Chapter 9
Exploring Feasible and Infeasible Regions in the Vehicle Routing Problem with Time Windows Using a Multi-Objective Particle Swarm Optimization Approach

J.P. Castro, D. Landa-Silva, and J.A. Moreno

Abstract This paper investigates the ability of a discrete particle swarm optimization algorithm (DPSO) to evolve solutions from infeasibility to feasibility for the Vehicle Routing Problem with Time Windows (VRPTW). The proposed algorithm incorporates some principles from multi-objective optimization to allow particles to conduct a dynamic trade-off between objectives in order to reach feasibility. The main contribution of this paper is to demonstrate that without incorporating tailored heuristics or operators to tackle infeasibility, it is possible to evolve very poor infeasible route-plans to very good feasible ones using swarm intelligence.

Key words: Particle Swarm Optimization, PSO, Multi-Objective, Vehicle Routing Problem with Time Windows, VRPTW

9.1 Introduction

The Vehicle Routing Problem (VRP) is a well-known complex combinatorial optimization problem that consists of creating a set of routes, called a *route-plan*, to serve N costumers with a fleet of k vehicles. Each customer has a demand d_i ($i = 1 \dots N$) and each vehicle has a capacity C. A distance matrix D is given where d_{ij} is the Euclidean distance between customer i and customer j. One main objective

J.P. Castro
Automated Scheduling, Optimisation and Planning Research Group (ASAP), University of Nottingham (UK) e-mail: `jpc@cs.nott.ac.uk`

D. Landa-Silva
Automated Scheduling, Optimisation and Planning Research Group (ASAP), University of Nottingham (UK) e-mail: `jds@cs.nott.ac.uk`

J.A. Moreno
Group of Intelligent Computing, Dpto. de Estadística, I.O. y Computación, Escuela Técnica Superior de Ingeniería Informática, Universidad de La Laguna (Spain) e-mail: `jamoreno@ull.es`

in the VRP is to minimize the *number of vehicles* needed because this represents the largest fixed cost for companies. Another main objective is to minimize the *total distance* in the route-plan, i.e. the sum of distances travelled by all vehicles. Common constraints in the VRP include among others: (a) not to exceed the maximum capacity C, (b) every vehicle must set off and arrive at the depot, (c) every costumer can only be visited once, and (d) every route is served by exactly one vehicle.

Due to its wide application on real-world scenarios, there are several variants of the VRP [1]. Many companies have more than one basement for goods so the problem becomes the multi-depot VRP (MDVRP). In the periodic VRP (PVRP) costumers may not be served all in the same day. Instead, a list of possible days within a time horizon is given for each customer. Applications of the PVRP arise in grocery industries, soft-drinks distribution and waste collection among others. The stochastic VRP (SVRP) refers to those scenarios with uncertain information about the number of costumers to be served, the travel time required for delivery, the demand, etc. In the split delivery VRP (SDVRP) the constraint no allowing to serve costumers more than once is relaxed in order to minimize the overall cost.

In this paper, we are interested in the VRPTW, an extension of the common VRP in which a time window (e_i, l_i) is given for the depot $(i = 0)$ and for every costumer $(i = 1 \ldots N)$. If the delivery vehicle arrives at the location of customer i at time $(t < e_i)$, then it will have to wait $(e_i - t)$ time until the costumer is ready to be served. Arriving at the location of customer i at time $(t > l_i)$ is not permitted. Similarly, no vehicle can depart the depot at time $(t < e_0)$ or arrive at the depot at time $(t > l_0)$. Although in the VRPTW delivery within the time windows is a hard-constraint, there is another version of the same problem in which delivery within the time windows is considered a soft-constraint so that tardiness is penalized in the objective function. For surveys on the VRPTW see the papers by Bräysy and Gendreau [2, 3].

The remainder if this paper is organized as follows. Section 9.2 describes the particle swarm optimization (PSO) paradigm which is used in this paper to tackle the VRPTW. Section 9.3 describes the multi-objective discrete PSO proposed in this paper to evolve infeasible solutions to feasible ones using a dynamic trade-off of the multiple objectives. Section 9.4 describes and discusses experiments and results while Section 9.5 concludes this paper.

