A new meta-heuristic framework: $A - B - Domain$

David Gómez-Cabrero[1], Carmen Armero[1], D. Nalin Ranashinge[2]

[1] Departamento de Estadística e Investigación Operativa, david.gomez@uv.es, carmen.armero@uv.es,
[2] University of Colombo School of Computing 35, Reid Avenue, Colombo 7, Sri Lanka dnr@ucsc.cmb.ac.lk

Abstract. The performance of any meta–heuristic algorithm is closely related to the fine-tuning of its parameters. For many algorithms there are no optimum set of parameters for all the instances of a given problem. Instead, fine-tuning phase works on the average performance over a selected set of instances. We propose a new algorithm framework, named *A–B–Domain*, that works as follows: a dominion where each point refers to a feasible set of parameters is constructed, for a given set of parameters fitness is computed by running a first algorithm, denoted as B, on the selected instance; a second algorithm, denoted as A, is used to check the most promising set of parameters. This $A - B - Domain$ scheme performs well on all the instances where B performs well on at least one set of parameters. In this paper we develop the $PSO - ACS - Domain$ example and improve its efficiency by implementing results from a statistical research.

1 Introduction

Heuristic algorithms were the first non-optimal approach to compute reasonable good solutions for NP-hard problems. $Meta\text{-}Heuristic$ (MH) algorithms are the most developed form of heuristics. To develop a MH algorithm there are three phases: design, programming and fine-tuning (FT). As a short review of the first two phases see [4] and [13].

The last phase is considered the most time consuming of all, and it is mainly interested on finding the optimum values for the set of parameters the MH algorithm uses. There are two main approximations for FT: statistical experimental design procedures (see [5]) and heuristic FT procedures (see CALIBRA [1], F-Race [3] and $PSO\text{-}ACO$ [15] as examples). For many MH algorithms there are no optimal set of parameters; instead for different set of instances there can be different optimum sets of parameters. FT procedures are based on the average performance over a selected set of instances and consequently if the set is modified FH phase must be repeated.

We propose a new framework able to work properly on a wider set of instances by finding the appropriate set of parameters for a given instance and a given heuristic. Let denote this scheme as *A–B–Domain*, where A and B are heuristic

algorithms and *Domain* is the set of all feasible sets of parameters on algorithm B. Given an instance *inst*, for each point $p \in Domain$ a fitness can be computed by running algorithm B using the set of parameters defined by p. Algorithm A is in charge of reviewing the most promising set of parameters related to *inst*.
As an initial approximation to this framework we develop an *ACS–PSO–Domain* structure working on the Travelling Salesman Problem (TSP). We have selected this combination because of the initial work proposed at [15] and because PSO is able to perform a *blindsearch* on any given *Dominion*. To improve the algorithm and reduce its computational cost a statistical research is performed and some new policies are implemented. Finally results obtained by *PSO-ACS-Domain* are compared with the results obtained by the *classical* ACS algorithm using several set of parameters from the literature.
Section 2 describes (shortly) ACS and PSO algorithms. Section 3 examines the *A-B-Domain* framework, and develops the *PSO-ACS-Domain* example. In Section 4 a statistical research on $ACS - TSP$ is performed and some recommended policies are defined. Computational research results are given in Section 5 and, finally, in Section 6 conclusions are set.

2 *PSO* and *ACS* Algorithms

ACS and PSO algorithms are examined in the necessary grade to develop an *A-B-Domain* example in following sections of the paper.

2.1 *AntColonySystem*(*ACS*)

In Ant Colony Optimization (ACO) algorithms ants are considered agents which construct tours by moving from node to node on the graph problem; in the TSP ants are constructing feasible tours. At the beginning of each iteration ants are initially set randomly to a node. Sequentially, for each ant a movement is computed: for each non-inserted node e, a p_e value is computed; this value considers the heuristic knowledge (powered to β) and the pheromone knowledge (powered to φ) gained until the actual iteration related to the arc that joins the actual node and the eligible one. Initially considering the neighbor set of the actual node, a random number in [0,1] is computed; if this value is less than a fixed $q_0 \in [0,1]$ the node with best p_e value is selected otherwise a random roulette method is applied. If it is not possible to select a node from the neighbor the selection is extended to all the non-visited nodes considering the same q_0. An iteration finishes when all ants have constructed a tour. For a given arc $j = (e_1, e_2)$, pheromone trail (τ) is updated at each iteration (global update) as follows:

$$\tau_j = (1 - \alpha)\tau_j + \alpha \Delta\tau_j, \tag{1}$$

where $\Delta\tau_j$ defines how much the arc appears in the best found solutions. We are using an *Ant Colony System* (ACS) algorithm that extends the first ACO algorithm by considering a local pheromone update at each node selection. The updating equation is similar to previous one but considering δ instead of α

and $\Delta\tau$ is equal to the initial value assigned to the pheromone trail. At each ant iteration ACS applies global update only to the arcs inserted in the best solution found. The maximum number of iterations is denoted as $AntRep$.
The first ACO algorithm was proposed in [6], [8] and [9] and was denominated $Ant–System$. A full review of ACO algorithms and applications can be found in [10]. ACS definition can be found at [7] and [12]. On ACS there has been a previous fine-tuning research, see [7], [10] and [19].
Note that a feasible set of parameters for running ACS is a combination of feasible q_0, φ, β, δ, α, ρ and number of ants (na). ρ defines the neighbor as follows: for a given node its neighbor set is defined by its $\rho\ |V|$ nearest nodes, where $\rho \in [0, 1]$.

2.2 *ParticleSwarmOptimization(PSO)*

As described by Eberhert and Kennedy [11], [16], [17], PSO is an adaptative algorithm based on a social environment where a set of particles, called population, are visiting possible "positions" of a given dominion. Each position has a fitness value (and it can be computed). At each iteration particles will move returning stochastically toward population best fitness position and its previous best fitness position. Particles of the population are sharing information of the best areas to search.
Let denote P as the set of parameters and let define PO as the population of particles. At each iteration x_{fp} and v_{fp} denotes respectively the actual value and the actual velocity of parameter $p \in P$ of the particle $f \in PO$. We denote x_f and v_f respectively as the actual position and the actual velocity of particle f. The movement of the particles are defined by the next equations:

$$v_{fp} = wv_{fp} + r_1c_1(x_{fp} - bl_{fp}) + r_2c_2(x_{fp} - bg_p), \tag{2}$$

$$x_{fp} = x_{fp} + \chi v_{fp}, \tag{3}$$

where c_1, c_2 are integer non-negative values, named *cognitive* and *social* respectively, r_1, r_2 are sample random values in [0,1], w and χ are non-negative real values, named respectively *inertia weight* and *constriction factor*, bg_p is the value of parameter p pertaining to the best set of parameters found by the population (social knowledge) and bl_{fp} is the value of parameter p pertaining to the best parameters set found by particle f (self-knowledge). In (2) the first factor refers to the previous velocity, second and third factors are related respectively to the distance to the best set of parameters found by the particle and to the distance to the best set of parameters found by the population.

3 *A–B–Domain* Framework

3.1 *A–B–Domain* Framework Design

To develop this new framework we work on an problem $Prob$. To solve any instance $inst$ of $Prob$ an algorithm B is used. Algorithm B is defined by its

internal structure and a number np of parameters. Be $Param$ the set of parameters, $Param = \{p_1, p_2, \ldots, p_{np}\}$ where each $p_i \in [p_{i(min)}, p_{i(max)}] \subset \Re$ (or $\aleph$). Then Domain can be defined as:

$$Domain = \prod_{1 \leq i \leq ns} [p_{i(min)}, p_{i(max)}] \subset \Re^{np}$$

Sets can be defined open or closed depending on the nature of the parameter. Let denote $fitness(p)$ for $p \in Domain$ as the value returned by running algorithm B on $inst$ using the set of parameters defined by p. Algorithm A is used to compute the minimum $fitness(p)$ for $p \in Domain$. Each time a fitness value is computed for a set of parameters $p \in Domain$ an algorithm B is performed, consequently it is computationally unavoidable to perform the check on all the points of the $Domain$. That is the main reason an heuristic algorithm A is used to compute the fitness on the most promising set of parameters. The new framework design previously explained is graphically developed in Figure 1: the generic framework is displayed in (a)and the example PSO–ACS–$Domain$ is shown in (b).

