DESIGN AND IMPLEMENTATION OF A PARALLEL GENETIC ALGORITHM FOR THE TRAVELLING PURCHASER PROBLEM

Luiz S. Ochi, Lúcia M. A. Drummond
Pós-Grad. Ciência da Computação, UFF, R. São Paulo, 24040-110, Niterói-Brazil
e-mail:{satoru,lucia}@dcc.uff.br

Rosa M. V. Figueiredo
COPPE-Sistemas/UFRJ CP.8511, 21945-970, RJ-Brazil
e-mail:rosam@cos.ufrj.br

Key words: Genetic Algorithms, Parallel Processing.

ABSTRACT

Nowadays genetic algorithms stand as a trend to solve NP-Complete and NP-Hard problems. In this paper we present the design and implementation of a parallel genetic algorithm - GENEPAR applied to a Job Sequencing Problem, more specifically the Travelling Purchaser Problem, which is considered as a NP-Hard problem. The parallel algorithm proposed is based on the island model and consequently implemented on a distributed memory MIMD computer. In order to analyze GENEPAR, a sequential heuristic was also implemented on the same environment using only one processor. Experimental results show that GENEPAR exhibits good performance in terms of time, compared to that other heuristic. It also improves the search quality.

1 Introduction

Genetic algorithms have proven to be very effective approaches to solving various hard problems in optimization [2] [4]. However, some of these algorithms may require a large amount of time to find solutions. Recent advances in technology and the increasing availability of parallel computers have motivated research on parallel genetic algorithms in order to reduce excessive execution time [7] [8].

In this paper we present a new parallel genetic algorithm called GENEPAR which solves a Job Sequencing Problem known as Travelling Purchaser Problem (TPP).

Although TPP has been used in many applications such as scheduling and routing problems, it has not been extensively studied in related literature. TPP was originally developed by Ramesh [6] who proposed a method based on a lexicographic search procedure. Golden, Levy and Dahl [3] proposed a heuristic based on saving and insertion concepts, but the authors did not present computational results. Ochi, Santos, Montenegro and Maculan [5] proposed a genetic algorithm, GENE2, based on crossover operators. The operator used in GENE2 introduces concepts of the nearest neighbourhood and optimization in the processing time to generate a new offspring. This operator always produces feasible offsprings without requiring aditional rules to code and decode chromosomes.

There are different ways to parallelize genetic algorithms. The generally used classification [7] divides parallel genetic algorithms in three categories: island, fine grain and panmictic models. The parallel genetic algorithm proposed is based on the island model. Consequently, it is implemented on a distributed memory MIMD computer.

The remainder of this paper is organized as follows. Section 2 describes the Travelling Purchaser Problem. The parallel algorithm proposed is described in Section 3. In sections 4 and 5 we present GENEPAR and its implementation details. In Section 6 we compare GENEPAR to GENE2 presenting the parameters used in our experiments, the search quality and execution time results. Finally Section 7 concludes the paper.

2 TPP Description

To describe the TPP we need the following data:

- A set of n markets $N = \{0, 1, 2, \cdots, n\}$ plus a source $s = \{0\}$;
- A set of m items $K = \{1, 2, \cdots, m\}$ to be purchased at n markets;
- An array $D = (d_{kj})$ such that $k \in K, j \in N - \{0\}$, where d_{kj} is the cost of item k at market j;
- An array $C = (c_{ij})$ such that $i, j \in N$, where c_{ij} is the cost of travel from i to j.

