A Parallel Variable Neighborhood Search approach for solving the Traveling Salesman Problem

RODRIGO E. MORALES-NAVARRO[1]
RAFAEL RIVERA-LOPEZ[2]
ABELARDO RODRIGUEZ-LEON[2]
Instituto Tecnológico de Veracruz
Departamento de Sistemas y Computación
M.A. Quevedo 2779, Col. Formando Hogar
Veracruz, VER
MÉXICO
[1]monroe.eskinka@gmail.com
[2]{rrivera,arleon}@itver.edu.mx

MARCO ANTONIO CRUZ-CHAVEZ
ALINA MARTINEZ-OROPEZA
Universidad Autónoma del Estado de Morelos
CIICAp
Av. Universidad 1001, Col. Chamilpa
Cuernavaca, MOR
MÉXICO
{mcruz,alinam}@uaem.mx

Abstract: This paper presents a parallel Variable Neighborhood Search (pVNS) algorithm for solving instances of the Traveling Salesman Problem (TSP). pVSN uses two parallelization levels in order to reach near-optimal solutions for TSP instances. This parallel approach is evaluated by means of an experimental multi-clusters of computers in which the nodes in each cluster uses several threads for calculating the path cost of a candidate solution and a message passing based communication scheme between two clusters is implemented for sharing their solutions. The results obtained indicate that the use of this parallelization scheme leads to an execution time reduction of the VNS method and one near optimal solution is reached due to a best exploitation of the solution space.

Key–Words: Variable neighborhood search, parallel algorithm, traveling salesman problem

1 Introduction

An objective of combinatorial optimization is to implement better algorithms for searching an optimal solution to a problem inside a finite collection of candidate solutions [1]. This area brings together many computationally complex problems in which usually the number of solutions is very high, making it impractical to evaluate all of them in order to find the optimal solution. The use of parallelization for solving combinatorial optimization problems is an approach that has gained importance in recent years [2]. Parallel combinatorial optimization algorithms are raising significant interest in science and technology since they are able to solve complex problems in different domains (communications, genomics, logistics and transportation, environment, engineering design, etc.) [3]. For example, in [4] it is described an efficient load balancing strategy for grid-based branch and bound algorithm and in [5] a parallel genetic annealing algorithm for solving the Job Shop Scheduling Problem is presented.

The traveling salesman problem (TSP) is perhaps the most well known combinatorial optimization problem. Applications of TSP and its variants go way beyond the route planning problem of a traveling salesman and spans over several areas of knowledge including mathematics, computer science, operations research, genetics, engineering and electronics [6]. TSP is among the NP-hard problems and many different approaches are used to solve this problem in an acceptable time especially when the number of cities is high [7]. Within the process of finding near-optimal TSP solutions, several techniques have been proposed that use different neighborhood structures and local search procedures ([8, 9, 10, 11, 12]). A hybrid approach that combines different neighborhood structures to find better solutions within an iterated local search algorithm is presented in [13].

High-performance computing (HPC) is an approach that has been used in recent years for solving complex problems in science, engineering and business. In [14] it is established that HPC can be achieved using a cluster of computers or a Grid computing or a workstation personal computer with a multi-core system. A cluster is one single system comprised of interconnected computers that communicate with one another either via message passing or by direct internode memory access using a single address space [15]. In México the Instituto Tecnológico de Veracruz (ITVer) and the Universidad Autónoma del Estado de

Morelos (UAEM) share the resources of their clusters in an experimental multi-clusters architecture named Tarantula minigrid [16].

This paper presents a parallel approach of a Variable Neighborhood Search (VNS) method described in [17]. This parallel approach, called pVNS, performs a concurrent finding of new candidate solutions for TSP instances in a multi-clusters of computers. pVNS uses two levels of parallelization: first, a set of threads in each node of a cluster is used for calculating the path cost of a TSP solution and then a message passing mechanism for the communication between clusters is implemented. The experimental results obtained show that a reduction in the execution time of pVNS method is produced and a substantial improvement in the solution is reached. The remainder of this paper is organized as follows: The definition of TSP, the neighborhood structures and the VNS method is presented in sections 2, 3 and 4, respectively. Section 5 describes the parallel approach for VNS proposed in this work and one analysis of the experimental results is provided in section 6. Finally, in section 7 the conclusions and future works are detailed.