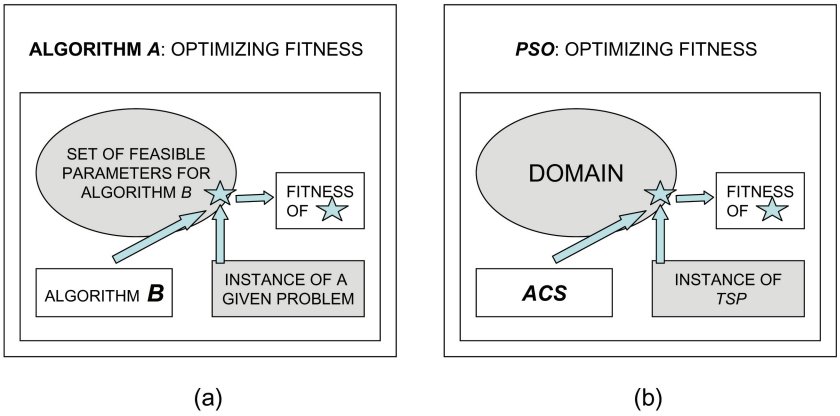

Fig. 1. General framework and example framework

3.2 *PSO–ACS–Domain* Example

The proposed framework is applied on the TSP problem and ACS algorithm is used as algorithm B. The $Domain$ is defined in Table 1 where the respective ranges are defined. as:

Those are, respectively, the feasible ranges of the parameters. For a given TSP instance $inst$ and $p \in Domain$, $fitness(p)$ is the length of the tour returned by the ACS algorithm on $inst$ using the set of parameters defined by p. Where

Table 1. Domain of ACS parameters

Parameter	q_0	$\times$	φ	$\times$	β	$\times$	δ	$\times$	α	$\times$	ρ	$\times$	na
Domain	[0,1]	$\times$	[0,9[	$\times$	[0,9[	$\times$	[0,1]	$\times$	[0,1]	$\times$	[0,1]	$\times$	[1,41[

integer values are necessary, as for φ, β and na, values are rounded down to the nearest integer value. Because ACS is an stochastic searching method, we cannot develop an exact evaluation for each point in the Domain, instead we can approximate the performance by several repeats of the algorithm. ACS is repeated ACS–*trial* number of times, and two fitness functions has been tested: the minimum value, denoted as $Eval1$, and the average value, denoted as $Eval2$. The pseudo-code of the algorithm works as shown in Figure 2.
This example works mainly in the PSO structure where each evaluation is found by running ACS a prefixed number of trials. The *inertia weight* is set initially to value 1 and gradually decreased from 1 to 0.1; at each PSO iteration $w = \epsilon \times w$ where $\epsilon \in]0, 1]$. Parameters have been fine-tuned as it is explained in computational results.

```
Select TSP-Instance.
Initialize particles.
Do while {Stop Condition = FALSE}
   For all the set of particles
      Position_Fitness (PF)=INFINITE
      Do ACS-trial
         Perform a trial ACS with particle's parameters.
         If  NewValue < PF
            PF=NewValue
         End if
      End Do
   End for
   Compute w=ε w
   Update Best Parameters Found by each Particle
   Update Best Parameters Found by the Population
   Compute Velocity
   Movement of Particles
   Check Stop Condition
End Do
Minimum value is returned.
```

Fig. 2. *PSO-ACS-Domain* Pseudocode

4 Statistical Research

Proposed framework needs a huge amount of computational time consequently any information extracted to reduce the search among the *Domain* is considered

very valuable. As a first approach marginal effects are reviewed for each parameter of the ACS, a full analysis about the joint effects are in progress. For each parameter we select a subset of "recommended values" extracted from [19], [10] and [15]. Table 2 shows the selected values. The experiment was run on a Pentium M 1,7 Ghz with 1Gb RAM and the experiment was designed as a factorial experiment with five trials at each combination of parameters. To discard any possible influence of the programming and the computer we randomly decided the order in which the parameters were going to be get into consideration. And for each parameter the possible order of the values were set randomly. The experiment was performed on three small instances: eil51, kroB100 and rat99.

