Particle Swarm Optimization for Integer Programming

E.C. Laskari, K.E. Parsopoulos and M.N. Vrahatis
Department of Mathematics,
University of Patras Artificial Intelligence Research Center (UPAIRC),
GR-26110 Patras, Greece
{elena, kostasp, vrahatis}@math.upatras.gr

Abstract - The investigation of the performance of the Particle Swarm Optimization (PSO) method in Integer Programming problems, is the main theme of the present paper. Three variants of PSO are compared with the widely used Branch and Bound technique, on several Integer Programming test problems. Results indicate that PSO handles efficiently such problems, and in most cases it outperforms the Branch and Bound technique.

I. INTRODUCTION

A remarkably wide variety of problems can be represented as discrete optimization models [17]. An important area of application concerns the efficient management of a limited number of resources so as to increase productivity and/or profit. Such applications are encountered in Operational Research problems such as goods distribution, production scheduling, and machine sequencing. There are applications in mathematics to the subjects of combinatorics, graph theory and logic. Statistical applications include problems of data analysis and reliability. Recent scientific applications involve problems in molecular biology, high energy physics and x–ray crystallography. A political application concerns the division of a region into election districts [17]. Capital budgeting, portfolio analysis, network and VLSI circuit design, as well as automated production systems are some more applications in which Integer Programming problems are met [17].

Yet another, recent, and promising application is the training of neural networks with integer weights, where the activation function and weight values are confined in a narrow band of integers. Such neural networks are better suited for hardware implementations compared to real weight ones [26].

The Unconstrained Integer Programming problem can be defined as

$$\min_x f(x), \quad x \in S \subseteq \mathbb{Z}^n, \qquad (1)$$

where $\mathbb{Z}$ is the set of integers, and S is a not necessarily bounded set, which is considered as the feasible region. Maximization of Integer Programming problems is very common in the literature, but we will consider only the minimization case, since a maximization problem can be easily transformed to a minimization problem and vice versa. The problem defined in Eq. (1) is often called "All–Integer Programming Problem", since all the variables are integers, in contrast to the "Mixed–Integer Programming Problem", where some of the variables are real.

Optimization techniques developed for real search spaces can be applied on Integer Programming problems and determine the optimum solution by rounding off the real optimum values to the nearest integer [17], [28]. One of the most common deterministic approaches for tackling Integer Programming problems, is the Branch and Bound (BB) technique [10], [18], [28]. According to this technique, the initial feasible region is split into several sub–regions. For each one of these sub–regions, a constrained sub–problem is solved, treating the integer problem as a continuous one. The procedure is repeated until the real variables are fixed to integer values.

Evolutionary and Swarm Intelligence algorithms are stochastic optimization methods that involve algorithmic mechanisms similar to natural evolution and social behavior respectively. They can cope with problems that involve discontinuous objective functions and disjoint search spaces [7], [14], [30]. Genetic Algorithms (GA), Evolution Strategies (ES), and the Particle Swarm Optimizer (PSO) are the most common paradigms of such methods. GA and ES draw from principles of natural evolution which are regarded as rules in the optimization process. On the other hand, PSO is based on simulation of social behavior. Early approaches in the direction of Evolutionary Algorithms for Integer Programming are reported in [8], [11].

In GA, the potential solutions are encoded in binary bit strings. Since the integer search space, of the problem defined in Eq. (1), is potentially not bounded, the representation of a solution using a fixed length binary string is not feasible [29]. Alternatively, ES can be used, by embedding the search space $\mathbb{Z}^n$ into $\mathbb{R}^n$ and truncating the real values to integers. However, this approach is not always efficient due to the existence of features of ES, which contribute to the detection of real valued minima with arbitrary accuracy. These features are not always needed in integer spaces, since the smallest distance of two points, in ℓ_1–norm, is equal to 1 [29].

This paper aims to investigate, the performance of the PSO method on Integer Programming problems. The truncation of real values to integers seems not to affect significantly the performance of the method, as the experimental results indicate. Moreover, PSO outperforms the BB technique for most test problems.