9.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a population-based search technique inspired by the social behavior of bird flocking or fish schooling. PSO is a kind of swarm intelligence algorithm developed by Kennedy and Eberhart [4]. Particles in a PSO implementation fly through a multi-dimensional continuous space. Each particle (identified by an id) has a position x^{id} given by the current solution that the particle is exploring, and a velocity v^{id} used to update the particle's position. In a basic PSO implementation, each particle knows at least its best position and the best position of its neighborhood. With this information, the new particle's position is computed as follows:

$$v^{id} = w \cdot v^{id} + c_1 \cdot rand() \cdot (g_i - x^{id}) + c_2 \cdot rand() \cdot (g - x^{id}) \tag{9.1}$$

$$x^{id} = x^{id} + v^{id} \tag{9.2}$$

Eqs. (9.1) and (9.2) determine how a particle updates its velocity and position respectively. The velocity update is highly affected by the inertial and social coefficients. The inertial weight component w encourages exploration by the particle while the social coefficients c_1 and c_2 represent attraction forces to the best location the particle has achieved and to the best position found by the swarm so far. In order to avoid a predictable behavior, the social coefficients are multiplied by random numbers *rand*() chosen from a uniform distribution U[0,1]. Once the velocity is updated using (9.1), the location of the particle changes by summing up the current position to the new velocity as in (9.2). A basic PSO algorithm consists of a number of iterations in which the swarm evolves by updating the position (current solution) and velocity of each particle in the swarm. Because particles also know the best position of its neighbourhood (best solution achieved by other particles), the whole swarm evolves towards better positions guided by the leading particle.

PSO has been applied too many different problems due to its relatively easy implementation and high performance with low demand for computational resources. See the survey by Eberhart and Shi for a sample of applications of the PSO algorithm [5].

9.2.1 Multi-Objective Particle Swarm Optimization

Many real-world optimization problems have more than one objective and in most cases these objectives are in a trade-off conflict. In multi-objective optimization (MOO) there is not a single solution that optimizes all objectives at the same time, so it is useful to provide a set of *good trade-off* solutions for the decision-maker to choose one that fits best the current requirements. A common approach to compare solutions in MOO is to use Pareto dominance which works as follows. Given two solutions x^1 and x^2 for a minimization problem with q objectives, we say that x^1 *dominates* x^2 ($x^1 \prec x^2$) if $x^1_i \leq x^2_i$ for every $i = 1 \ldots q$ and $x^1_i < x^2_i$ for at least one i. Moreover, if $x^1_i < x^2_i$ for every i we say that x^1 *strictly dominates* x^2, otherwise we say that x^1 *weakly dominates* x^2 (usually, weak dominance is simply referred to as dominance). A solution x is set to be non-dominated with respect to a set S of solutions if there is not solution in S that (weakly or strictly) dominates x. A set of non-dominated solutions is usually called non-dominated set or Pareto front.

Coello Coello et al. [6] state that three main considerations must be taken into account when dealing with MOO using PSO:

1. The selection of the particles that will act as leaders.
2. The storage of non-dominated solutions found by the swarm.
3. The preservation of diversity within the swarm.

In recent years, a number of alternatives has been suggested to handle the optimization of multiple objectives using PSO. We briefly review some of these approaches. Moore and Chapman [7] maintain a list of non-dominated solutions (NDS) for each particle from which each particle's best position is chosen at random, the best neighbour is then selected using Pareto dominance. Coello Coello and Lechuga [8] used an external archive to store NDS and a crowding-based hyper-grid to select leader particles and to encourage diversity within the swarm. Later, Coello Coello et al. [9] used an archive which was allowed to grow as necessary in order to avoid missing good NDS. They selected leader particles based on the concept of global attraction using the history of NDS solutions in the archive. Hu and Eberhart [10] introduced the concept of *dynamic neighborhood* in which the neighborhood's best position is chosen based on a compound criterion that takes into account the optimization objective and the proximity between particles. This work was extended by Hu et al. [11] by incorporating an external NDS archive as in [8]. In order to overcome the drawbacks of having an external repository with fixed size, Fieldsend and Singh [12] proposed a *dominated tree* to store NDS and to select leader particles. Parsopoluos and Vrahatis [13] proposed three aggregation criteria for the optimization of continuous functions: static Conventional Weighted Approach (CWA), dynamic Bang-bang Weighted Aggregation (BWA) and Dynamic Weighted Aggregation (DWA). Their tests showed that DWA is the best approach for continuous optimization functions. Parsopolous et al. [14] suggested a swarm to optimize each objective, each swarm communicates its global best to the other swarms to guide the search towards the Pareto front. Their approach does not store NDS, instead each swarm reports the best solution found with respect to its objective. Santana-Quintero et al. [15] extended the work in [8]. They reserved q particles in the swarm, each one to optimize one of the q objectives and maintain an *ideal vector* with the best value for each objective from the q particles. The leader particle is the one closest to the ideal vector. They used a local search heuristic based on rough sets theory to spread the NDS and adaptive ϵ-dominance to adjust the hyper-boxes according to certain geometric characteristics of the Pareto front.