Table 2. Values tested for each parameter

Parameter	Tested Values
q_0	0, 0.2, 0.4, 0.6, 0.8, 1
φ, β	0, 1, 2, 3
δ, α	0.01, 0.05, 0.1, 0.15, 0.20
ρ	0.33, 0.66, 1
na	5, 10, 15, 20

For each parameter, differences among variances where significative and no tested transformation improved homogeneity of the samples, consequently we selected the following post-hoc tests: Tamhanes T2 and Dunnetts T3. Results can be mainly resumed in two proposed new policies: *Domain-Reduced* and $No\text{-}More(perc)$. The main idea behind these policies is to analyze the effects of the information extracted from selected instances on a wider range of types of instances.

4.1 *Domain-Reduced* policy.

Domain-Reduced policy: Dominion is reduced by a statistical research. For the ACS the $DomainReduced$ is :
$Domain - R = [0.5, 1] \times [1, 7[\times[1, 7[\times[0, 0.1] \times [0.05, 0.25]\times]0, 0.75] \times [5, 15]$
For each parameter the proposed intervals following the results are defined as follows:

- δ: as much as this value increases the performance decreases, there is a clear difference when it goes over 0.10 so we set this parameter in the interval $[0, 0.10]$.

- α: as it decreases the performance decreases, the limit can be put at 0.05, so we recommend the interval $[0.05, 0.25]$ (upper limit is increased because we observed the best performance at value 0.20 indicating that possibly and

increased value would lead to better performance but it is not recommended to increase this value indefinitely).

- β and φ: the only "clear" agreement in all experiments indicates that value 0 is evidently decreasing the efficiency. So we set the interval to $[1, 6]$.

- na: performance decreases on extreme values, so we set the interval $[5, 15]$.

- q_0: values smaller than 0.5 are working clearly worse than the rest, 0.8 has the best performance. We recommend the interval $[0.5, 1]$.

- ρ: performance decreases as value increases, difference between 0.333 and 0.666 is not as bigger as between 0.666 and 1 so we set the interval $]0, 0.75]$.

4.2 *No-More(perc)* policy

Let introduce some notation: *best* denotes the best value found by the *A–B–Domain* algorithm at the moment, $fitness1(p)$ denotes the value obtained by algorithm B on the first trial where $p \in D$. We propose the following policy: *No–More(perc)*: after the first trial is performed following trials are discarded if Equation 4 accomplishes, where $perc \in [0,1]$. *perc* works as a percentage deviation from the best value obtained, if the deviation is more or equal to *perc* following trials are discarded.

$$perc > \frac{fitness1(p) - B}{fitness1(p)} \tag{4}$$

This policy works on reducing the computational cost by discarding non–promising computational experiments. Figure 3 displays the scatter plot of the first value (over five) against the average and the minimum values obtained for each combination of parameters considered (from instance kroB100, but graphics are similar for the rest of instances). It is clear that those values are closely related, consequently if the first iteration for any set of parameters can be classified as "bad" most probably the rest will be classified similarly. This fact can provide a reason to reduce the number of trials.

5 Computational Results

Computational results are in a preliminary stage, further experiments are being performed to fine-tune the PSO-ACS-Domain algorithm and to compare computational results with other algorithms. Algorithms were coded in C++ and ran in a Pentium M 1.7 with 1GB RAM. Experiments were designed to gain iteratively more information about the algorithm. At tables, notation works as follows: *GAP-Min* is the average of the *GAP* (considering *GAP* as $100 * (minimum\_obtained - lower\_bound)/lower\_bound$) for a given instance) , similar