The rest of the paper is organized as follows: in Section II, the PSO method is described. In Section III, the BB algorithm is briefly exposed. In Section IV, the experimental results are reported, and conclusions are reported in Section V.

II. THE PARTICLE SWARM OPTIMIZATION METHOD

PSO is a Swarm Intelligence method for global optimization. It differs from other well–known Evolutionary Algorithms (EA) [2], [4], [7], [14], [30]. As in EA, a population of potential solutions is used to probe the search space, but no operators, inspired by evolution procedures, are applied on the population to generate new promising solutions. Instead, in PSO, each individual, named *particle*, of the population, called *swarm*, adjusts its trajectory toward its own previous best position, and toward the previous best position attained by any member of its topological neighborhood [12]. In the global variant of PSO, the whole swarm is considered as the neighborhood. Thus, global sharing of information takes place and the particles profit from the discoveries and previous experience of all other companions during the search for promising regions of the landscape. For example, in the single–objective minimization case, such regions possess lower function values than others, visited previously.

Several variants of the PSO technique have been proposed so far, following Eberhart and Kennedy [4], [13], [14]. In our experiments, three different global versions of PSO were investigated. They are all defined using the equations, described in the following paragraph [14].

First, let us define the notation adopted in this paper: assuming that the search space is D–dimensional, the i-th particle of the swarm is represented by the D–dimensional vector $X_i = (x_{i1}, x_{i2}, \ldots, x_{iD})$ and the best particle of the swarm, i.e. the particle with the smallest function value, is denoted by index g. The best previous position (i.e. the position giving the lowest function value) of the i-th particle is recorded and represented as $P_i = (p_{i1}, p_{i2}, \ldots, p_{iD})$, and the position change (velocity) of the i-th particle is $V_i = (v_{i1}, v_{i2}, \ldots, v_{iD})$.

The particles are manipulated according to the following equations (the superscripts denote the iteration):

$$\begin{aligned} V_i^{n+1} &= wV_i^n + c_1 r_{i1}^n (P_i^n - X_i^n) + \\ &\quad + c_2 r_{i2}^n (P_g^n - X_i^n), \qquad (2) \\ X_i^{n+1} &= X_i^n + \chi\, V_i^{n+1}, \qquad (3) \end{aligned}$$

where $i = 1, 2, \ldots, N$; N is the swarm's size; χ is a *constriction factor* used to control and constrict velocities; w is the *inertia weight*; c_1 and c_2 are two positive constants, called the *cognitive* and *social* parameter respectively; r_{i1}^n and r_{i2}^n are two random numbers uniformly distributed within the range $[0, 1]$.

Eq. (2) is used to calculate at each iteration, the i-th particle's new velocity. Three terms are taken into consideration. The first term, wV_i^n, is the particle's previous velocity weighted by the inertia weight w. The second term, $(P_i^n - X_i^n)$, is the distance between the particle's best previous position, and its current position. Finally, the third term, $(P_g^n - X_i^n)$, is the distance between the swarm's best experience, and the i-th particle's current position. The parameters $c_1 r_{i1}^n$, $c_2 r_{i2}^n$, provide randomness that render the technique less predictable but more flexible [12]. Eq. (3) provides the new position of the i-th particle, adding its new velocity, to its current position. In general, the performance of each particle is measured according to a fitness function, which is problem–dependent. In optimization problems, the fitness function is usually the objective function under consideration.

The role of the inertia weight w is considered crucial for PSO's convergence behavior. The inertia weight is employed to control the impact of the history of velocities on the current velocity. In this way, the parameter w regulates the trade–off between the global (wide–ranging) and the local (nearby) exploration abilities of the swarm. A large inertia weight facilitates global exploration (searching new areas), while a small one tends to facilitate local exploration, i.e. fine–tuning the current search area. A suitable value for the inertia weight w provides balance between the global and local exploration ability of the swarm, resulting in better convergence rates. Experimental results suggest that it is better to set the inertia to a large initial value, in order to promote global exploration of the search space, and gradually decrease it to obtain refined solutions [31]. Our approach, employs a time–decreasing inertia weight value.

