MOPSO : A Proposal for Multiple Objective Particle Swarm Optimization

Carlos A. Coello Coello
CINVESTAV-IPN
Depto. de Ing. Elect./Sección de Computación
Av. IPN No. 2508, Col. San Pedro Zacatenco
México, D. F. 07300
ccoello@cs.cinvestav.mx

Maximino Salazar Lechuga
Maestría en Inteligencia Artificial
Sebastián Camacho No. 5
LANIA-Universidad Veracruzana
Xalapa, Veracruz, México 91090
msalazar@mail.mia.uv.mx

Abstract- This paper introduces a proposal to extend the heuristic called "particle swarm optimization" (PSO) to deal with multiobjective optimization problems. Our approach uses the concept of Pareto dominance to determine the flight direction of a particle and it maintains previously found nondominated vectors in a global repository that is later used by other particles to guide their own flight. The approach is validated using several standard test functions from the specialized literature. Our results indicate that our approach is highly competitive with current evolutionary multiobjective optimization techniques.

1 Introduction

The use and development of heuristics-based multiobjective optimization techniques has significantly grown in the last few years [2]. One of the recent research trends has been to produce algorithms that are more efficient. This aim for efficiency normally requires clever techniques to maintain diversity (such as the adaptive grid used by PAES [8]) and the use of small population sizes [1]. Particle swarm optimization (PSO) is a relatively recent heuristic inspired by the choreography of a bird flock. Despite its current success in diverse optimization tasks [7], PSO remains as one of the heuristics for which not much work on multiobjective optimization has been done so far.

In this paper, we present a proposal, called "multiobjective particle swarm optimization" (MOPSO), which allows the PSO algorithm to be able to deal with multiobjective optimization problems. The approach is relatively simple to implement, it is population-based, it uses an external memory (called "repository") and a geographically-based approach to maintain diversity. MOPSO is validated using some standard test functions reported in the specialized literature and compared against two highly competitive EMO algorithms: the Pareto Archived Evolution Strategy [8] (PAES) and the Non-dominated Sorting Genetic Algorithm II [4] (NSGA II).

2 Particle Swarm Optimization

Kennedy & Eberhart [7] proposed an approach called "particle swarm optimization" (PSO) which was inspired on the choreography of a bird flock. The approach can be seen as a distributed behavioral algorithm that performs (in its more general version) multidimensional search. In the simulation, the behavior of each individual is affected by either the best local (i.e., within a certain neighborhood) or the best global individual. The approach uses then the concept of population and a measure of performance similar to the fitness value used with evolutionary algorithms. Also, the adjustments of individuals are analogous to the use of a crossover operator. However, this approach introduces the use of flying potential solutions through hyperspace (used to accelerate convergence) which does not seem to have an analogous mechanism in traditional evolutionary algorithms. Another important difference is the fact that PSO allows individuals to benefit from their past experiences whereas in an evolutionary algorithm, normally the current population is the only "memory" used by the individuals. PSO has been successfully used for both continuous nonlinear and discrete binary single-objective optimization [7]. Particle swarm optimization seems particularly suitable for multiobjective optimization mainly because of the high speed of convergence that the algorithm presents for single-objective optimization [7].

To the best of our knowledge, there have been only two previous proposals to extend PSO to handle multiobjective objectives. One of them is an unpublished document [9], and the other one is a paper yet to be published [10]. In both cases, Pareto dominance is used to generate a list of leaders that guide the search. Note, however, that in neither of these cases the proposed algorithm uses a secondary population nor is compared against other evolutionary multiobjective techniques using standard test functions and metrics.

3 Description of the Approach

The analogy of particle swarm optimization with evolutionary algorithms makes evident the notion that using

a Pareto ranking scheme [6] could be the straightforward way to extend the approach to handle multiobjective optimization problems. The historical record of best solutions found by a particle (i.e., an individual) could be used to store nondominated solutions generated in the past (this would be similar to the notion of elitism used in evolutionary multiobjective optimization). The use of global attraction mechanisms combined with a historical archive of previously found nondominated vectors would motivate convergence towards globally nondominated solutions.

Therefore, our proposal is based on the idea of having a global repository in which every particle will deposit its flight experiences after each flight cycle. Additionally, the updates to the repository are performed considering a geographically-based system defined in terms of the objective function values of each individual. This technique is inspired on the external file used with the Pareto Archive Evolution Strategy (PAES) [8]. The repository previously mentioned is used by the particles to identify a leader that will guide the search. We implemented a mechanism such that each particle may choose a different guide. Our mechanism is based on the generation of hypercubes which are produced dividing the search space explored.