9.2.2 Discrete Particle Swarm Optimization

The PSO algorithm was originally proposed to tackle continuous optimization problems. That is because particles in the swarm fly on a continuous space updating their position and velocity in every iteration. However, for combinatorial optimization problems, the notion of *smooth flying* disappears. Then, the associated concepts of velocity and position updates in the traditional PSO algorithm (Eqs. 9.1 and 9.2 above) lose their sense. Combinatorial optimization scenarios like the VRPTW require to redefine how a particle moves in a discrete space and hence to redefine the mechanism for updating the velocity and position of particles.

A discrete PSO was proposed by Kennedy et al. [16]. They interpreted changes of velocity in terms of probabilities by encoding the particle's position as a binary vector and using a stochastic velocity scheme. Since then, several authors have used this approach (e.g. [17, 18]). Other implementations of discrete particle swarm optimization (DPSO) have been introduced (e.g. [19, 20, 21]), see the survey in the paper by Martinez Garcia and Moreno Perez [22].

9.2.3 Jumping Frog Optimization

The Jumping Frog Optimization (JFO) approach proposed in [22, 23, 24] is based on the particles point of view instead of the solutions or particle's position. The metaphor of JFO is that of a group of frogs looking around for food while jumping from lilypad to lilypad. This group of frogs competes for food by jumping to the best locations so that if a frog is well-placed, then other frogs tend to move towards it. The JFO approach uses an interesting scheme without the need of velocity to update the particle's position. Instead, the position is updated using a *follower-attractor* system. When a particle wants to *jump* to a new better position, the particle uses a better positioned particle as a reference. For this particular case, a particle that wants to move (follower) towards the best positioned particle in the swarm (attractor) tries to be similar to its attractor, i.e. the follower will analyze the components of the attractor's position to somehow copy them, and eventually jump to a new better position.

9.3 Proposed MOJFO Algorithm

This section describes the proposed multi-objective JFO algorithm which is showed in Program 1. The particles in the swarm can perform four types of moves (jumps) depending on which particle acts as the attractor in each iteration. For this purpose, a segment [0,1] is divided into four sub-segments with variable width subject to a certain probability c_i so that $\sum_{i=1}^{4} c_i = 1$. The longer a sub-segment is, the more probability a particle has to strike out the particular associated move. Then, a random number r with uniform distribution $U[0,1]$ is chosen and the particle will move to a new position depending on the value of r as follows:

- If $r \in c_1$ then no particle acts as attractor and the particle makes a random move with respect to its current position x^{id}. The purpose of this *Inertial Move* is to explore the area around the particle's position. We decided to let the particle jump around the neighbourhood by using a λ-interchange (with $\lambda = 1$) to simply swap two nodes (customers) in different sub-routes.

- If $r \in c_2$ then the particle moves towards the best positioned particle in its current neighborhood. This type of move is done by applying a route-exchange algorithm that consists of copying a random sub-route from the attractor to the follower, removing those nodes in the follower that are in the new sub-route. This represents a *Cognitive Move* to influence an attraction force within the swarm.
- If $r \in c_3$ the move is based on the best position b^{id} found so far by the moving particle. The current position x^{id} takes a sub-route from b^{id} using the route-exchange algorithm. This *Local Move* helps the particle to explore similar structures to the ones in b^{id} in order to improve its current position x^{id}.
- If $r \in c_4$ then the attractor is the best position g found by the swarm during the search process so far. A route-exchange algorithm is carried out between x^{id} and g. This *Global Move* encourages reusing parts of the overall best solution g found so far on the design of different structures.