The initial population, as well as the velocities, can be generated either randomly or by a Sobol sequence generator [27], which ensures that the D-dimensional vectors will be uniformly distributed within the search space.

Some variants of PSO impose a maximum allowed velocity V_{max} to prevent the swarm from exploding. Thus, if $v_{id}^{n+1} > V_{max}$ in Eq. (2), then $v_{id}^{n+1} = V_{max}$ [14].

PSO resembles, to some extent, EA. Although it does not rely on a direct recombination operator, the recombination concept is accounted for by the stochastic movement of each particle toward its own best previous position, as well as toward the global best position of the entire swarm or its neighborhood's best position, depending on the variant of the PSO that is used [6]. Moreover, PSO's mutation–like behavior is directional, due to the velocity of each particle, with a kind of momentum built in. In other words, PSO is considered as performing mutation with a "conscience", as pointed out by Eberhart and Shi [6].

The PSO technique has proved to be very effective in solving real valued global optimization problems, in static, noisy as well as continuously changing environments, and for performing neural networks training [19]–[22], exhibiting competitive results with the EA [1]. Moreover, it can cope efficiently with Multiobjective Optimization problems [25] and specialized problems, like the ℓ_1 norm errors–in–variables problems [23]. Its convergence rates can be improved by properly initializing the population e.g. using a derivative–free method like the Nonlinear Simplex Method of Nelder and Mead [24].

III. THE BRANCH AND BOUND TECHNIQUE

The BB technique is widely used for solving optimization problems. In BB, the feasible region of the problem is relaxed, and subsequently partitioned into several sub–regions; this is called *branching*. Over these sub–regions, lower and upper bounds for the values of the function can be determined; this is the *bounding* part of the algorithm.

The BB technique can be algorithmically sketched as follows [3], [15], [16]:

Step 1. Start with a relaxed feasible region $M_0 \supset S$ and partition M_0 into finitely many subsets $M_i, i = 1, 2, \ldots, m$, where S is the feasible region of the problem.

Step 2. For each subset M_i, determine lower (and if possible) upper bounds, $\beta(M_i)$ and $\alpha(M_i)$, respectively, satisfying

$$\beta(M_i) \leqslant \inf f(M_i \cap S) \leqslant \alpha(M_i),$$

where f is the objective function under consideration.

Then, the bounds defined as

$$\beta := \min_{i=1,2,\ldots,m} \beta(M_i),$$

and

$$\alpha := \min_{i=1,2,\ldots,m} \alpha(M_i),$$

are "overall" bounds, i.e.

$$\beta \leqslant \min f(S) \leqslant \alpha.$$

Step 3. If $\alpha = \beta$ (or $\alpha - \beta \leqslant \varepsilon$, for a predefined constant $\varepsilon > 0$), then stop.

Step 4. Otherwise, choose some of the subsets M_i and partition them, in order to obtain a more refined partition of M_0. Determine new (hopefully better) bounds on the new partition elements, and repeat the procedure.

An advantage of the BB technique is that, during the iteration process, one can usually delete subsets of S, in which, the minimum of f cannot be attained. Important issues that arise during the BB procedure are that of properly partitioning the feasible region and selecting which sub–problem to evaluate.

The BB technique has been successfully applied to Integer Programming problems. The algorithm applied in this paper, transforms the initial integer problem to a continuous one. Consecutively, following the prescribed procedure, it restricts the domain of the variables, which are still considered continuous, and solves the generated sub–problems using the Sequential Quadratic Programming method. This process is repeated until the variables are fixed to an integer value. For the branching, Depth–First traversal with backtracking was used.

IV. EXPERIMENTAL RESULTS

Seven Integer Programming test problems were selected to investigate the performance of the PSO method. Each particle of the swarm was truncated to the closest integer, after the determination of its new position using Eq. (3).