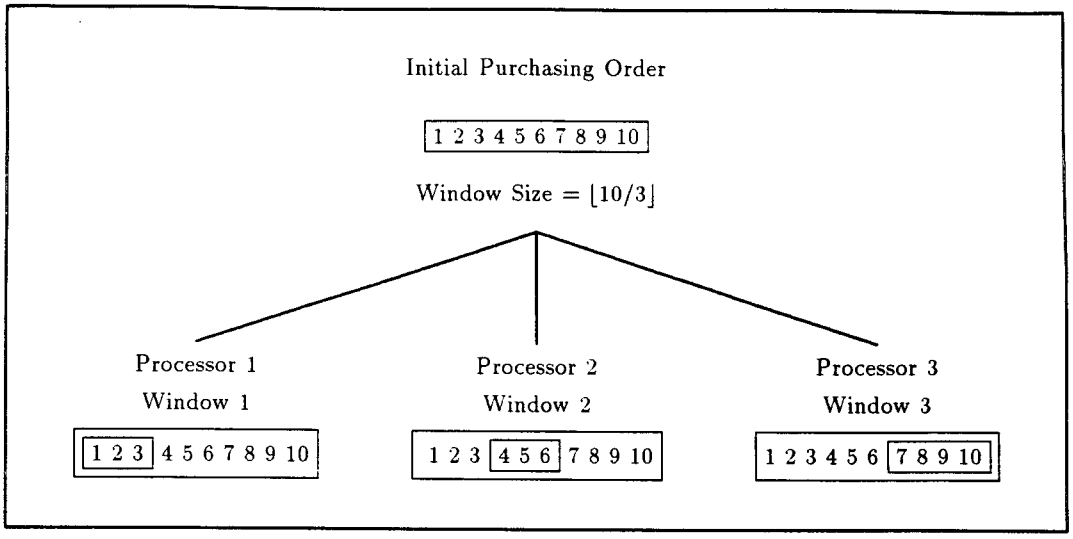

Figure 1: Initial permutation distribution

It is assumed that each item is available in at least one market $j \in N - \{0\}$; in source $s = \{0\}$ no item can be purchased; the traveller may pass through a market any number of times without purchasing an item there; and the traveller may purchase as many items as there are available at each market.

We define a complete directed graph $G = (N, A)$, without loops, where N is the already defined set of markets and each arc $(i, j) \in A$ represents a link from i to j . An array (d_{ki}) such that $k \in K$ is associated with each vertex (market) $i \in N$; that is, the cost of item k in a vertex i. The cost of travel from i to j is given by c_{ij}, such that $(i, j) \in A$.

The aim of the TPP is to obtain a directed cycle including source s and passing through a subset $J \subseteq N$ such that the total of travel and purchase costs is minimized.

TPP is classified as NP-Hard and can be seen as a generalization of the Travelling Salesman Problem (TSP)[6].

3 The parallel genetic algorithm for TPP

The proposed parallel genetic algorithm for TPP, GENEPAR, is based on the island model [7]. It consists of a distributed algorithm in which the population is partitioned into several subpopulations which evolve in parallel and periodically migrate their individuals among themselves. The island model is used in coarse grained parallel genetic algorithms and it is usually implemented on distributed memory MIMD computers.

GENEPAR is composed of the following stages described in the subsections below: generation of the initial purchasing order, purchasing order updating, population partitioning, nucleus of genetic algorithm and migration of individuals.

3.1 Generation of the initial purchasing order

The genetic algorithm for TPP works with several purchasing orders of a set of m items, called permutations. We develop a special mechanism which generates purchasing orders.

The first purchasing order is generated in each executing process through a procedure proposed by Golden, Levy and Dahl [3] for the TPP problem. Thus, the initial purchasing order is defined as follows:

Step 1 - Find the item which may be purchased in the largest number of markets. Insert this item in a list which holds the inital purchasing order.

Step 2- Insert item k in the list if it still does not belong to the purchasing order, where $c_{ji} + d_{ki}$ is the minimum, i is the market where item k is available and j is the market where the last item of the list was purchased.

Step 3- If the list does not contain all items go to step 2, otherwise, stop.

3.2 Purchasing order updating

The initial purchasing order of items, also called initial permutation, is distributed among all processors and an initial population is generated for each one of them individually. We use a strategy which avoids different processors to work with the same permutation so that the algorithm goes over the largest possible part of search space, thus covering more local minima.

