Parallel Genetic Algorithms on a GPU to Solve the Travelling Salesman Problem

Leopoldo Noel Gaxiola Sánchez, Juan José Tapia Armenta, Víctor Hugo Díaz Ramírez

Instituto Politécnico Nacional, CITEDI
Ave. del Parque 1310, Mesa de Otay, Tijuana, B.C., 22510, México.
lgaxiola@citedi.mx, jtapiaa@ipn.mx, vdiazr@ipn.mx

2014 Published by $\mathcal{DIFU}_{100}ci@$ http://nautilus.uaz.edu.mx/difu100cia

Abstract

The implementation of parallel genetic algorithms on a graphic processor GPU to solve the Travelling Salesman Problem instances is presented. Two versions of parallel genetic algorithms are implemented, a Parallel Genetic Algorithm with Islands Model and a Parallel Genetic Algorithm with Elite Island; the two versions were executed on a GPU. In both cases, each individual is represented by a thread, and each island is represented by a block of threads. The main feature of the Parallel Genetic Algorithm with Elite Island in this work is that there is not migration between islands, instead, an Elite Island is created with the best individuals from each of the islands to share the best individuals. The individual with minimal fitness function is the sought solution. The results show that the Elite Island model is better than the island model with migration of individuals.

Keywords: GPU, Island Model, Parallel Genetic Algorithms, Travelling Salesman Problem

1. Introduction

The Travelling Salesman Problem (TSP) is a typical representative of a large class of problems known as combinatorial optimization problems. The study of combinatorial optimization problems is of great importance from a theoretical and practical point of view. These problems are presented in real-world situations, such as routing and scheduling, belonging to NP-complete and NP-hard problems. Among them, TSP is one of the most important, since it is very easy to describe, but has an extremely large search space and is very difficult to solve, it is probably the most studied combinatorial optimization problem and has become a standard testbed for new algorithmic ideas. Genetic Algorithms (GAs) are efficient search methods based on the biological principles of natural selection and genetics. They are being applied successfully to find acceptable solutions to problems in business, engineering, and science [1]. As occurs in nature, GAs are based on the survival of the fittest individuals in a population. The GA starts with an initial population that evolves from one generation to another, through the creation of new individuals with better fitness values and elimination of individuals with low fitness values. In GAs the populations evolves by applying genetic operators such as

selection, crossover and mutation, whose functionality and implementation depends on the problem to solve. One of the main features of the genetic algorithm is its ease of parallelization, since they are based on populations of independent individuals, thereby calculating the fitness function and the results of genetic operators of an individual not depend on the calculation of other individuals [2].

Typically the initial population is generated randomly with an uniform probability distribution. The initial population size is important because it influences whether the GA can find good solutions and the time it takes to reach them. If the population is too small, it may not be an adequate sample of the search space, and it will be difficult to identify good solutions [3]. If the population is too large, it is necessary to use a lot of computational resources and processing time. In each iteration of the GA a new population of individuals is created on the basis of their predecessors, having more chance to reproduce those with better fitness function. GAs are generally able to find good solutions of combinatorial optimization problems in a reasonable time, but as applied to the hardest and biggest problems an increase in the time required to find adequate solutions occurs. Consequently, there have been many efforts to implement faster GAs, and one of the most promising alternatives is to use parallel implementations, which can obtain a substantial reduction of processing time. In recent years the development of powerful graphics processors has had a major boost, as a result we may have computing platforms with high performance and low cost.

While the first implementations of parallel GAs were designed for multi-core CPUs, clusters, and grids (e.g. ParadisEO [4]), with current Graphics Processing Units (GPU) there exist another parallel architecture, offering a high degree of parallelism and huge amounts of computational power. As a consequence, it seems natural to employ GPUs for parallel GAs. Most of the work up to now focuses on implementing GAs on older GPUs; even using the graphics hardware of previous generations, some claim to achieve speedups higher than 1000 when comparing their GPU implementations to normal CPUs. However, Speedups of this magnitude can only be achieved by comparing optimized GPU code to poor CPU implementations [5].

In this paper two versions of parallel genetic algorithms were implemented on a GPU to solve the TSP, a Parallel Genetic Algorithm with Island Model (PGAIM) and a Parallel Genetic Algorithm with Elite Island (PGAEI), those already were used in [6] to solve the problem of approximation of NURBS curves to a set of points of a medical image. A comparison between them is done in terms of the best solution and time.