The considered test problems, defined immediately below, are frequently encountered in the relevant literature:

TEST PROBLEM 1, [29]:

$$F_1(x) = \|x\|_1 = |x_1| + \ldots + |x_D|,$$

with $x = (x_1, \ldots, x_D) \in [-100, 100]^D$, where D is the corresponding dimension. The solution is $x_i^* = 0$, $i =$

$1, \dots, D$, with $F_1(x^*) = 0$. This problem was considered in dimensions 5, 10, 15, 20, 25, and 30.

Test Problem 2, [29]:

$$F_2(x) = x^\top x = \begin{pmatrix} x_1 & \dots & x_D \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_D \end{pmatrix},$$

with $x = (x_1, \dots, x_D)^\top \in [-100, 100]^D$, where D is the corresponding dimension. The solution is $x_i^* = 0$, $i = 1, \dots, D$, with $F_2(x^*) = 0$. This is a quite trivial problem and it was considered in dimension 5.

Test Problem 3, [9]:

$$\begin{aligned} F_3(x) &= -\begin{pmatrix} 15 & 27 & 36 & 18 & 12 \end{pmatrix} x + \\ &+ x^\top \begin{pmatrix} 35 & -20 & -10 & 32 & -10 \\ -20 & 40 & -6 & -31 & 32 \\ -10 & -6 & 11 & -6 & -10 \\ 32 & -31 & -6 & 38 & -20 \\ -10 & 32 & -10 & -20 & 31 \end{pmatrix} x. \end{aligned}$$

with best known solutions $x^* = (0, 11, 22, 16, 6)^\top$ and $x^* = (0, 12, 23, 17, 6)^\top$, with $F_3(x^*) = -737$.

Test Problem 4, [9]:

$$F_4(x) = (9x_1^2 + 2x_2^2 - 11)^2 + (3x_1 + 4x_2^2 - 7)^2,$$

with solution $x^* = (1, 1)^\top$ and $F_4(x^*) = 0$.

Test Problem 5, [9]:

$$\begin{aligned} F_5(x) &= (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + \\ &+ (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4, \end{aligned}$$

with solution $x^* = (0, 0, 0, 0)^\top$ and $F_5(x^*) = 0$.

Test Problem 6, [28]:

$$F_6(x) = 2x_1^2 + 3x_2^2 + 4x_1x_2 - 6x_1 - 3x_2,$$

with solution $x^* = (2, -1)^\top$ and $F_6(x^*) = -6$.

Test Problem 7, [9]:

$$\begin{aligned} F_7(x) &= -3803.84 - 138.08x_1 - 232.92x_2 + 123.08x_1^2 + \\ &+ 203.64x_2^2 + 182.25x_1x_2, \end{aligned}$$

TABLE I
SUCCESS RATE, MEAN NUMBER, STANDARD DEVIATION, AND MEDIAN OF FUNCTION EVALUATIONS, FOR THE TEST PROBLEM F_1.

Function	Method	Succ.	Mean	St.D.	Median
F_1 5 dim	PSO-In	30/30	1646.0	661.5	1420
	PSO-Co	30/30	744.0	89.8	730
	PSO-Bo	30/30	692.6	97.2	680
	BB	30/30	1167.83	659.8	1166
F_1 10 dim	PSO-In	30/30	4652.0	483.2	4610
	PSO-Co	30/30	1362.6	254.7	1360
	PSO-Bo	30/30	1208.6	162.7	1230
	BB	30/30	5495.8	1676.3	5154
F_1 15 dim	PSO-In	30/30	7916.6	624.1	7950
	PSO-Co	30/30	3538.3	526.6	3500
	PSO-Bo	30/30	2860.0	220.2	2850
	BB	30/30	10177.1	2393.4	10011
F_1 20 dim	PSO-In	30/30	8991.6	673.3	9050
	PSO-Co	30/30	4871.6	743.3	4700
	PSO-Bo	29/30	4408.3	3919.4	3650
	BB	30/30	16291.3	3797.9	14550
F_1 25 dim	PSO-In	30/30	11886.6	543.7	11900
	PSO-Co	30/30	9686.6	960.1	9450
	PSO-Bo	25/30	9553.3	7098.6	6500
	BB	20/30	23689.7	2574.2	25043
F_1 30 dim	PSO-In	30/30	13186.6	667.8	13050
	PSO-Co	30/30	12586.6	1734.9	12500
	PSO-Bo	19/30	13660.0	8863.9	7500
	BB	14/30	25908.6	755.5	26078

with solution $x^* = (0, 1)^\top$ and $F_7(x^*) = -3833.12$.