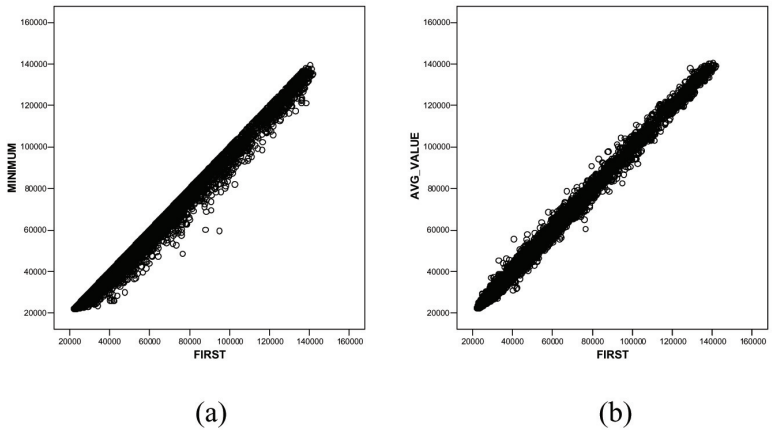

Fig. 3. (a) First vs Minimum (b) First vs Average

to previous definition but considering *average_obtained* instead of *minimum_obtained* is denoted as *GAP-Avg* and *CPU–Best* denotes the average time to find the minimum value obtained. We set $c_1 = c_2 = 2$, $w = 1$, $\chi = 0.729$, $\epsilon = 0.9$ on all the experiments. Initial parameters (particles) are set randomly on all experiments but the last because we wanted to check the performance without any previous information. Stop conditions are: 500 as the maximum number of iterations, 50 as the maximum number of iterations without improving and 2 000 seconds as the maximum computational time. For each particle of the population its initial velocity is set randomly.

5.1 An initial setting of parameters

At this experiment number of particles were set to 10. We compute all possible combinations from selected values/forms of *AntRep*, *ACS–trial* and evaluating forms. This experiment was performed in two parts.
In the first part we consider 100, 1 000, 3 000, 5 000, 10 000 values for *AntRep*; 1, 3, 5 and 10 values for *ACS–trial* and both evaluating forms. This first part of the experiment is performed on small instances to get an initial information about the algorithm: eil51, eil76, eil101, kroA100, kroB100, rat99.
Table 3 shows that there is no significant difference between evaluating forms. There is a clear difference (worst values) between 100, 1 and the rest of possible values for *AntRep* and *ACS–trial* respectively. Values 10 000 and 10 are discarded because they lead to high CPU usage with no significant improvement respectively for *AntRep* and *ACS–trial*. For the second part of the experiment following instances were selected: berlin52, a280, bier127, fl417, ch130, ch150, eil51, eil76, eil101, kroA100, kroB100, rat99; all use euclidian distance and are

Table 3. Results by Evaluation Methods

Evaluation	*GAP-Min*	*CPU-Best*
Eval1	0.0375	256.5
Eval2	0.0377	166.9

not exceeding 500 nodes. We compare the same parameters with the previous selected values.
Again there is no clear difference among evaluation methods even if now *Eval*2 shows higher computational cost. Among *ACS–trial* there is an slightly improvement between 3 and 5 with a *reasonable* increase in the *CPU–Best*. Among *AntRep* there is clear difference between 1 000 and the rest of values, consequently in further experiments we do not consider 1 000, see Table 4.

Table 4. Comparison of results among number of iterations of ants

AntRep	*GAP-Min*	*GAP-Avg*	*CPU-Best*
1 000	2.101	2.276	717.32
3 000	1.578	1.767	1149.14

5.2 Setting the number of particles.

In previous experiments the number of particles were set to 10, now we want to check if reducing this number the quality of the solution remains (with lower computational cost). We test 3, 5 and 10 particles with all the allowable combinations considering the last defined set of instances. Computational results are given in Table 5. Because maximum computational time was set to 2 000 seconds it is clear that 10 particles is performing worse and the rest of values are worth considering, even if 5 particles performs better.