As in the original (single-objective) implementation of the JFO approach, after finding a new solution, local search is applied to improve its quality (improvements are accepted based on Pareto dominance). Finally, a particle finishes its jumping round by updating its best position b^{id} and the best global position g achieved so far. The update is based on Pareto dominance, i.e. when b^{id} and/or g are dominated by the new position n^{id}.

9.3.1 Solution Representation and Initilization

Since in a DPSO scheme particles do not fly on a continuous space, the position or solution is given by a codification of a VRPTW solution. A *route-plan* is given by a sequence of costumers who are visited by a number of vehicles that set off from the depot. A solution or *route-plan* for a problem with $i = 1$ to N customers and one $i = 0$ depot looks like:

$$x^{id} = (0, c^i, \ldots, c^j, 0, \ldots, 0, c^m, \ldots, c^n, 0)$$

For example, the *route-plan*: (0 2 1 4 0 5 3 6 0 9 8 0) has three routes. The first route is (0 2 1 4 0), the second route is (0 5 3 6 0) and the third route is (0 9 8 0). In the first route, the vehicle sets off from the depot 0, then visits costumer 2, followed by customers 1 and 4 before returning to the depot 0. The other two routes in the encoding have similar interpretations.

Our primary goal is to investigate the ability of the proposed algorithm to find feasible route-plans even if starting the search in the infeasible region. Hence, no heuristics were used to initialize the positions of the particles with feasible or near-feasible route-plans. Instead, the position of each particle is randomly generated which provokes starting solutions to have a very high number of violations of both vehicle capacity and time windows. Then, the swarm in our algorithm has to truly evolve solutions by making transitions between the feasible and infeasible regions

Program 1 JFO Algorithm Pseudo-code

```
do {
   forEach(p in swarm) {
      n = rand();
      case (n) {
                 // Inertial Move
         n is in c1: ni = lambda-i(p.xi)
                 // Cognitive Move
         n is in c2: gi = getBestPosInNeighborhood()
                  ni = subRouteCopy(gi, p.xi)
                 // Local Move
         n is in c3: ni = subRouteCopy(p.bi, p.xi)
                 // Social Move
         n is in c4: ni = subRouteCopy(g, p.xi)
      }
      localSearch(ni)
      computeFitness(ni)
      p.xi = ni
      if (firstIsParetoCompatibleOrBetter(ni, p.bi) {
         update(p.bi)
         if (firstIsParetoCompatibleOrBetter(ni, g)
            update(g)
      }
   }
} while (!stopCriterion)
```

Legend:
$p.x^{id}$ is the current position of the particle.
$p.b^{id}$ is the best position found by the own particle.
g_{iter} is the best position in the swarm/neighborhood in the current iteration.
g is the best position found by the swarm.
n^{id} is the new position of the particle.
lambda-i is an operator that exchanges two costumers in different routes.
getBestPosInNeighborhood() is a procedure that picks the best solution communicated by its neighborhood.
subRouteCopy(source, dest) is a function that copies an entire sub-route from ‘source’ to ‘dest’. All the costumers who are placed in the new sub-route are previously deleted in ‘dest’.

of the search space as necessary. Examples of starting and evolved tour-plans are shown later in this paper.

9.3.2 Constraints and Objectives

We conducted preliminary experiments to understand how the setup of various objectives affects the convergence towards feasible solutions. These experiments were conducted using eight objectives, four of which are constraints and the other four are genuine objectives representative from the most commonly used in the VRP literature. The constraints are: *Time Window Violation (R_1), Number of Time Window*

Violations (R_2), Capacity Violation (R_3) and Number of Capacity Violations (R_4). The proper objectives are: *Number of Vehicles (O_1), Total Distance (O_2), Waiting Time (O_3) and Elapsed Time (O_4).* The waiting time is the sum of waiting times for all vehicles and the elapsed time is the total time elapsed between the departure of vehicles and the arrival of the last vehicle to the depot. Following our preliminary experiments, we decided to guide the swarm towards finding feasible solutions using four objectives: O_1, O_2, R_2 and R_3 while values of the other four objectives were just traced for our posterior analysis.

9.4 Experimental Analysis

This section describes our computational experiments and results. We used the popular Solomon's problem instances [1] which are divided into three categories: with cluster-located costumers (cxxx), with random-located costumers (rxxx) and a mix of these two (rcxxx). We took two instances from each category for our experiments, namely $c101, c201, r101, r201, rc101, rc201$. For the six instances, we fixed the number of vehicles to its best known solution, the reason for this is that we wanted to find out whether the swarm was able to move towards feasible areas after starting with violations of all constraints, so no operators were included to minimize the number of routes.