Each processor i works on its initial permutation and generates other permutations changing only a segment s_i of the initial permutation (called window s_i).

Hence, one permutation is partitioned into several disjoint windows, where each window is

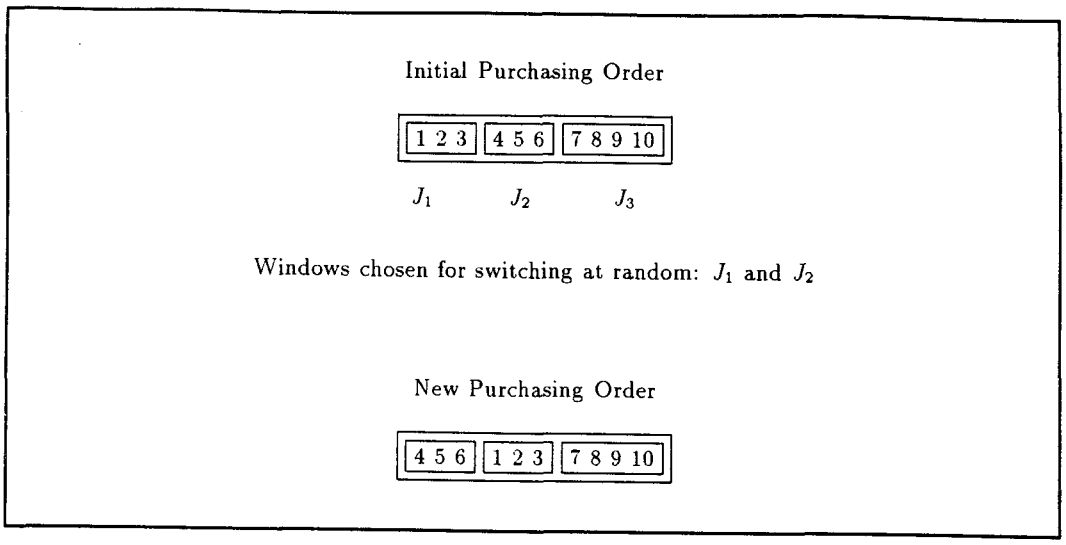

Figure 2: Window switching

associated with a processor and also composed of a fixed number of elements of the permutation. A processor can only change those elements associated with its window. The window size, except for the last one, is limited to [number of items/number of processors]. Figure 1 illustrates the initial permutation distribution and the corresponding window of each processor in the case of ten items and three processors.

In every process the permutation updating procedure is the same. Each one generates other purchasing orders from the initial one, changing the permutation in its window and retaining the remaining part. Although each process can generate (window size)! purchasing orders, this number may be limited in order to prevent the algorithm from becoming exhaustive.

We also propose a procedure called window switching which causes a change of greater impact in the permutation. Each process executes the window switching procedure as follows:

- Select two windows, J_k and J_l.
- Switch these windows generating a new permutation.

Figure 2 illustrates this mechanism.

3.3 Population Partitioning

In GENEPAR, the initial population partitioning is slightly different from the one usually accomplished in the island model where the population is simply partitioned into small subpopulations. Virtually, in GENEPAR, each processor generates an initial subpopulation from different purchasing orders. Thus, each processor goes over a different region of the search space. In Figure 3, each process generates a population based on the initial purchasing order and executes a permutation of the elements of its window in order to generate a new purchasing order. For each purchasing order a new population is also generated.

3.4 Nucleus

The nucleus of GENEPAR executes three steps: selection, crossover and mutation. These steps, as in GENE2 [5], are based on the closest neighbour criterium and minimization of item cost in the offspring.