2. Traveling Salesman Problem

The TSP is to find a Hamiltonian tour of minimal length on a fully connected graph. The TSP is a well-known NP-complete problem, thus, there is no polynomial algorithm to find the optimal result [7]. Heuristic algorithms for the TSP can be broadly divided into two classes: tour construction procedures, which build a tour by successively adding a new node at each step, for example Ant Colony Optimization [8]; and tour improvement procedures, which start from an initial tour and seek a better one by iteratively moving from one solution to another, according to adjacency relationships defined by a given neighborhood structure, for example GA, this approach was used in this work. The goal in the TSP is to find a minimum length Hamiltonian tour, where a Hamiltonian tour is a closed path visiting each of n nodes of a graph G exactly once. Thus, an optimal solution to the TSP is a permutation of the nodes index $1, 2, \ldots, n$ such that the length $f(\pi)$ is minimal, where $f(\pi)$ is given by

$$f(\pi) = \sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)} + d_{\pi(n)\pi(1)} \qquad (1)$$

where $d_{\pi(i)\pi(i+1)}$ is the distance between nodes i and $i+1$ and $d_{\pi(n)\pi(1)}$ is the distance between node n and the first node.

In general, the TSP includes two different kinds of problems, the Symmetric TSP and the Asymmetric TSP. In the symmetric form there is only one way between two adjacent cities, i.e., the distance between cities A and B is equal to the distance between cities B and A, while in the asymmetric form there is not such symmetry and it is possible to have two different distances between two cities. The number of tours in the ATSP and STSP on n vertex (cities) is $(n-1)!$ and $(n-1)!/2$, respectively.

3. Parallel genetic algorithms

Parallel genetic algorithms arise from the need of computation required for extremely complex problems whose running time using sequential genetic algorithms is a limitation [9]. The implementation of parallel genetic algorithms aims to break a problem into several subproblems and solve them simultaneously on multiple processors, which improves the performance and quality of search of genetic algorithms. In general, the behavior of parallel algorithms is the same as in the sequential algorithms. However this is not necessarily the case

of the parallel genetic algorithms. In the structure of a task, parallel genetic algorithms can be divided into subtasks so that there is a balanced distribution of activities; members of the populations of a genetic algorithm can be divided into sub-populations that are distributed on different processors through communication and control mechanisms that help to generate a solid structure at the level of genetic algorithms in a parallel environment. There are several ways to parallelize a genetic algorithm. The first and most intuitive is the global, which is basically to parallelize the evaluation of the fitness function of individuals holding a single stock. A better option for parallelization of genetic algorithm is to divide the population into subpopulations that evolve separately and exchange individuals every few generations [9]. The following describes each of the stages of a genetic algorithm.

3.1. Individual representation

An important consideration that must be taken into account when designing a GA is to define a representation for individuals in the population that model the solution of the problem [2]. Individuals are represented by the chromosome that contains a gene vector symbolizing the data we want to optimize. In this work each individual represents a tour.

3.2. Fitness Funtion

The fitness function determines the potential of the solution that individuals have within the population [2]. In this work, the cost function which represents the fitness function is given by Eq. (1) and the goal of the genetic algorithm is to minimize this cost function.

3.3. Genetic operators

3.3.1. Selection operator

Selection is the process of choosing two parents from the population for crossover [2]. The selection operator produces new points in the search space, it determines which individual will leave offspring for the next generation. There are various selection methods used in GAs, such as roulette selection, tournament selection, and fitness proportional. In this paper a tournament selection is implemented, in which randomly chooses a small sample of the population and it is selected the individual with best fitness value. For PGAIM an individual from the same island is selected and in the case of the PGAEI an individual from same island or Elite Island is selected. In both cases only one individual is selected to be crossed and the other individual is defined by the thread, because in both algorithms each individual is assigned to a thread on the GPU.