2 Traveling salesman problem

TSP is a classical reference in combinatorial optimization and it has become a standard case for testing the effectiveness of several optimization methods [18]. In [19] it is established that TSP is probably the most important among the combinatorial optimization problems that it has served as a testbed for almost every new algorithmic idea, and was one of the first optimization problems conjectured to be *hard* in a specific technical sense.

In its mathematical formulation, cities are the vertices of a graph K, edges of this graph represent the opportunity of to travel between two cities, and a tour is a hamiltonian cycle in K. The task in TSP is to arrange a tour of several cities such that each city is visited only once and the length of the tour (or some other cost function) is minimized [20].

In [9] it is defined that given N cities, TSP requires a search for a permutation $\pi : \{1, \ldots, N\} \rightarrow \{1, \ldots, N\}$, using a $N \times N$ cost matrix $\mathbf{C}$, where $c_{i,j}$ denotes the cost of travel from city i to j, that minimizes the path length in equation 1.

$$\sum_{i=1}^{N-1} c_{\pi(i),\pi(i+1)} + c_{\pi(N),\pi(1)} \tag{1}$$

where $\pi(i)$ denotes the city at i-th location in the tour.

Different classes of TSP can be identified by the properties of the cost matrix. In symmetric TSP $c_{i,j} = c_{j,i}$ for $i, j : 1, \ldots, N$, otherwise the problem is referred as a asymmetric TSP. Euclidean TSP is a special case of symmetric TSP in which the vertices are points in the Euclidean space and the weight on each edge is the Euclidean distance between its endpoints.

3 Neighborhood structures

Any combinatorial optimization problem requires the use of techniques that allows a better exploitation of the solution space in order to reach near-optimal solutions. Local search techniques utilize a neighborhood structure for defining a set $\mathcal{N}$ of candidate solutions. Therefore, any solution s' is directly reachable from s through a movement ρ, thus $s' \in \mathcal{N}(s)$. This neighborhood structure is implemented in an iterative technique with the purpose of improving the quality of solutions, according to the objective function of the problem. A critical aspect of designing some optimization algorithms is choosing an appropriate neighborhood structure.

In [17] four different neighborhood structures were applied for searching a near-optimal solution of a TSP instance:

- **One adjacent pair** [21, 22, 23]: One element in position i of a solution s is randomly selected and it is permuted with the element in position $i + 1$ of the same solution.
- **One random pair** [22, 24, 25]: This structure performs a single permutation between two non-adjacent elements of s placed on random positions i and j.
- **Two adjacent pairs** [26]: Two elements of s in random positions i and j are selected and they are interchanged with their adjacent elements: Element in position i permutes with element in position $i + 1$, and element in position j permutes with element in position $j + 1$.
- **Two random pairs** [25, 27, 28]: This structure performs two permutations between four non-adjacent elements of s placed on random positions i_a, i_b, j_a and j_b. Element in position i_a permutes with element in position i_b and element in position j_a permutes with element in position j_b.

4 Variable Neighborhood Search

VNS is a framework for building heuristics based upon systematic changes of neighborhoods both in a descent phase, to find a local minimum, and in a perturbation phase to escape from the corresponding valley [29]. VNS is able to handle variable neighborhood sizes, due to the random interaction of the

structures during the execution time, which improves the exploitation of the solution space. In [17] the four different neighborhood structures previously described have been implemented in order to search near-optimal solutions of TSP instances using a VNS method. Figure 1 shows the structure of this VNS method.