The algorithm of MOPSO is the following:

1. Initialize the population POP:
   (a) FOR $i = 0$ TO MAX /* MAX = number of particles */
   (b) Initialize $POP[i]$
2. Initialize the speed of each particle:
   (a) FOR $i = 0$ TO MAX
   (b) $VEL[i] = 0$
3. Evaluate each of the particles in POP.
4. Store the positions of the particles that represent nondominated vectors in the repository REP.
5. Generate hypercubes of the search space explored so far, and locate the particles using these hypercubes as a coordinate system where each particle's coordinates are defined according to the values of its objective functions.
6. Initialize the memory of each particle (this memory serves as a guide to travel through the search space. This memory is also stored in the repository):
   (a) FOR $i = 0$ TO MAX
   (b) $PBESTS[i] = POP[i]$
7. WHILE maximum number of cycles has not been reached DO
   (a) Compute the speed of each particle[1] using the following expression:

$$VEL[i] = W \times VEL[i] + R_1 \times (PBESTS[i] - POP[i]) + R_2 \times (REP[h] - POP[i])$$

   where W (inertia weight) takes a value of 0.4; R_1 and R_2 are random numbers in the range [0..1]; $PBESTS[i]$ is the best position that the particle i has had[2]; $REP[h]$ is a value that is taken from the repository; the index h is selected in the following way: those hypercubes containing more than one particle are assigned a fitness equal to the result of dividing any number $x > 1$ (we used $x = 10$ in our experiments) by the number of particles that they contain. This aims to decrease the fitness of those hypercubes that contain more particles and it can be seen as a form of fitness sharing [5]. Then, we apply roulette-wheel selection using these fitness values to select the hypercube from which we will take the corresponding particle. Once the hypercube has been selected, we select randomly a particle within such hypercube. $POP[i]$ is the current value of the particle i.
   (b) Compute the new positions of the particles adding the speed produced from the previous step:

$$POP[i] = POP[i] + VEL[i] \quad (1)$$

   (c) Maintain the particles within the search space in case they go beyond its boundaries (avoid generating solutions that do not lie on valid search space).
   (d) Evaluate each of the particles in POP.
   (e) Update the contents of REP together with the geographical representation of the particles within the hypercubes. This update consists of inserting all the currently nondominated locations into the repository. Any dominated locations from the repository are eliminated in the process. Since the size of the repository is limited, whenever it gets full, we apply a secondary criterion for retention: those particles located in less populated areas of objective space are given priority over those lying in highly populated regions.

[1] Each particle has a dimensionality that can vary depending on the problem solved. When we say that we compute the speed of a particle, we refer to computing the speed for each of its dimensions.

[2] We will explain later on how do we define "better" in this context.

(f) When the current position of the particle is better than the position contained in its memory, the particle's position is updated using:

$$PBESTS[i] = POP[i] \quad (2)$$

The criterion to decide what position from memory should be retained is simply to apply Pareto dominance (i.e., if the current position is dominated by the position in memory, then the position in memory is kept; otherwise, the current position replaces the one in memory; if neither of them is dominated by the other, then we select one of them randomly).

(g) Increment the loop counter

8. END WHILE

4 Comparison of Results

Several test functions were taken from the specialized literature to compare our approach. However, due to space limitations, only three were included here. To compare our results in a quantitative way we used two criteria: average running time of the algorithm (using the same number of fitness function evaluations), and the following metric defined in objective space by Zitzler et al. [12]:

$$M_1^* = \frac{1}{\mid Y' \mid} \sum_{\mathbf{d}' \in Y'} \min \left\{ \parallel \mathbf{d}' - \bar{\mathbf{d}} \parallel^*; \bar{\mathbf{d}} \in \bar{Y} \right\} \quad (3)$$

where: $Y', \bar{Y} \subseteq Y$ are the sets of objective vectors that correspond to a set of pairwise nondominating decision vectors $X', \bar{X} \subseteq X$, respectively, and X corresponds to the decision variables of the problem. It should be obvious that M_1^* gives the average distance to the Pareto optimal set. Therefore, we should aim to minimize this value (see [12] for further details). To compute M_1^*, we generated the global Pareto front for each of the test functions by enumeration.

MOPSO was compared against two recent algorithms that are representative of the state of the art in evolutionary multiobjective optimization: the NSGA II [4] and PAES[3] [8].

In the following examples, the NSGA II was run using a population size of 200, a crossover rate of 0.8, tournament selection, and a mutation rate of $1/vars$, where $vars$ = number of decision variables of the problem. In the following examples, PAES was run using a depth of five, a size of the archive of 200, and a mutation rate of $1/L$, where L refers to the length of the chromosomic string that encodes the decision variables.