Table 5. Comparison among number of particles

♯ particles	*GAP-Min*	*GAP-Avg*	*CPU-Best*
3	1.73	2.276	341.83
5	1.05	1.767	412.71
10	1.65	1.844	1058.39

5.3 Comparison between *PSO-ACS-Domain* and *ACS*

We compare results and CPU–$Best$ of the proposed algorithm against the simple ACS considering 5 000 $AntRep$ and using the parameters proposed at [10], [19]; we denote respectively ACS-1, ACS-2. For each of these sets we compute computational results for all $\rho \in \{0, 0.1, \ldots, 1\}$ and the best result is considered. At this experiment, half of the particles of the population are set initially equal to recommended sets of parameters from the the literature and the rest are initially set randomly. We consider eight versions of PSO-ACS-$Domain$ as shown in Table 5, where $Reduced$-$Domain$ (RD) and No-$more(perc)$ ($NM(perc)$) are the policies proposed in Section 4. AntRep, ACS-trial and number of particles are set respectively to 5 000, 5 and 5.
Computational results are given in table 6 (GM, GA and CA stand respectively for GAP-Min, GAP-Avg and CPU-$Best$) and we can observe that only when policy RD is considered the algorithm improves, on instances tested, GAP-Avg and in most instances also GAP-Min is improved. Computational times are higher as it was expected. Policy RD works well on improving fitness value and reducing computational time; Policy NM does not present strong modifications by itself but, $V4$, that considers both policies, has the best performance among the combinations considered.

Table 6. Versions considered

Name	RD	$NM(0.15)$
V1	No	No
V2	Yes	No
V3	No	Yes
V4	Yes	Yes

Table 7. Computational Results by Instance.

	ACS1			ACS2			V1			V2			V3			V4		
	GM	*GA*	*CPU*	*GM*	*GA*	*CPU*	*GM*	*GA*	*CPU*	*GM*	*GA*	*CPU*	*GM*	*GA*	*CPU*	*GM*	*GA*	*CPU*
eil51	0.00	0.19	1.0	0.00	0.19	2.3	0.00	0.12	1014.1	0.00	0.23	23.5	0.00	0.12	1012.0	0.00	**0.00**	23.9
eil76	0.00	0.76	0.4	0.00	0.00	0.7	0.74	1.02	1264.0	0.00	0.00	56.4	0.74	1.02	1249.4	0.00	0.00	56.4
ch130	0.52	1.34	27.3	0.42	1.20	25.3	2.75	3.63	834.0	**0.10**	**0.38**	1238.9	2.75	3.63	807.8	**0.10**	**0.38**	818.9
ch150	0.18	1.21	4.6	**0.00**	1.02	18.1	2.28	4.62	1833.2	0.06	0.07	786.5	2.28	4.62	1860.1	0.06	**0.07**	686.3
kroA100	0.12	0.69	13.7	0.00	0.76	5.9	6.27	8.93	1446.6	0.00	**0.00**	101.7	6.27	8.93	1445.2	0.00	**0.00**	81.6
kroB100	0.19	0.92	7.1	0.17	0.88	15.0	5.01	5.89	667.5	0.00	**0.09**	1365.0	5.01	5.89	667.0	0.00	**0.09**	1364.9
rat99	0.17	0.84	2.4	0.00	0.46	13.6	1.98	2.64	769.1	0.00	**0.00**	237.0	1.98	2.64	746.6	0.00	**0.00**	237.1
berlin52	0.00	0.76	0.4	0.00	0.00	0.7	0.00	0.00	156.5	0.00	0.00	4.8	0.00	0.00	156.4	0.00	0.00	4.8
d198	**1.06**	3.31	11.6	2.37	3.63	67.9	7.85	9.91	3264.2	1.24	**2.85**	684.9	7.85	9.91	3222.8	1.14	**2.85**	686.4

6 Conclusion

A new framework for heuristics have been proposed that theorically is able to perform well on a wider set of instances for any selected algorithm B; and a *PSO-ACS-Domain* example is developed. Computational results on preliminary versions of the algorithm show an improvement on the average deviations with an increase in the computational time comparing with ACS algorithm. Proposed policies to improve the algorithm have been tested and it has been showed that the performance is improved. Still it is necessary to perform further experiments to fine-tune the algorithm and to compare on a wider set of instances. Future research are: to increase the statistical research with the idea of implement more search-policies, to develop new examples that can lead to a general framework research, to extend the 2 000 seconds constraint and to consider local search methods on the ACS algorithm. As a final idea we propose that *A-B-Domain* framework can lead to better results if the computational time is not a primary constraint.