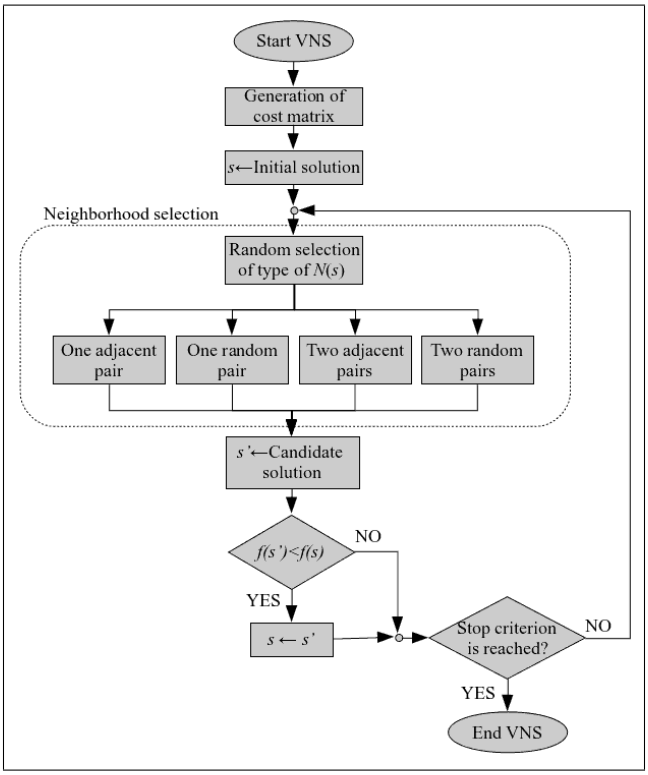

Figure 1: Flowchart of VNS method proposed in [17].

5 Parallel Variable Neighborhood Search

An analysis of the execution time profile of the VNS sequential version proposed in [17] indicates that the determination of the path cost of each candidate solution constructed using the neighborhood structures is the most expensive procedure since it consumes 99% of the total execution time, then one parallel VSN (pVNS) is implemented in order to reduce this execution time. For improving the exploitation of the search space, pVNS uses a group of solutions instead only one candidate solution in its iterative process.

pVNS performs a concurrent finding of new candidate solutions of a TSP instance in a multi-clusters of computers in order to reach a near-optimal solution. pVNS uses two levels of parallelization: first, a set of threads on each cluster node for calculating the path cost of a TSP solution is used and then a message passing mechanism for the communication process between clusters is implemented. For constructing this parallelization scheme one node in each cluster is asigned as the *master node* in which the VNS procedure is executed. Master node sends a candidate solution to each one of the other nodes in the cluster (the slave nodes) in order to calculate the path costs of this group of candidate solutions (figure 2). Implementation details of these two parallelization levels are described in the next paragraphs.

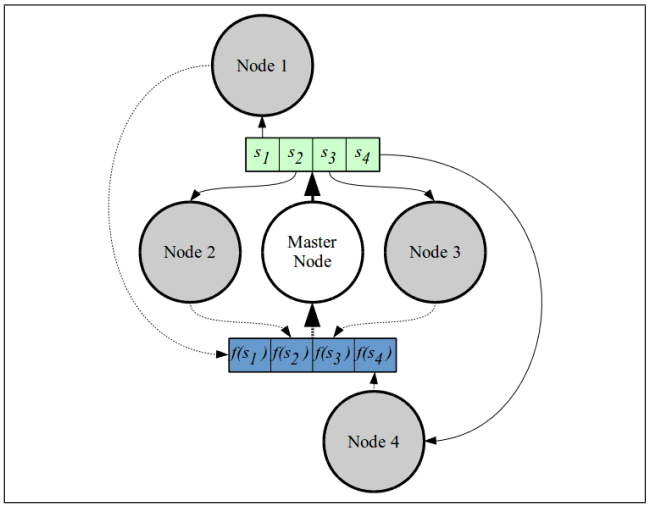

Figure 2: Master node sends candidate solutions and receives the path cost of each one.