3.3.2. Crossover operator

The genetic crossover operator is used to improve individuals of the population. To perform the crossover operation, two individuals are selected to be combined and the resulting individual has the ability to replace one of the parents or the individual with the worst fitness in the population [2]. The new individual will replace a parent provided that have better fitness function. If the new individual is not better than parents can try to replace the worst individual in the population. The process is aimed at the sub-regions of the search space, where it is assumed that there is an better solution. There are several ways to apply the crossover operator such as crossing a single point, multi-point crossover, uniform crossover, among others. In this work, each individual in the population makes cross with another individual chosen by the selection operator in each iteration and cross used by two points, where two positions in the chain of genes on the chromosomes of both parents are selected randomly. One parent provides information that is outside the range between the two positions and the other provides the information that is provided in the two positions.

3.3.3. Mutation operator

The mutation operator creates an individual performing some type of alteration, usually small, on an individual of the population chosen randomly. The mutation is intended to disperse the search algorithm as it gives rise to individuals with new genetic material. In this paper, the process of mutation used is by inversion where two positions of individual chromosomes are randomly selected and chromosome genes of the individual that fall into these two positions are inverted.

4. Implementation of parallel genetic algorithms on a GPU

This section described in detail the two versions of parallel genetic algorithms that have been implemented on a GPU. Importantly, the number of threads is equal to the number of individuals because each individual in the population is mapped to a GPU thread and each island is mapped to a GPU threads block.

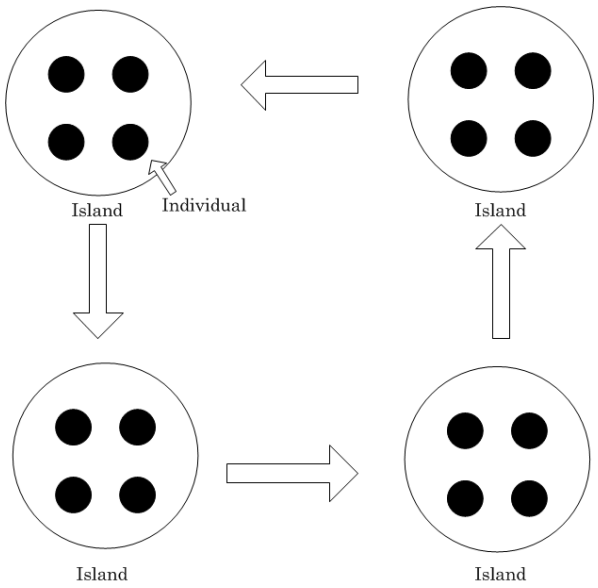

Figure 1. Parallel Genetic Algorithm with Islands Model.

4.1. Parallel Genetic Algorithm with Islands Model

On the PGAIM the population is divided into subsets [3]. The exchange of individuals is called migration, and is considered as a new genetic operator. In Fig. 1 is shown the organization of individuals in the population of PGAIM. The pseudocode for PGAEI is presented in Algorithm 1. The algorithm starts by allocating memory on the CPU and the GPU, with the next step the CPU calculates the cost matrix, after cost matrix is copied from CPU memory to GPU memory.

Then the initial population is generated by the kernel *IniPop* where each thread generates the individual will process. At the end of this kernel fitness function for each individual is evaluated, which is the sum of the distances. The individuals generated by *IniPop* kernel are stored in a variable called *I* that represents the old solution. The population then enters a loop that processes the PGAIM kernel, overall stopping criterion is determined by the number of iterations or when an individual achieves a certain value of the fitness function. In this work we chose to end when a certain number of iterations is reached. The PGAIM kernel first thing is to select the best population individual of each island in order to keep, select the worst of the next island and the best of the island replaces the worst of the next island, for it was necessary to assign two variables in shared memory one to store the fitness value of the best individual and another variable to store the thread identifier of the best individual, so that only individuals of an island may access these variables and keep isolated islands.

After that, an individual to cross is selected with tournament selection operator, which compares five individuals and selects the one with higher fitness value. As can be seen, all the threads come to make the crossing in each iteration. The crossover operator is then applied between the two parents. Subsequently, the mutation operator is applied. The individual generated by crossing is stored in the variable I_{new} representing the new population. If the thread between mutation also make the individual generated goes directly into the population without replacing old best individual. Finally, in the PGAIM kernel, generated individuals are evaluated and their fitness function if the individual of the new population has a better fitness than the individual of the old population is replaced. When the iterations terminate the old population is copied to the CPU. Then the best solution generated by the GA is obtained and finally the reports of CPU and GPU are released.