The nucleus of GENEPAR, in population renovation, is based on the operators used in GENE2. Thus, x parents generate $x/2$ children. To each generated child, the worst parent is analyzed in order to check if the child is more fit than that parent. If so, the child replaces the parent, else the number of substitutions of parent for child in this generation is compared. If it is below a specific number, there is a switch even if the offspring is worse than the parent. This specific number which sets the probability for a "bad" child to be accepted has been tested by Ochi, Santos, Montenegro and Maculan [5] and defined as number of generations $\times$ 0.05.

A 2-optimal tuning and a migration operator were added to the basic algorithm. The 2-optimal heuristic is applied to the best individual generated by GENE2 operators. The best offspring generated is compared to the best individual of all previous generations. If the difference between them is less than 20% then a 2-optimal is applied in order to improve the solution. The migration operator is described in the next section.

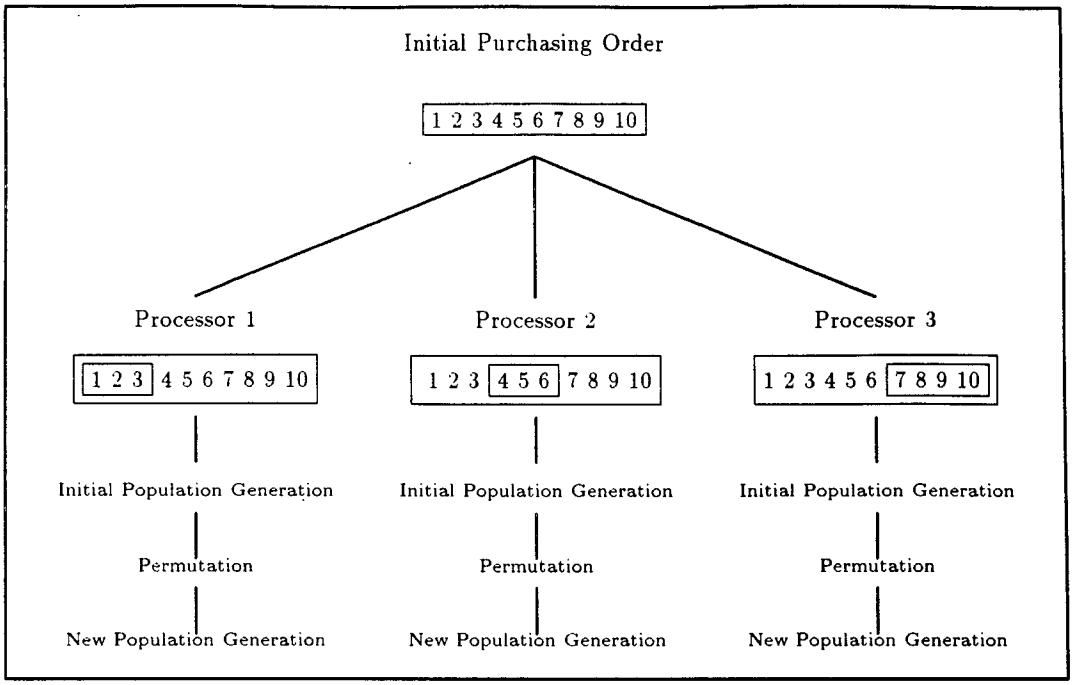

Figure 3: Population generation

$p_L = (3\ 2\ 2\ 5\ 4\ 2\ 3\ 5)$

$p_M = (2\ 2\ 5\ 5\ 3\ 4\ 4\ 4)$

$k_L = (\mathbf{2}\ 1\ 3\ 4\ 5\ 6\ 7\ 8)$ with window J_1

$k_M = (1\ 2\ 3\ 4\ \mathbf{6}\ \mathbf{5}\ 7\ 8)$ with window J_3

Figure 4: Reproduction process in migration

$O = (3\ 2\ X\ X\ 4\ 2\ X\ X)$

Figure 5: Offspring generated by migration

3.5 Individual migration

The migration operation is triggered before a new initial purchasing order is generated by the window permutation process. Each process sends its best solution to its neighbour subpopulations. Since the processors are equally distant in the parallel machine used for tests (SP/2), each subpopulation allocated in a different processor receives (sends) individuals from (to) all other subpopulations.