Parallelization Level 1: It is a shared memory parallelization scheme in which a slave node i in the cluster receives a candidate solution s_i and evaluates it. This solution is divided into n segments and each one is proccesed by a independent thread T_j in order to determine its partial path cost $f(s_{i,T_j})$ of each segment. When all threads have finished their part of the parallel computation, slave node i determines the total path cost $f(s_i)$ and sends it to the master node. This parallelization level is depicted in figure 3.

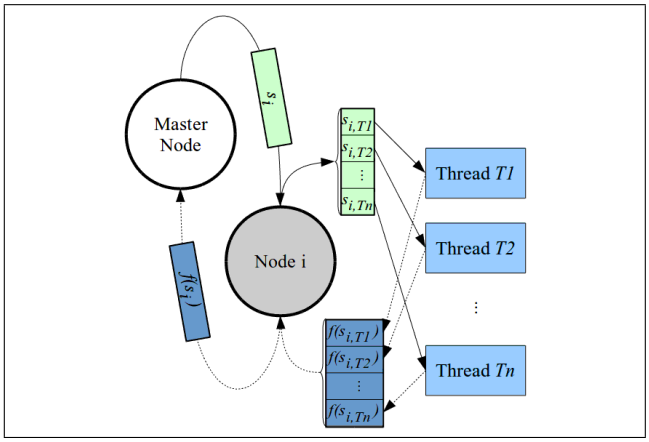

Figure 3: A set of threads calculates the path cost of solution s_i.

Parallelization Level 2: In order to exploit the advantages of a parallel processing scheme, pVNS utilizes a group of candidate solutions instead of only one as in the sequential version of VNS proposed in [17]. First, a master node delivers candidate solutions and receives their path costs calculated by the slave nodes in the cluster (figure 2) and then updates the group of solutions comparing the path costs received. This send-receive process with the slave nodes is repeated until a stop condition is reached. Additionally, before of delivering new candidate solutions, each master node of the multi-clusters architecture shares its better solutions with the others using a message passing mechanism as is shown in figure 4. With this interchange procedure only the best solutions of this distributed searching scheme is preserved as a the new set of candidate solutions.

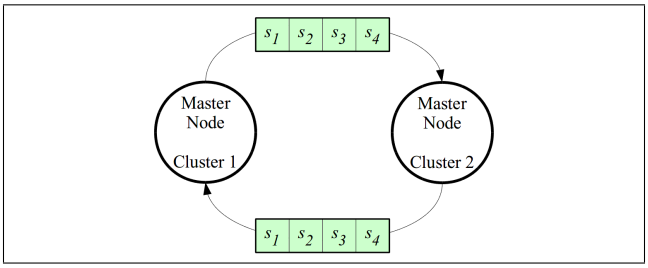

Figure 4: Solutions interchange between two clusters.

6 Experimental results

Experimental tests of pVNS were realized in the Tarantula minigrid that is a multi-clusters architecture configured as a Virtual Private Network (VPN) using OpenVPN in order to avoid the need for grid packages such that Globus or gLite 32. Basic components of Tarantula miniGrid are shown in table 1.

Table 1: Tarantula miniGrid components.

Software
Red Hat Enterprise Linux 4, Compiler gcc version 3.4.3, OpenMPI 1.2.8, MPICH2-1.0.8, Ganglia 3.0.6, NIS ypserv-2.13-5, NFS nfs-utils-1.0.6-46,OpenVPN, Torque + Maui.
Hardware
ITVer cluster: A master node with a dual-core system of 3.2 Ghz and 14 slave nodes (dual and quad core systems) of 2.33 GHz. **UAEM cluster**: A master node with a dual dual-core system of 2.8 GHz and 18 slave nodes with a dual-core system of 2.0 Ghz.

pVNS method is codified in C language and two libraries for implementing parallel tasks are utilized: OpenMP for the parallelization level 1 and MPI for the parallelization level 2.

One TSP instance randomly generated for 4000 cities was utilized for evaluating the pVNS performance and the average results of 30 executions of the method using a group of 60 candidate solutions were obtained. Due to the heterogeneity of the nodes used in the multi-clusters architecture, tests were performed using different node configurations:

Test 1: Sequential VNS (1 quad-core node).