As mentioned above, each particle in the swarm starts with a purely random solution. This initial solution consists of a shuffled vector with the ids of the costumers and the number of zeros specified by the best known solution for the current instance. For example, for the Solomon's $c101$ the best known solution uses 10 vehicles to serve all costumers, so the initialization process involves to take the vectors of ids, to add 11 zeros (number of vehicles plus one) and to shuffle this vector. The c_i segments that the particles use to choose the next type of move, are all of constant length equal to 0.25 so that a particle has the same probability for choosing any of the above four types of move.

Regarding the issue of deciding when to update the best solution of the swarm g, our experiments showed that much better results are obtained when the update (as it is stated in 1) is done using weak-dominating solutions and not only strict-dominating solutions. Another observation made was that when we update g based on constraints violations, i.e. accepting a new swarm leader only when it improves upon the satisfaction of constraints, the convergence speed towards feasible solutions increases significantly.

Table 9.1 shows some illustrative results from our experiments. For each problem instance, the table presents the best solution g within the swarm in the *First Iteration* and in the *Last Iteration*. At the end of each row, there is the best known result for each instance as it can be found in [25]. Note that in all six instances the swarm is capable of moving from very bad infeasible solutions (high values of R_2 and R_3) to feasible (rows c101, r201 and rc201) or very close to feasible (rows c201, r101 and rc101) route-plans. In most cases, the value of the *Total Distance* objective

Table 9.1: Results obtained with the proposed MOJFO algorithm

	First Iteration				Last Iteration				Best Known	
Problem	O_1	O_2	R_2	R_3	O_1	O_2	R_2	R_3	O_1	O_2
c101	10	4327.17	95	810	10	828.94	0	0	10	828.94
c201	3	4233.16	95	290	3	1073	2	0	3	591.53
r101	19	3663.51	87	142	19	1752	1	0	19	1645.79
r201	4	3315.15	94	0	4	1460.86	0	0	4	1252.37
rc101	14	4829.68	90	535	14	1641.68	2	0	14	1696.94
rc201	4	4599.99	95	0	4	1825.56	0	0	4	1406.91

Legend:
O_1: Number of Vehicles
O_2: Total Distance
R_2: Number of Time Windows Violations
R_3: Capacity Violation

O_2 is very close to or matches (row c101) that of the best known solution. For the three cases in which the final route-plan is not feasible, the number of time windows violations is very low indeed. We are quite satisfied with these results which demonstrate that it is possible for our algorithm to effectively evolve very poor infeasible solutions towards good feasible ones without incorporating specific initilization heuristics or search operators to tackle infeasibility. Figure 9.1 shows an example of a starting (very poor) infeasible route-plan and the final (good-quality) feasible route-plan evolved by the swarm in our algorithm.

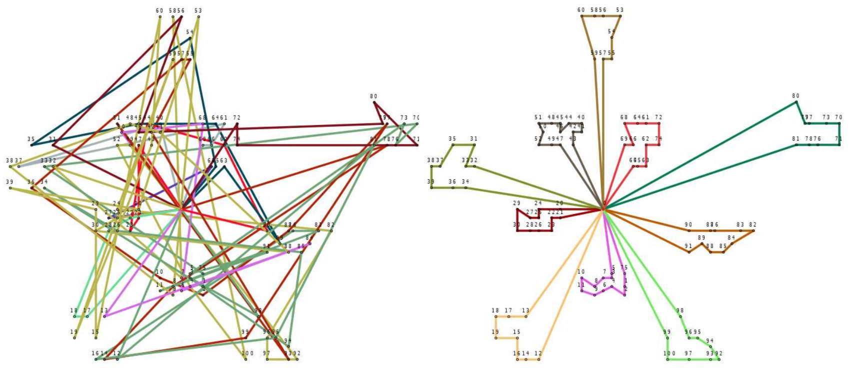
(a) Initial route-plan (b) Final route-plan

Fig. 9.1: Example of starting infeasible route-plan and evolved feasible route-plan