Algorithm 1 Pseudocode of PGAIM

```
main(){
  memory allocation on the CPU and the GPU
  calculate cost matrix
  copy of memory from CPU to GPU
  threads = IND_NUM
  blocks = ISLA_NUM
  IniPop<<<blocks,threads>>>()
  while condition_end==FALSE do
    PGAIM<<<blocks, threads>>>(parameters)
  end while
  copy of memory Population from GPU to CPU
  get best solution
  free memory on GPU and CPU
}
__global__IniPop(){
  PobGen(I)
  Evalutate(I)
  set the best indiviaul in each island
  NextIslandMigrate(I_{I+1})
}
__global__ PGAIM(){
  SetParents(I)
  I_new ← ParentsCross(I)
  Evaluate(I_new)
  if Fitness(I_new) < Fitness(I) then
    I = I_new
  end if
  I_new ← Mutation(I)
  set the best indiviaul in each island
  NextIslandMigrate(I_{I+1})
}
```

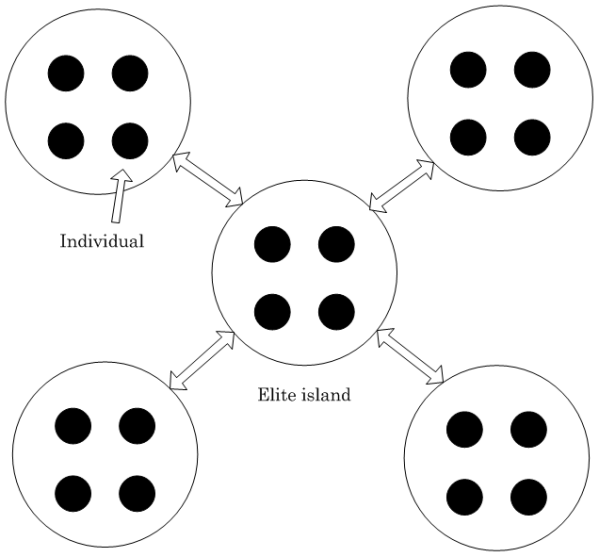

Figure 2. Parallel Genetic Algorithm with Elite Island.

4.2. Parallel Genetic Algorithm with Elite Island

In the PGAEI, instead of migrating individuals between islands, an island with the best individuals of each subpopulation, called Elite Island is built. This island of best individuals is shared with all the other islands. In Fig. 2 is shown the organization of individuals in the population of PGAEI.

The pseudocode for PGAEI is presented in Algorithm 2. The implementation of PGAEI in the GPU is very similar to the way in which the PGAIM was implemented. Begin with the kernel *IniPop* where the population is created, with the difference that after evaluating the fitness function of the best individual is selected each island and migrated to the Elite Island. Individuals generated in the kernel *IniPop* are stored in the *I* variable representing the older population. In addition there is a variable called Elite *E*, where there is a space allocated to store the best individual of each island. Then enters a loop that processes the kernel PGAEI, which ends when a certain number of iterations is reached. At the time of the crossing only a random individual of the same island *I* or Elite Island *E* is selected. Then generated individuals are evaluated with the fitness function. If the new individual in the population I_{new} have better fitness than the individual of the old population *I*, it is replaced. The mutation operator is applied similarly as in PGAIM, with the difference that all individuals make mutation and only enter the population if they are better than replacing the individual is then applied. At the end of the PGAEI kernel, the best individual of each island is selected and migrates to Elite Island. When the iterations end only the Elite Island CPU is copied. Then the best solution generated by the GA is obtained and finally the reports of CPU and GPU are released.

Algorithm 2 Pseudocode of PGAEI

```
main(){
  memory allocation on the CPU and the GPU
  calculate cost matrix
  copy of memory from CPU to GPU
  threads = IND_NUM
  blocks = ISLA_NUM
  IniPop<<<blocks,threads>>>()
  while condition_end==FALSE do
    PGAEI<<<blocks, threads>>>(parameters)
  end while
  copy of memory Elite Island from GPU to CPU
  get best solution
  free memory on GPU and CPU
}
__global__ IniPop(){
  PobGen(I)
  Evalutate(I)
  set the best indiviaul in each island
  EliteMigrate(E)
}
__global__ PGAEI(){
  SetParents(I, E)
  I_new ← ParentsCross(I, E)
  Evaluate(I_new)
  if Fitness(I_new) < Fitness(I) then
    I = I_new
  end if
  I_new ← Mutation(I)
  set the best indiviaul in each island
  EliteMigrate(E)
}
```