Test 2: Sequential VNS (1 dual-core node).

Test 3: pVNS, 10 processes (2 quad-core nodes, 1 dual-core node).

Test 4: pVNS, 10 processes (3 quad-core nodes).

Test 5: pVNS, 20 processes (5 dual-core nodes).

Test 6: pVNS, 20 processes (5 quad-core nodes).

Test 7: pVNS, 20 processes (9 dual-core nodes, 1 quad-core node).

Test 8: pVNS, 30 processes (9 dual-core nodes, 3 quad core nodes).

6.1 Execution time and total path cost

Figures 5 and 6 show the average total execution time and the average total path cost obtained applying the eigth tests described previously. Figure 5 shows a substantial reduction in the execution time of pVNS in comparison with the better execution time of the sequential version of VNS. The generation of 60 candidate solutions using 30 processes (test number eight) yields a total cost much smaller than the sequential one: the execution time was reduced approximately 23 times compared to the execution time of sequential version using one quad-core node (test number two).

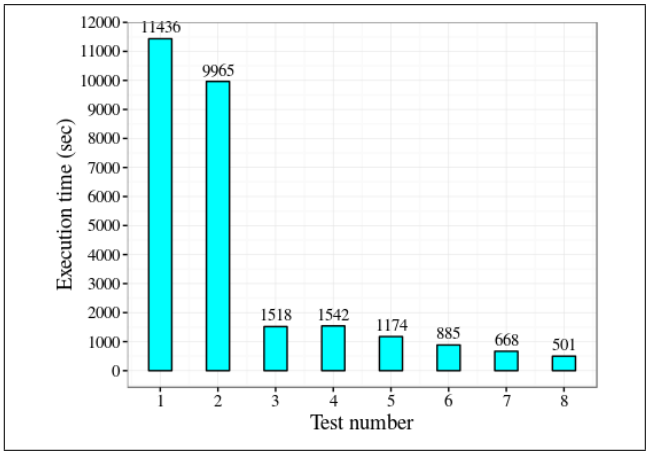

Figure 5: Average total execution time of pVNS.

A comparative graph of several average total path costs produced by the different node configurations is shown in figure 6. The better average total path cost produced by the tests with the sequential algorithm was 187,339 (test number two), while the better average total path cost was 186,025 (test number three) for the tests with the parallel version. Parallel versions of VNS always produce better candidate solutions due to the fact that they implement a better exploitation of the search space by using a group of candidate solutions instead of to use only one candidate solution in each iteration of a sequential version of VNS.

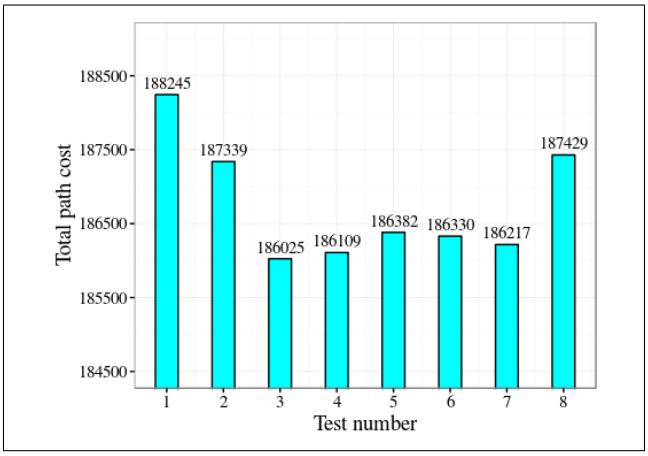

Figure 6: Average total path costs of pVNS.

6.2 Speedup

Speedup is the measure for the gain of the parallel program over the sequential version. In figure 7 is depicted the speedup obtained using different node configurations for the pVNS method. It can be seen that the speedup varies depending on the combination of the nodes of the Tarantula minigrid. The speedup increase curve turns out to be as expected, obtaining an efficiency of 66% when the node configuration use 30 processes (9 dual-core nodes and 3-quad core nodes).