Since no stop condition is used to wipe out the swarm and since the global best g accepts weakly-dominating solutions, the swarm does not stop searching for better solutions. This provokes a large set of solutions to be found in the current non-dominated front once the optimal solution has been found or once g is very close to a feasible area and its objective is better than the optimum solution (see the row of $rc101$ in Table 9.1). After conducting repeated experiments with several instances and without forcing the global g to avoid the comparison of the *Total Distance* O_2, we observed that the swarm tends to an equilibrium state. That is, some time after the swarm starts moving, all the objectives begin to oscillate between a minimum and a maximum values, just like if the swarm was moving in circles. The same phenomenon occurs when we force the global g to ignore O_2 and not until the constraints get to value of zero, allow g to start comparing the *Total Distance*, then the constraints suddenly increase to that state of balance. Figure 9.2 shows the evolution of the swarm for a typical run of our algorithm on the c101 problem instance. The number of iterations (generations of the swarm) is in the X-axis while the Y-axis shows a normalized scale for all objectives. The range [max,min] explored during the search for each objective is also shown. We can see that the algorithm starts with high values of time windows violations (in fact, only 5% of costumers are served within the time window) and finishes with no time windows violations (same happens with the capacity violations). We can see that the algorithm manages the trade-off between the objectives allowing some objectives to worsen in order to improve others (because we use weak Pareto dominance). The graph shows abrupt rises and falls of the objective values. Note that around the 1150th iteration all the objectives suddenly go up except the Capacity Violation which achieves a value near to zero. This gives us an indication that an adaptive-MOJFO in which the particles decide which objectives to improve and how at different times during the search is a promising approach.

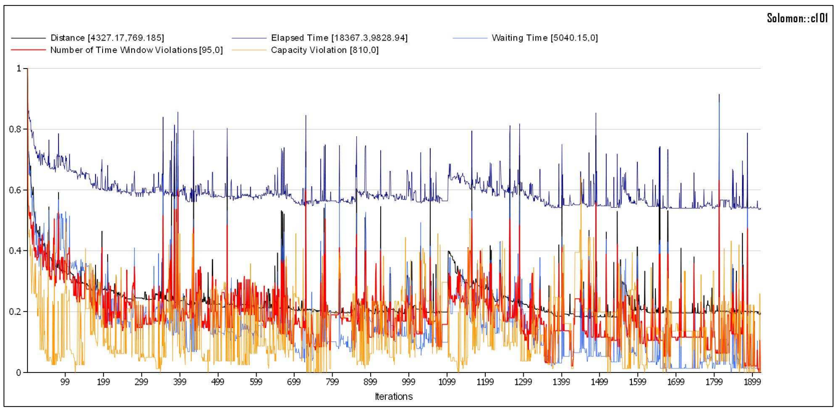

Fig. 9.2: Swarm evolution for typical run of the MOJFO algorithm on instance c101

9.5 Conclusions

We proposed an adaptation of the Jumping Frog Optimization algorithm incorporating some principles of multi-objective optimization to tackle the Vehicle Routing Problem with Time Windows. The proposed MOJFO algorithm uses Pareto dominance to guide the search by particles in the swarm allowing a dynamic trade-off between objectives during the search. Particles explore the discrete search space using four types of moves. Two objectives (number of vehicles and total distance) and two constraints (number of time window violations and capacity violation) are used as objectives to guide the search. It has been shown in this paper that the proposed approach is capable of starting from very poor infeasible solutions and evolve them to obtain very good feasible solutions without incorporating tailored heurisitics or operators to tackle infeasibility. In our future work we intend to incorporate some adaptive capability in the algorithm to allow particles to choose which move to perform and which objectives to tackle (and when) during the search for a more effective search process.

9.6 Acknowledgements

This work has been partially funded by the projects TIN-2005-08404-C04-03 and TIN2008-06872-C04-01/TIN of the Spanish Government.

References

[1] The VRP Web http://neo.lcc.uma.es/radi-aeb/WebVRP/

[2] Olli Bräysy, Michel Gendreau Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms *Transportation Science* Vol. 39, No. 1, pp. 104-118, 2005

[3] Olli Bräysy, Michel Gendreau Vehicle Routing Problem with Time Windows, Part II: Metaheuristics *Transportation Science* Vol. 39, No. 1, pp. 119-139, 2005

[4] J. Kennedy, R.C. Eberhart. Swarm Intelligence - Morgan Kaufmann. 2001

[5] Eberhart, R. C., Shi, Y. Particle swarm optimization: developments, applications and resources *Proceedings of the 2001 IEEE Congress on Evolutionary Computation* pp. 81-86, 2001