In GENEPAR, the migration operation has a peculiarity: the migrated solutions do not take part of the receptor process population because of their different purchasing orders. In fact, a recombination process between the best local solution and each migrated solution takes place. The local one must be replaced by each better resulting offspring. This peculiarity became necessary because in TPP distinct process solutions need to have distinct purchasing orders, which make it impossible for them to be part of the process population they moved to. However, "crossbred" offspring may be generated, since their characteristcs allow them to be part of the population one of their parents moved to. This is so, because purchasing orders of distinct processes have similar characteristics which means that they differ in only two windows from each permutation.

A migration operator that takes this fact for granted generates only one offspring keeping its local parents characteristcs in parts which differ from the migrated parent and recombines ordinary parts using the minimal path algorithm.

Consider the example in Figure 4 where 8 items can be purchased in 5 markets with the data below:

- local parent p_L = better solution obtained by local process with permutation k_L.
- migration parent p_M = better solution obtained by one of (number of processes-1) with the permutation k_M.

If k_L and k_M are compared it can be observed that they only differ in the positions referring to windows coming from each parent, where each offspring will have the same local parent solutions, as well as it will inherit its purchasing order k_L, as shown in Figure 5.

The positions left will be chosen the same way used by the chromosome operator of GENE2, described in detail in [5].

4 GENEPAR

The stages of GENEPAR are described next. Notation:

- $P(t)$ = population in stage t
- S_t^* = best solution of population P(t)
- S^* = best solution so far
- $T1$ = minimum{10, 10% of number of items}
- $T2$ = minimum{5!, (window size)!}
- $STOP$ = parameter based on the criteriom used in [5]
- GENE2 = operator used in [5] to generate $P(t)$ from $P(t-1)$

Using the previously presented definitions our genetic algorithm can be described as follows:

Procedure GENEPAR
  Initialization:
    Generate initial permutation
    Calculate number of windows
  While $k_1 \leq T1$ **do**
    While $k_2 \leq T2$ **do**
      Generate initial population
      Evaluate population
      Select S^*
      Renovate population:
        $t := 0$
        While not $STOP$ **do**
          $t := t + 1$
          Apply GENE2
          Evaluate $P(t)$
          Apply 2-optimal
          Select S_t^*
          If $S_t^* < S^*$ **then**
            $S^* := S_t^*$
        End While
      Generate new permutation
      $k_2 := k_2 + 1$
    End While
    Do migration
    Switch windows
    $k_1 := k_1 + 1$
  End While
  Select best global solution
End

5 Implementation details

The parallel genetic algorithm is composed of two programs: the Master and the Slave. The master program is basically responsible for:

- Reading the name of the file which has the arrays of cost $C = (c_{ij})$ and $D = (d_{kj})$.
- Initialization of slaves in a specific number of processors which will be run in parallel.
- Generation of pair windows to be switched.

After triggering the slaves, the master keeps awaiting the best solution obtained among them. After getting a final solution, the master calculates the execution time of the algorithm and displays the final solution. The master is left to do window permutation so the slaves will switch the same windows in the purchasing order.

6 Experimental results

Our experiments were run on an 8-processor SP/2 using the PVM system [1] for parallelism.

Although TPP is a problem in practical use, few papers in this area present heuristic algorithms for its solution. One of them has been developed by Golden, Levy and Dahl [3]. A more recent approach, GENE2 [5] performed better when compared to the first. Thus, in order to analyze GENEPAR, GENE2 was also implemented on the same environment using only one processor.

Initially, TPP was analyzed with $20 \leq n \leq 500$ and $20 \leq m \leq 500$ where n is the number of markets and m is the number of items. An average of results is shown in the figures 6 and 7. They present four cases where for each one of them 20 instances were run and: (a) $20 \leq n, m \leq 50$; (b) $20 \leq n \leq 50$ and $100 \leq m \leq 500$; (c) $100 \leq n \leq 500$ and $20 \leq m \leq 50$; (d) $100 \leq n, m \leq 200$.