5. Results

SSeveral experiments are presented in order to compare two strategies of parallel genetic algorithms on GPU. The problems were obtained from the Travelling Salesman Problem Library (TSPLIB) [10]. In all experiments a mutation rate of 5% for PGAIM and 100% for PGAEI was used, and each individual to mutate was selected by tournament of five individuals chosen at random. The algorithms were implemented in CUDA C on a 3.4 Ghz Intel-i7 desktop computer with 16 GB of RAM memory, running on a GNU/Linux 64-bits operating system. The algorithms are processed on a GPU GeForce GTX 780. Results for different TSP problems of TSPLIB library with PGAIM are presented in Table 1 and results with PGAEI are presented in Table 2. The Time column

Table 1. TSP solution for different problems of TSPLIB with PGAIM.

Problem	TSPLIB	Best	Worst	Mean	Error	Stan. dev. (σ)	Time (sec)
XQG237	1019	1038	1065	1049	1.86	11.24	9.01
BCL380	1621	1639	1676	1663	1.09	9.78	32.87
PBM436	1443	1485	1517	1503	2.91	9.62	66.02
XQL662	2513	2643	2694	2661	5.17	13.91	293.07
RBU737	3314	3442	3496	3467	3.86	24.03	355.69
DKG813	3199	3396	3481	3435	6.15	13.15	535.38
XIT1083	3558	3808	3902	3852	7.02	25.06	625.27
DKA1376	4666	5013	5115	5065	7.43	25.63	901.54
RBY1599	5533	6153	6276	6198	11.20	32.07	1052.36

Table 2. TSP solution for different problems of TSPLIB with PGAEI.

Problem	TSPLIB	Best	Worst	Mean	Error	Stan. dev. (σ)	Time (sec)
XQG237	1019	1030	1052	1043	1.08	7.83	9.83
BCL380	1621	1633	1669	1654	0.74	12.67	35.36
PBM436	1443	1474	1509	1497	2.14	12.41	70.19
XQL662	2513	2606	2666	2646	3.70	18.68	309.02
RBU737	3314	3416	3453	3437	3.07	20.47	367.54
DKG813	3199	3316	3351	3332	3.65	9.83	550.31
XIT1083	3558	3744	3810	3781	5.22	20.93	644.71
DKA1376	4666	4934	4996	4960	5.74	21.95	976.16
RBY1599	5533	5943	6141	6025	7.41	25.63	1161.18

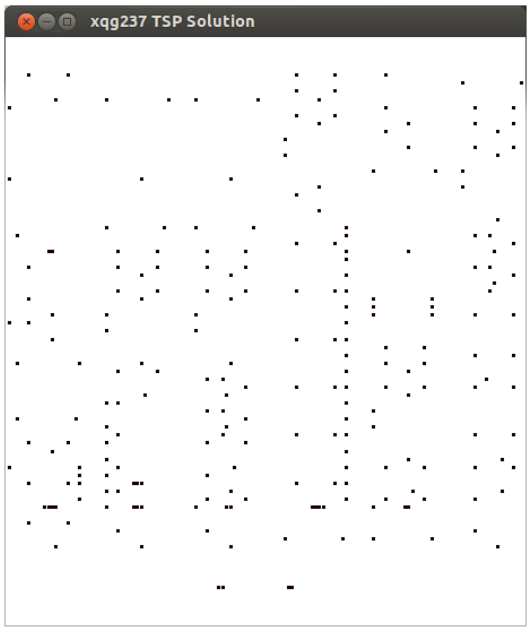

Figure 3. Representation of cities for the problem xqg237 of TSPLIB.