[6] M. Reyes-Sierra, C.A. Coello Coello *International Journal of Computational Intelligence Research* Research India Publications pp. 287-308, 2006

[7] J Moore and R Chapman Application of Particle Swarm to Multiobjective Optimization Department of Computer Science and Software Engineering, Auburn University, 1999

[8] C.A.Coello-Coello, M. Salazar Lechuga MOPSO: A proposal for multiple objective particle swarm optimization *Proceedings of the IEEE Congress on Computational Intelligence* pp. 12-17, 2002

[9] C.A. Coello Coello, G.T. Pulido, M. Salazar Lechuga Handling Multiple Objectives with Particle Swarm Optimization *IEEE Transactions on Evolutionary Computation* Vol. 8, No. 3, pp. 256-279, 2004

[10] Xiaohui Hu, Russell Eberhart Multiobjective Optimization Using Dynamic Neighborhood Particle Swarm Optimization *Proceedings of the 2002 Congress on Evolutionary Computation* pp. 1677-1681, 2002

[11] X. Hu, R.C. Eberhart, Y. Shi Particle Swarm Optimization with extended memory for multiobjective optimization *Proceedings of the IEEE Swarm Intelligence Symposium 2003* pp. 193-197, 2003

[12] Fieldsend, J.E. and Singh, S. A multi-objective algorithm based upon particle swarm optimisation, an efficient data structure and turbulence UK Workshop on Computational Intelligence (UKCI'02) pp. 37-44, 2002

[13] Parsopoulos, K.E., Vrahatis, M.N. Recent Approaches to Global Optimization Problems Through Particle Swarm Optimization *Natural Computing* pp. 235-306, Springer, 2002

[14] Parsopoulos, K.E., Vrahatis, M.N. On the Computation of All Global Minimizers Through Particle Swarm Optimization *IEEE Transactions on Evolutionary Computation, 8* pp. 211-224, 2004

[15] Luis V. Santana-Quintero, N. Ramírez-Santiago, C. A. Coello Coello, J. Molina-Luque and A. G. Hernández-Díaz A new proposal for multiobjective optimization using particle swarm optimization and rough sets theory *Lecture notes in computer science* Springer, pp. 483-492, 2006

[16] Kennedy, J and Eberhart, R.C. A discrete binary version of the particle swarm optimization algorithm *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics* pp. 4104-4109, 1997

[17] Chang, R. F. and Lu, C. N Feeder reconfiguration for load facto improvement *Proceedings of the IEEE Power Engineering Society Transmission and Distribution Conference* pp. 980-984, 2002

[18] Mohan, C.K. and Al-kazemi, B. Discrete Particle Swarm Optimization *Proceeding of the Workshop on Particle Swarm Optimization* 2001

[19] S. Yang, M. Wang, L. Jiao A quantum particle swarm optimization *Proceedings of the 2004 IEEE Congress on Evolutionary Computation* Vol. 1, pp. 320-324, 2004

[20] B. Al-kazemi, C.K. Mohan Multi-phase discrete particle swarm optimization *Proceedings of the Fourth International Workshop on Frontiers in Evolutionary Algorithms* 2000

[21] Combining particle swarm optimisation with angle modulation to solve binary problems *IEEE Congress on Evolutionary Computing* Vol. 1, pp. 89-96, 2005

[22] F. Javier Martinez Garcia, Jose A. Moreno Perez. Jumping Frogs Optimization: a new swarm method for discrete optimization. *Documentos de Trabajo del DEIOC*. N. 3/2008, Universidad de La Laguna

[23] Discrete Particle Swarm Optimization for the p-median problem F.J. Martinez, Jose A. Moreno, *MIC 2007, Metaheuristics International Conference* Montreal, Canada, 2007

[24] Discrete Particle Swarm Optimization for the minimum labelling Steiner tree problem Consoli S., Moreno-Perez J.A., Darby-Dowman K., Mladenovic N. :*Nature Inspired Cooperative Strategies for Optimization (NICSO 2007)* Vol. 129, Studies in Computational Intelligence, pp. 313-322, 2007

[25] Best Known Solutions Identified by Heuristics for Solomon's (1987) Benchmark Problems http://www.sintef.no/static/am/opti/projects/top/vrp/bknown.html