van Gucht, "Genetic Algorithms for the Traveling Salesman Problem", *Proc. of an Int'l Conf. on Genetic Algorithms and Their Applications*, pp 160-168 (July 1985).
8. J.J. Grefenstette, "Incorporating Problem Specific Knowledge into Algorithms", To appear.
9. J.Y. Suh, D. Van Gucht, "Incorporating Heuristic Information into Genetic Search", Technical Report, Indiana University, February 1987
10. L. Davis, "Job shop scheduling with genetic algorithms", *Proc. of an Int'l Conf. on Genetic Algorithms and Their Applications*, pp 136-140 (July 1985).
11. M.P. Fourman, "Compaction of symbolic layout using genetic algorithms", *Proc. of an Int'l Conf. on Genetic Algorithms and Their Applications*, pp 141-153 (July 1985)
12. D.E. Goldberg and R. Lingle, "Alleles, loci, and the traveling salesman problem", *Proc. of an Int'l Conf. on Genetic Algorithms and Their Applications*, pp 154-159 (July 1985)
13. D. Smith, "Bin packing with adaptive search", *Proc. of an Int'l Conf. on Genetic Algorithms and Their Applications*, pp 202-206 (July 1985).

TABLE I

tpg	total effort	success rate	total success	experiments aborted	variance
5	1505	0.030	45.009	5	0.092
10	1820	0.034	61.320	3	0.123
20	2160	0.036	77.385	3	0.093
40	2640	0.038	99.261	1	0.107
80	3360	0.035	117.112	1	0.110
160	4160	0.034	140.610	0	0.157
320	5440	0.030	160.867	0	0.245
640	6400	0.027	172.721	2	0.590
1280	7680	0.022	170.522	2	0.922
2560	10240	0.019	192.836	2	1.857
5120	15360	0.015	222.767	0	2.114

tpg - trials per generation
total effort = tpg * (generations to get to within 10%)
success rate = fraction of trials successful
total success = total number of successful improvements done
experiments aborted are out of total of 50 for each tpg

TABLE II

tpg	total effort
5	1500
10	2058
20	2812
40	3760
80	4480
160	5600
320	6080
640	7040
1280	7680
2560	7680
5120	10240