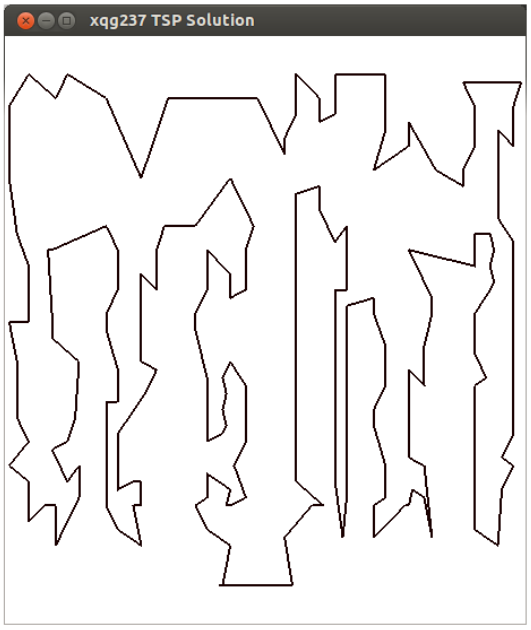

Figure 4. Best solution obtained with the PGAEI algorithm for the problem xqg237.

shows the time it takes to run the entire program, and the Error column shows the error of approximation of the value of the fitness function of the best individual that provides the genetic algorithm (best individual is the solution having the minimum distance over the cities) and the TSPLIB optimum. The error of the best path found with regard to the optimal route in the TSPLIB reported is calculated as follows:

$$Error = \left(\frac{BestSolution - OptimumTSPLIB}{OptimumTSPLIB} \right) x100, \quad (2)$$

where *BestSolution (Best column) is the result throw by implemented algorithm and* OptimumTSPLIB (TSPLIB column) is the TSPLIB optimum. Figure 3 presents the problem xqg237, the cities are represented by the holes of a printed circuit board. Also, the Figure 4 shows a solution to implement PGAEI to solve problem xqg237.

In Table 1, we can see that the more cities have the TSP problem the algorithm solution could be further from the desired optimum, this due to the how the complexity of the TSP in relation to the number of cities increases. For the mean results in both cases 10 runs of the algorithm were made.

The Table 2 shows that the behavior is very similar to the PGAIM but PGAEI algorithm yields better results than PGAIM in very similar time. In both cases we can see that the error increases as the number of cities increases, and the PGAEI got less than 8% error for all the problems.

6. Conclusions and Future Work

In this paper parallel genetic algorithms were implemented on a GPU to approximate the solution of TSP problem. Two versions of parallel genetic algorithms were implemented, a Parallel Genetic Algorithm with Island Model and a Parallel Genetic Algorithm with Elite Island. The Parallel Genetic Algorithm with Elite Island gives better results than the Parallel Genetic Algorithm with Island Mode. This advantage is provided by the Elite Island because it shared the fittest individuals among all the islands in almost the same time.

Currently, we are developing a version of parallel genetic algorithm on multi-GPU to solved TSP bigger problems, the main idea is to have a Parallel Genetic Algorithm with Elite Island by GPU and make migration between Elite Islands.

7. Acknowledgements

It is appreciated the support from Instituto Politécnico Nacional (Project grants SIP20140679 and SIP20140678).

References

[1] A. E. Eiben, and J. E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.
[2] S. N. Sivanandam, and S. N. Deepa. *Introduction to Genetic Algorithms*. Springer, 2008.
[3] G. Luque, and E. Alba. *Parallel Genetic Algorithm: Theory and Real World Applications*. Springer, 2011.
[4] S. Cahon and N. Melab and E.-G. Talbi. "ParadisEO: A Framework for the Reusable Desing of Parallel and Distributed Metaheuristics", *Journal of Heuristics*, Vol. 10, pp. 357-380, May 2004.
[5] J. Hofmann, and S. Limmer and D. Fey. "Performance investigations of genetic algorithms on graphics cards", *Swarm Evol. Comput.*, Vol. 12, pp. 33-47, October 2013.
[6] L. Gaxiola and J. Tapia and J. Rolón. "Optimización en la aproximación de curvas NURBS con Algoritmos Genéticos Paralelos en un GPU", *DIFU100ci@*, Vol. 7, No. 2, pp. 8-16, 2013.
[7] T.H. Cormen and Ch.E. Leiserson and R.L. Rivest and C. Stein. *Introduction to algorithms*. MIT Press, third edition, 2009.
[8] M. Dorigo and T. Stutzle. *Ant Colony Optimization*. MIT Press, 2004.
[9] E. Cant-Paz. *Efficient and accurante parallel genetic algorithms*. Kluwer Academic Publishers, 2001.
[10] Georgia Institute of Technology. *The Travelling Salesman Problem*. www.tsp.gatech.edu/data/index.htmlL March 2003.