Figure 6 shows that GENEPAR achieved the best results in terms of cost for the cases (b), (c) and (d). The Figure 7 shows that GENEPAR exhibits the best performance in terms of time for the case (b) where the number of items is high.

7 Concluding remarks

Our results so far show significant advantages for GENEPAR compared to GENE2, not only with respect to the running time but also with respect the search quality.

The great saving in time of execution is an important indication that problems with high dimension should likely be solved through parallel algorithms.

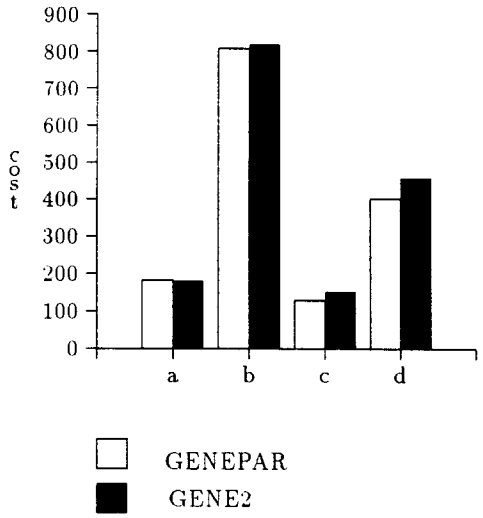

Figure 6: Average cost results

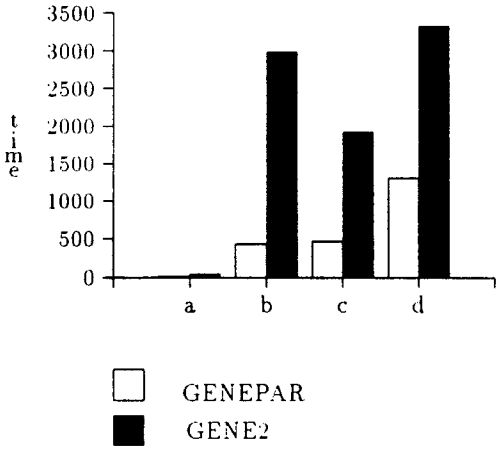

Figure 7: Average time results in seconds

Experimental tests should continue and theoretical analyzes should be conducted for the optimization of the following parameters: migrated subpopulation size, frequency of migrations and determination of subpopulation neighbourhood.

Acknowledgments

We would like to thank the LNCC (Scientific Computation National Laboratory) team for their help with the parallel platform SP/2. This work was supported by the brazilian agency PROTEM/CNPq.

References

[1] GEIST, A.; BEGUELIN, A.; DONGARRA J.; JIANG, W.; MANCHEK, R. and SUNDERAM, V.,1994, *PVM: Parallel Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing*, The MIT Press.

[2] GOLDBERG, D.E.,1989, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley.

[3] GOLDEN, B.; LEVY, L. and DAHL, R., 1981, Two Generalizations of The Travelling Salesman Problem, *OMEGA The International Journal of Management Science*, 19, pp. 439–445.

[4] MICHALEWICZ, Z., 1992, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer Verlag.

[5] OCHI, L.S.; SANTOS, E.M.; MONTENEGRO, A.A. and MACULAN, N., 1995, A New Genetic Algorithm for the Travelling Purchaser Problem, *Proceedings of the Metaheuristics International Conference*, pp. 52–57.

[6] RAMESH, T., 1981, Travelling Purchaser Problem, *Opsearch*, 18, pp.78–91.

[7] RIBEIRO, J. L., 1995, *An Object-Oriented Programming Environment for Parallel Genetic Algorithm*, Ph.D. Thesis, Department of Computer Science, University College London.

[8] TANESE, R., 1989, Distributed Genetic Algorithms, *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 434–439.