Three variants of PSO were used in the experiments: one with inertia weight and without constriction factor, denoted as PSO-In; one with constriction factor and without inertia weight, denoted as PSO-Co; and one with both constriction factor and inertia weight, denoted as PSO-Bo. For all experiments, the maximum number of allowed function evaluations was set to 25000; the desired accuracy was 10^{-6}; the constriction factor χ was set equal to 0.729; the inertia weight w was gradually decreased from 1 towards 0.1; $c_1 = c_2 = 2$; and $V_{max} = 4$. The aforementioned values for all PSO's parameters are considered default values, and they are used widely in the relevant literature [14]. There was no preprocessing stage that might yield more suitable values for the parameters. For each test problem, 30 experiments were performed, starting with a swarm and velocities uniformly distributed within the range $[-100, 100]^D$, where D is the dimension of the corresponding problem, and truncated to the nearest integer.

For the BB algorithm, 30 experiments were performed for each test problem, starting from a randomly selected point within $[-100, 100]^D$, and truncated to the nearest

TABLE II
SUCCESS RATE, MEAN NUMBER, STANDARD DEVIATION, AND MEDIAN OF FUNCTION EVALUATIONS, FOR THE TEST PROBLEMS F_2–F_7.

Function	Method	Succ.	Mean	St.D.	Median
F_2 5 dim	PSO-In	30/30	1655.6	618.4	1650
	PSO-Co	30/30	428.0	57.9	430
	PSO-Bo	30/30	418.3	83.9	395
	BB	30/30	139.7	102.6	93
F_3	PSO-In	30/30	4111.3	1186.7	3850
	PSO-Co	30/30	2972.6	536.4	2940
	PSO-Bo	30/30	3171.0	493.6	3080
	BB	30/30	4185.5	32.8	4191
F_4	PSO-In	30/30	304.0	101.6	320
	PSO-Co	30/30	297.3	50.8	290
	PSO-Bo	30/30	302.0	80.5	320
	BB	30/30	316.9	125.4	386
F_5	PSO-In	30/30	1728.6	518.9	1760
	PSO-Co	30/30	1100.6	229.2	1090
	PSO-Bo	30/30	1082.0	295.6	1090
	BB	30/30	2754.0	1030.1	2714
F_6	PSO-In	30/30	178.0	41.9	180
	PSO-Co	30/30	198.6	59.2	195
	PSO-Bo	30/30	191.0	65.9	190
	BB	30/30	211.1	15.0	209
F_7	PSO-In	30/30	334.6	95.5	340
	PSO-Co	30/30	324.0	78.5	320
	PSO-Bo	30/30	306.6	96.7	300
	BB	30/30	358.6	14.7	355

integer. The maximum number of allowed function evaluations and the desired accuracy were the same as for PSO.

For both algorithms, the number of successes in detecting the integer global minimum of the corresponding problem, within the maximum number of function evaluations, the mean, the standard deviation, and the median of the required number of function evaluations, were recorded and they are reported in Tables I and II. The swarm's size was problem dependent. The swarm's size for each test problem is reported in Table III. It should be noted at this point, that although the swarm's size was problem dependent, the maximum number of allowed function evaluations was equal to 25000, for all cases.

In a second round of experiments, a PSO with gradually truncated particles was used. Specifically, the particles for the first 50 iterations were rounded to 6 decimal digits (d.d.), for another 100 iterations they were rounded to 4 d.d., for another 100 iterations they were rounded to 2 d.d., and for the rest iterations they were rounded to the nearest integer. The results obtained using this gradually truncated variant of PSO, were almost similar to the results reported in Tables I and II for the plain PSO.

TABLE III
DIMENSION AND SWARM'S SIZE FOR ALL TEST PROBLEMS.