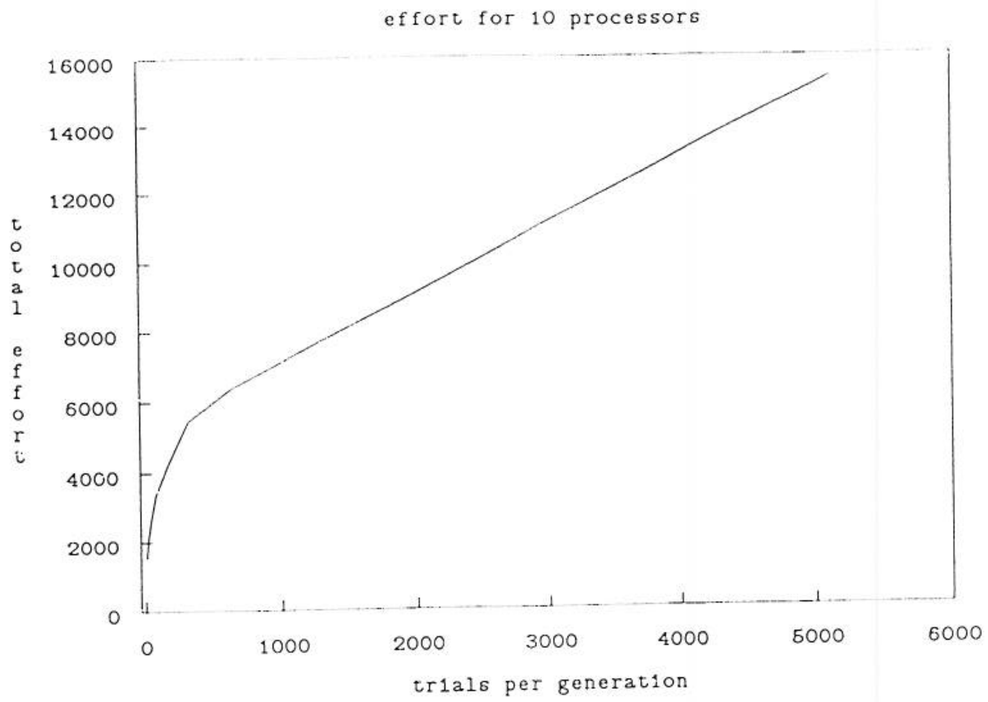

Figure 4

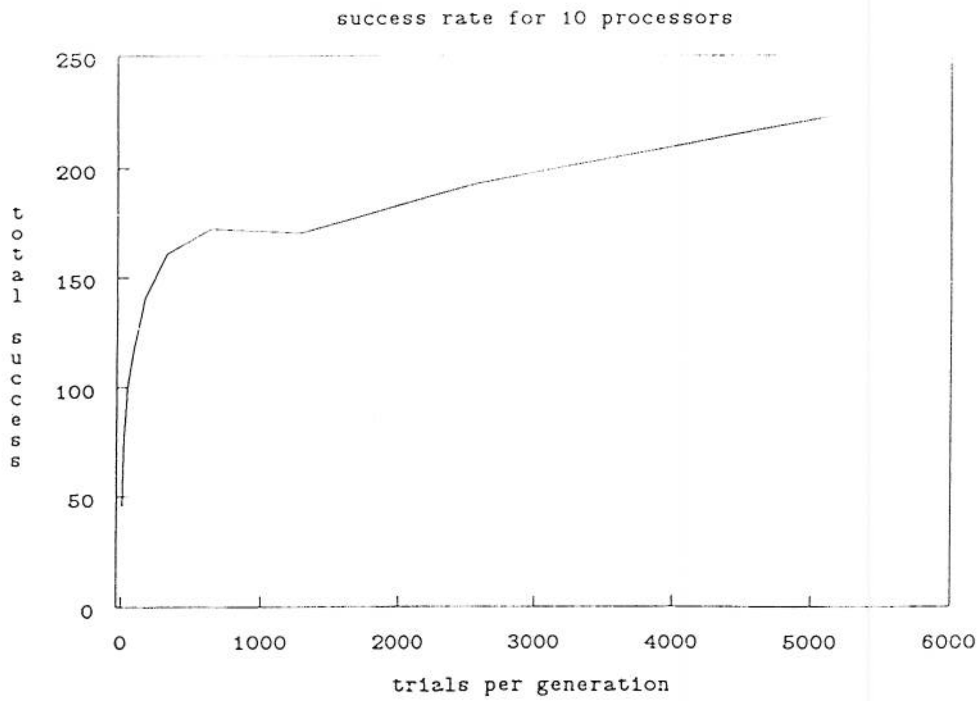

Figure 5

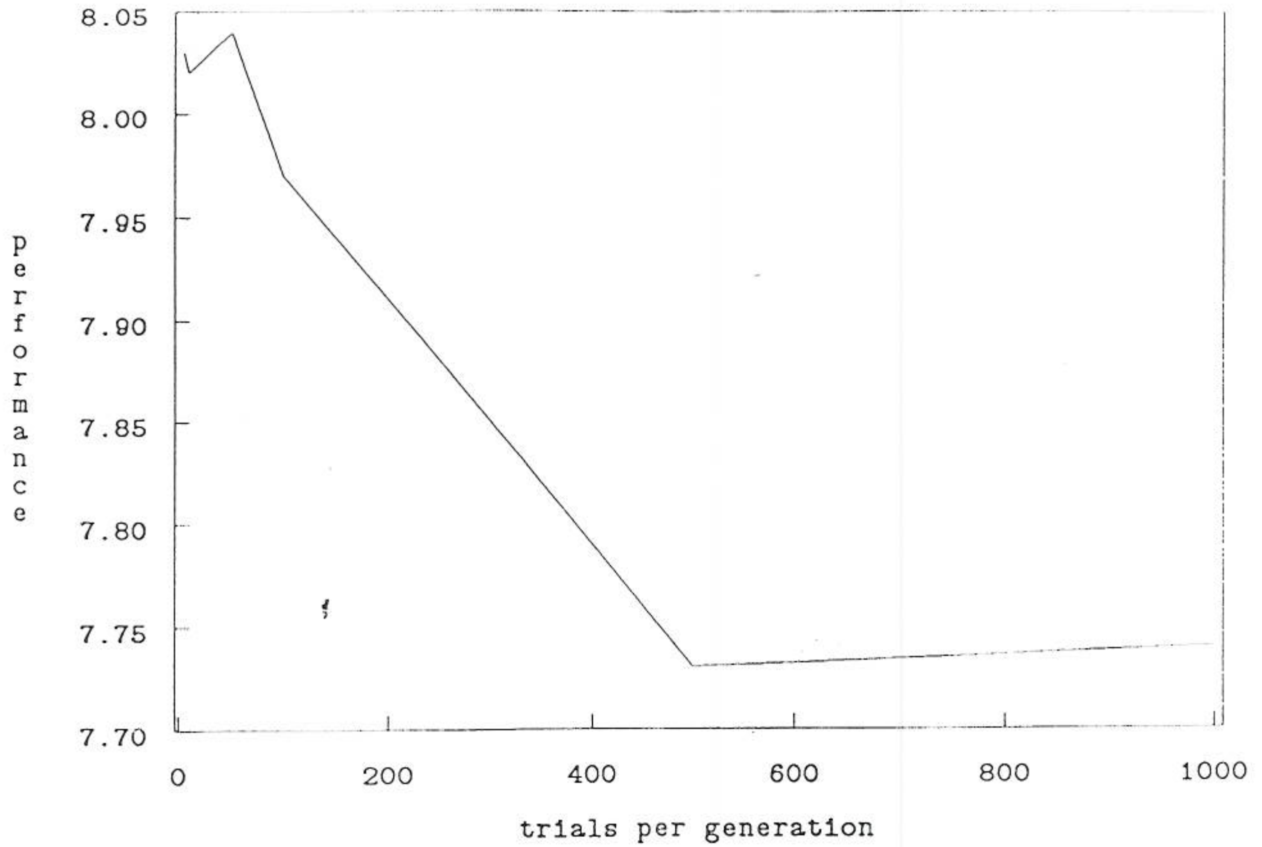

Figure 6