7 Conclusions and future work

The implementation of this proposal of two parallelization levels using OpenMP and MPI generates a substantial reduction in the execution time for the VNS method, obtaining results in less time than the sequential version of VNS. Parallelization with MPI allows to expand the search in the space of solutions, since having several candidate solutions improves the explotaition of the solution space and it is more probable that a near-optimal solution can be reached.

In addition, with the use of a multi-clusters architecture, a greater computing power is obtained that

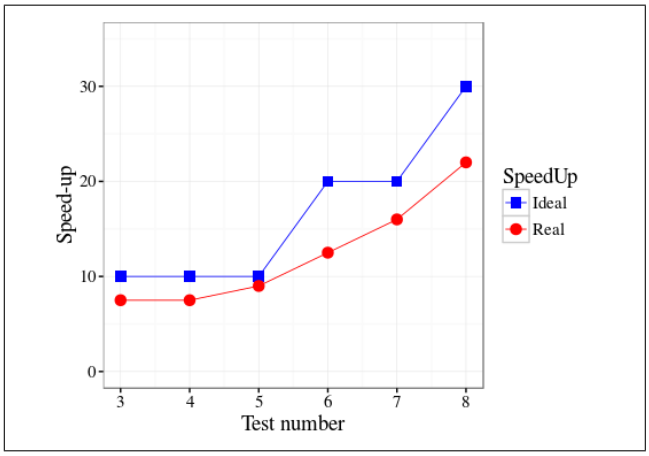

Figure 7: Speed-up of pVNS.

allows concurrently evaluating several candidate solutions. It can be seen that the use of parallelization reduces approximately 23 times the execution time of the algorithm.

The hybridization of OpenMP and MPI is an interesting approach that merits further study. The next steps in this work will be oriented in two directions: First the experimental study will be improved using several test cases obtained from the literature and then a study about the effect of latency when the nodes of different clusters are shared across the multi-clusters architecture using MPI will be developed. Another future work is to implement different heuristics as evolutionary algorithms and swarm intelligence methods in order to solve TSP instances in a parallel environment.

References:

[1] A. Schrijver, *Combinatorial optimization: polyhedra and efficiency*, Springer, 2002.

[2] K. Fujisawa, M. Kojima, A. Takeda and M. Yamashita, Solving Large Scale Optimization Problems via Grid and Cluster Computing, *Journal of the Operations Research Society of Japan*, 47(4), 2004, pp. 265–274.

[3] E.G. Talbi, *Parallel combinatorial optimization*, Wiley, 2006.

[4] M. Mezmaz, N. Melab and E.G. Talbi, An efficient load balancing strategy for grid-based branch and bound algorithm, *Parallel computing*, 33(4), 2007, pp. 302–313.

[5] M.A. Cruz-Chávez, A. Rodríguez-León, E.Y. Ávila-Melgar, F. Juárez-Pérez, M.H. Cruz-Rosales and R. Rivera-Lopez, Genetic-Annealing Algorithm in Grid Environment for Scheduling Problems, *Communications in Computer and Information Science: Security-Enriched Urban Computing and Smart Grid*, Springer, 2010, pp. 1–9.

[6] G. Gutin and A.P. Punnen, *The traveling salesman problem and its variations*, Springer, 2002.

[7] V. Zharfi and A. Mirzazadeh, A Novel Metaheuristic for Travelling Salesman Problem, *Journal of Industrial Engineering*, 2013, 5 pages, 2013.

[8] M. Hahsler and K. Hornik, TSP – Infrastructure for the Traveling Salesperson Problem, *Journal of Statistical Software*, (23)2, pp. 1–21, 2007.

[9] D.S. Johnson and L.A. McGeoch, The Traveling Salesman Problem: A Case Study in Local Optimization, *Local Search in Combinatorial Optimization*, Wiley, pp. 215–310, 1997.

[10] S.B. Liu, K.M. Ng and H.L. Ong, A New Heuristic Algorithm for the Classical Symmetric Traveling Salesman Problem, *World Academy of Science, Engineering and Technology*, 27(48), pp. 34–348 2007.