Function	Dimension	Swarm's Size
F_1	5	20
F_1	10	20
F_1	15	50
F_1	20	50
F_1	25	100
F_1	30	100
F_2	5	10
F_3	5	70
F_4	2	20
F_5	4	20
F_6	2	10
F_7	2	20

V. CONCLUSIONS

The ability of the PSO method to cope with Integer Programming problems formed the core of the paper. Experimental results for seven widely used test problems indicate that PSO is a very effective method and should be considered as a good alternative to handle such problems.

The behavior of PSO seems to be stable even for high dimensional cases, exhibiting high success rates even in cases in which the BB technique failed. In most cases, PSO outperformed the BB approach, by means of the mean number of required function evaluations.

Moreover, the method appears not seem to suffer from search stagnation. The aggregate movement of each particle towards its own best position and the best position ever attained by the swarm, added to its weighted previous position change, ensures that particles maintain a position change during the process of optimization, which is of proper magnitude.

Regarding the three different variants of PSO, PSO-Bo, which utilizes both inertia weight and constriction factor, was the fastest, but the other two approaches posses better global convergence abilities, especially in high dimensional problems. In most experiments, PSO-Co, which utilizes only a constriction factor, was significantly faster than PSO-In, which utilizes only inertia weight.

In general, PSO seems an efficient alternative for solving Integer Programming problems, when deterministic approaches fail, or it could be considered as an algorithm for providing good initial points to deterministic methods, as the BB technique, and thus, help them converge to the global minimizer of the integer problem.

VI. ACKNOWLEDGEMENT

Part of this work was done while the authors (K.E.P. and M.N.V.) were at the Department of Computer Science, University of Dortmund, D–44221 Dortmund, Germany. This material was partially supported by the Deutsche Forschungsgemeinschaft–DFG (German National Research Foundation) as a part of the collaborative research center "Computational Intelligence" (SFB 531).

References

[1] P.J. Angeline, "Evolutionary Optimization Versus Particle Swarm Optimization: Philosophy and Performance Differences", Evolutionary Programming VII, pp. 601–610, 1998.

[2] W. Banzhaf, P. Nordin, R.E. Keller and F.D. Francone, Genetic Programming–An Introduction, Morgan Kaufmann: San Francisco, 1998.

[3] B. Borchers and J.E. Mitchell, "Using an Interior Point Method In a Branch and Bound Algorithm For Integer Programming", Technical Report, Rensselaer Polytechnic Institute, July 1992.

[4] R.C. Eberhart, P.K. Simpson and R.W. Dobbins, Computational Intelligence PC Tools, Academic Press Professional: Boston, 1996.

[5] R.C. Eberhart and Y.H. Shi, "Evolving Artificial Neural Networks", Proc. Int. Conf. on Neural Networks and Brain, Beijing, P.R. China, 1998.

[6] R.C. Eberhart and Y.H. Shi, "Comparison Between Genetic Algorithms and Particle Swarm Optimization", Evolutionary Programming VII, pp. 611–615, 1998.

[7] D.B. Fogel, Evolutionary Computation: Toward a New Philosophy of Machine Intelligence, IEEE Press: New York, 1995.

[8] D.A. Gall, "A Practical Multifactor Optimization Criterion", A. Levi, T.P. Vogl (Eds.), Recent Advances in Optimization Techniques, pp. 369–386, 1966.

[9] A. Glankwahmdee, J.S. Liebman and G.L. Hogg, "Unconstrained Discrete Nonlinear Programming", *Engineering Optimization*, Vol. 4, pp. 95–107, 1979.

[10] R. Horst and H. Tuy, Global Optimization, Deterministic Approaches, Springer, 1996.

[11] R.C. Kelahan and J.L. Gaddy, "Application of the Adaptive Random Search to Discrete and Mixed Integer Optimization", *International Journal for Numerical Methods in Enginnering*, Vol. 12, pp. 289–298, 1978.

[12] J. Kennedy, "The Behavior of Particles", Evolutionary Programming VII, pp. 581–587, 1998.