7 Acknowledgement

The contribution by D. Gómez–Cabrero has been partially supported by the AVCiT of the Generalitat Valenciana (Ref: GRUPOS03/174). The contribution by C. Armero has been partially supported by the Spanish Ministry of Science and Education under the Grant MTM2004-03290 and FEDER funds.

References

[1] Adenso-Díaz, B., Laguna, M.: Fine-tuning of Algorithms Using Fractional Experimental Designs and Local Search. To appear in Operations Research.

[2] Barr, R. S., Golden, B. L., Kelly, J. P., Resende M. G. C., Stewart, W. R.: Designing and Reporting on Computational Experiments with Heuristic Methods. J. Heuristics **1:1** (1995) 9–35

[3] Birrattari, M., Sttzle, T., Paquete, L., Varrentrapp, K.: A Racing Algorithm For Configuring Metaheuristics. In W. B. Langdon et al editors, GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann Publishers, San Francisco, CA, USA, (2002) 11–18

[4] Burke, E. K., Kendall, G.: Search Metodologies - Introductory Tutorials in Optimization and Decision Support Techniques. Springer (2005)

[5] Coy, S. P., Golden, B. L., Runger G. C., Wasil, E. A.: Using Experimental Design to Find Effective parameter settings for heuristics. Journal of Heuristics **7:1**(2001) 77–7

[6] Dorigo, M.: Optimization, Learning, and Natural Algorithms (in Italian). PhD Thesis, Dip. Elettronica, Politecnico di Milano, (1992)

[7] Dorigo, M., Gambardella, L.M. : Ant Colony System: A Cooperative Learning Aproach to the Travelling Salesman Problem. IEEE Transactions on Evolutionary Computation, **1:1** (1997) 53-66

[8] Dorigo, M., Maniezzo, V., Colorni, A. : Positive Feedback as a Search Strategy. Tech. Report 91016 Dip. Elettronica Politecnico di Milano (1991)

[9] Dorigo, M., Maniezzo, V., Colorni, A. : The Ant System: Optimization by a Colony of Cooperating Agents. IEEE Transactions on Systems, Man and Cybernetics, Part B, **26:1** (1996) 29–42
[10] Dorigo, M., Sttzle, T.: Ant Colony Optimization. MIT Press (2004)
[11] Eberhart, R.C., Simpson, P.K., Robbins, R.W.: Computational Intelligence PC Tools. Academic Press Professional Boston (1996).
[12] L.M. Gambardella and M. Dorigo, "Ant-Q: A reinforcement Learning Approach to the Symmetric and Asymmetric Travelling Salesman Problems", in Proceedings of the IEEE International Conference on Evolutionary Computation (ICEC'96). Morgan Kaufmann, (1995) 252–260 .
[13] E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns - Elements of Reusable Object-Oriented Software. Addison-Wesley (1995).
[14] Gies, D., Rahmat-Samii,,Y.: Particle Swarm Optimization for Reconfigurable Phase Differentiated Array Design. Microwave and Optical Technology Letters **38:3** (2003) 168–175
[15] Gómez-Cabrero, D., Ranashinge, D. N.: Fine-tuning the Ant Colony System algorithm through Particle Swarm Optimization. Proceedings of the International Conference on Information and Automation, 2005, Colombo, Sri Lanka.
[16] Kennedy, J.,Eberhart, R.C. : Particle Swarm Optimization. Proc. IEEE International Conference on Neural Networks, Piscataway, NJ, USA, (1995) 1942–1948
[17] Kennedy, J., Eberhart, R.C. : Swarm Intelligence. Morgan Kaufmann Publishers (2001)
[18] Laskari, E.C., Parsopoulos, K.E., Vrahatis, M.N.: Particle Swarm Optimization for Integer Programming. Proc. IEEE Congress on Evolutionary Computation (2002) 1582–1587
[19] Pilat, M. L., White, T.: Using Genetic Algorithms to optimize ACS-TSP. Proc. Third International Workshop on Ant Algorithms (2002) 282–287
[20] Reinelt, G. : TSPLIB- A travelling Salesman Problem Library for TSP Applications. ORSA Journal of Computing, **3** (1991) 376–384