[11] M.K. Rafsanjani, S. Eskandari and A.B. Saeid, A similarity-based mechanism to control genetic algorithm and local search hybridization to solve traveling salesman problem, *Neural Computing and Applications*, 26(1), pp. 213–222, 2015.

[12] Y. Lin, Z. Bian and X. Liu, Developing a dynamic neighborhood structure for an adaptive hybrid simulated annealing–tabu search algorithm to solve the symmetrical traveling salesman problem. *Applied Soft Computing*, 49, pp. 937–952, 2016.

[13] M.A. Cruz-Chávez, A. Martínez-Oropeza and S.A. Serna-Barquera, Neighborhood Hybrid Structure for Discrete Optimization Problems, *Proc. of IEEE CERMA 2010*, pp. 108–113, 2010.

[14] E. Mahdi, A Survey of R Software for Parallel Computing, *American Journal of Applied Mathematics and Statistics*, 2(4), pp. 224–230, 2014.

[15] G. Bell and J. Gray, What's next in high-performance computing?, *Communications of the ACM*, 45(2), pp. 91–95, 2002.

[16] R. Rivera-Lopez, A. Rodríguez-León, M.A. Cruz-Chávez and I.Y. Hernández-Baez, Tarántula: Una grid de clusters de cómputo para el desarrollo de aplicaciones paralelas y en grid en México, *Memorias del Coloquio de Investigación Multidisciplinaria*, Orizaba, MEX, 2011.

[17] M.A. Cruz-Chávez, A. Martínez-Oropeza, J. del Carmen Peralta-Abarca, M.H. Cruz-Rosales and M. Martínez-Rangel, Variable Neighborhood Search for Non-deterministic Problems, *ICAISC 2014* LNAI 8868, Springer, pp. 468–478, 2014.

[18] D.L. Applegate, R.E. Bixby, V. Chvátal, and W.J. Cook, *The Traveling Salesman Problem: A Computational Study*, Princeton University Press,2006.

[19] D.S. Johnson and C.H. Papadimitriou, Computational Complexity, *The Traveling Salesman Problem*, Wiley, pp. 37–85, 1985.

[20] D.B. Fogel, An Evolutionary Approach to the Traveling Salesman Problem, *Biological Cybernetics*, 60, Springer, pp. 139–144, 1988.

[21] S. Lin and W. Kernighan, An Effective Heuristic for the Traveling Salesman Problem, *Operations Research* 21(2), pp. 498–516, 1973.

[22] R. González-Velázquez and M.A. Bandala-Garcés, Hybrid Algorithm: Scaling Hill and Simulated Annealing to Solve the Quadratic Allowance Problem, *Proc. of 3th. Latin-Iberoamerican Workshop of Operation Research*, 2009.

[23] J. Pacheco and C. Delgado, Different Experiences Results with Local Search Applied to Path Problem, *Electronic Journal of Electronics of Comunications and Works*, 2(1), pp. 54–81, 2000.

[24] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization, Algorithms and Complexity*, Dover, 1998.

[25] D.B. Kenneth, Cost Versus Distance in the Traveling Salesman Problem, *Tech Report*, UCLA Computer Science Dept, 1995.

[26] H.R. Lourenço and O.C. Martin, Iterated Local Search, *Handbook of Metaheustics*, 57, Springer, pp. 320–353, 2003.

[27] O. Martin, S.W. Otto and E.W. Felten, Large Step Markov Chains for the Traveling Salesman Problem, *Complex Systems*, 5(3), pp. 299–326, 1991.

[28] O. Martin, S.W. Otto and E.W. Felten, Large Step Markov Chains for the TSP Incorporating Local Search Heuristics, *Operations Reasearch Letters*, 11(4), pp. 219–224, 1992.

[29] P. Hansen, N. Mladenović, R. Todosijević and S. Hanafi, Variable neighborhood search: basics and variants, *EURO Journal on Computational Optimization*, pp. 1–32, 2016.