[13] J. Kennedy and R.C. Eberhart, "Particle Swarm Optimization", Proc. of the IEEE International Conference on Neural Networks, Piscataway, NJ, USA, pp. 1942–1948, 1995.

[14] J. Kennedy and R.C. Eberhart, Swarm Intelligence, Morgan Kaufmann Publishers, 2001.

[15] E.L. Lawler and D.W. Wood, "Branch and Bound Methods: A Survey", *Operations Research*, Vol. 14, pp. 699–719, 1966.

[16] V.M. Manquinho, J.P. Marques Silva, A.L. Oliveira ans K.A. Sakallah, "Branch and Bound Algorithms for Highly Constrained Integer Programs", Technical Report, Cadence European Laboratories, Portugal, 1997.

[17] G.L. Nemhauser, A.H.G. Rinnooy Kan and M.J. Todd (Eds.), Handbooks in OR & MS, Vol. 1: Optimization, Elsevier, 1989.

[18] G.L. Nemhauser and L.A. Wolsey, Integer and Combinatorial Optimization, John Wiley and Sons, 1988.

[19] K.E. Parsopoulos, V.P. Plagianakos, G.D. Magoulas and M.N. Vrahatis, "Objective Function "Stretching" to Alleviate Convergence to Local Minima", *Nonlinear Analysis TMA*, Vol. 47(5), pp. 3419–3424, 2001.

[20] K.E. Parsopoulos, V.P. Plagianakos, G.D. Magoulas and M.N. Vrahatis, "Stretching Technique for Obtaining Global Minimizers Through Particle Swarm Optimization", Proc. of the Particle Swarm Optimization Workshop, Indianapolis (IN), USA, pp. 22–29, 2001.

[21] K.E. Parsopoulos and M.N. Vrahatis, "Modification of the Particle Swarm Optimizer for Locating All the Global Minima", V. Kurkova, N. Steele, R. Neruda, M. Karny (Eds.), Artificial Neural Networks and Genetic Algorithms, Springer: Wien (Computer Science Series), pp. 324–327, 2001.

[22] K.E. Parsopoulos and M.N. Vrahatis, "Particle Swarm Optimizer in Noisy and Continuously Changing Environments", M.H. Hamza (Ed.), Artificial Intelligence and Soft Computing, IASTED/ACTA Press, pp. 289–294, 2001.

[23] K.E. Parsopoulos, E.C. Laskari and M.N. Vrahatis, "Solving ℓ_1 Norm Errors-In-Variables Problems Using Particle Swarm Optimizer", M.H. Hamza (Ed.), Artificial Intelligence and Applications, IASTED/ACTA Press, pp. 185–190, 2001.

[24] K.E. Parsopoulos and M.N. Vrahatis, "Initializing the Particle Swarm Optimizer Using the Nonlinear Simplex Method", Proc. WSES Evolutionary Computation 2002 Conference, Interlaken, Switzerland, in press.

[25] K.E. Parsopoulos and M.N. Vrahatis, "Particle Swarm Optimization Method in Multiobjective Problems", ACM SAC 2002 Conference, Madrid, Spain, in press.

[26] V.P. Plagianakos and M.N. Vrahatis, "Training Neural Networks with Threshold Activation Functions and Constrained Integer Weights", Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN 2000), Como, Italy, 2000.

[27] W.H. Press, W.T. Vetterling, S.A. Teukolsky and B.P. Flannery, Numerical Recipes in Fortran 77, Cambridge University Press: Cambridge, 1992.

[28] S.S. Rao, Engineering Optimization–Theory and Practice, Wiley Eastern: New Delhi, 1996.

[29] G. Rüdolph, "An Evolutionary Algorithm for Integer Programming",Y. Davidor, H.–P. Schwefel, R. Männer (Eds.), Parallel Problem Solving from Nature 3, pp. 139–148, Springer, 1994.

[30] H.-P. Schwefel, Evolution and Optimum Seeking, Wiley, 1995.

[31] Y. Shi and R.C. Eberhart, "Parameter Selection in Particle Swarm Optimization", EvolutionaryProgramming VII, pp. 591–600, 1998.