MOPSO[4] used a population of 40 particles, a repository size of 200 particles and 30 divisions for the adaptive grid. Our implementation uses a real-numbers representation and it is therefore intended for continuous search spaces. Note, however, that PSO can also be used with binary representation (see [7] for details).

To allow a fair comparison of running times, all the experiments were performed on a PC with a Pentium II processor running at 333 MHz, 128 Mb of RAM and a hard drive of 6 Gbytes. Our implementation was compiled using GNU C running under Linux Red Hat release 7.1.

4.1 Test Function 1

Our first test function was proposed by Deb [3]:

$$\text{Min } f_1(x_1, x_2) = x_1 \quad (4)$$

$$\text{Min } f_2(x_1, x_2) = (1 + 10x_2) \times \left[1 - \left(\frac{x_1}{1 + 10x_2}\right)^{\alpha} - \frac{x_1}{1 + 10x_2} \sin(2\pi q x)\right]$$

where:

$$0 \leq x_1, x_2 \leq 1 \quad (5)$$

and $q = 4$, $\alpha = 2$. This problem has its Pareto front disconnected and consistent on 4 Pareto curves.

Figure 1 shows the Pareto fronts produced by the NSGA-II, PAES and MOPSO for the first test function. For this example, the three algorithms performed 4000 evaluations of the fitness function. The average running time of each algorithm (over 20 runs) were the following: 2.402 seconds for the NSGA II, 2.08 seconds for PAES and only 0.076 seconds for our MOPSO. The average values of the metric M_1^* were the following: 0.002536 for the NSGA II (with a standard deviation of 0.000138), 0.002881 for PAES (with a standard deviation of 0.00213), and 0.002057 for MOPSO (with a standard deviation of 0.000286).

4.2 Test Function 2

Our second test function was proposed by Schaffer [11]:

$$\text{Min } f_1(x) = \begin{cases} -x & \text{if } x \leq 1 \\ -2 + x & \text{if } 1 < x \leq 3 \\ 4 - x & \text{if } 3 < x \leq 4 \\ -4 + x & \text{if } x > 4 \end{cases} \quad (6)$$

$$\text{Min } f_2(x) = (x - 5)^2 \quad (7)$$

[3]The source code of the NSGA II and PAES (original versions from their corresponding authors) may be downloaded from the EMOO repository located at `http://www.lania.mx/~ccoello/EMOO/EMOOsoftware.html`

[4]MOPSO will soon become available for download from the EMOO repository previously indicated.

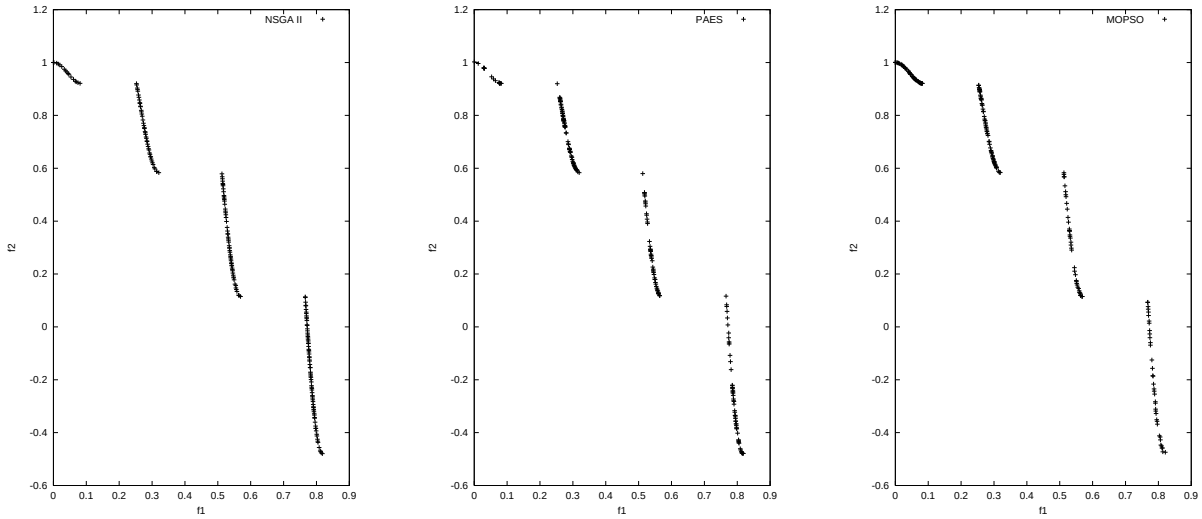

Figure 1: Pareto fronts produced by the NSGA II (left), PAES (middle), and MOPSO (right) for the first test function

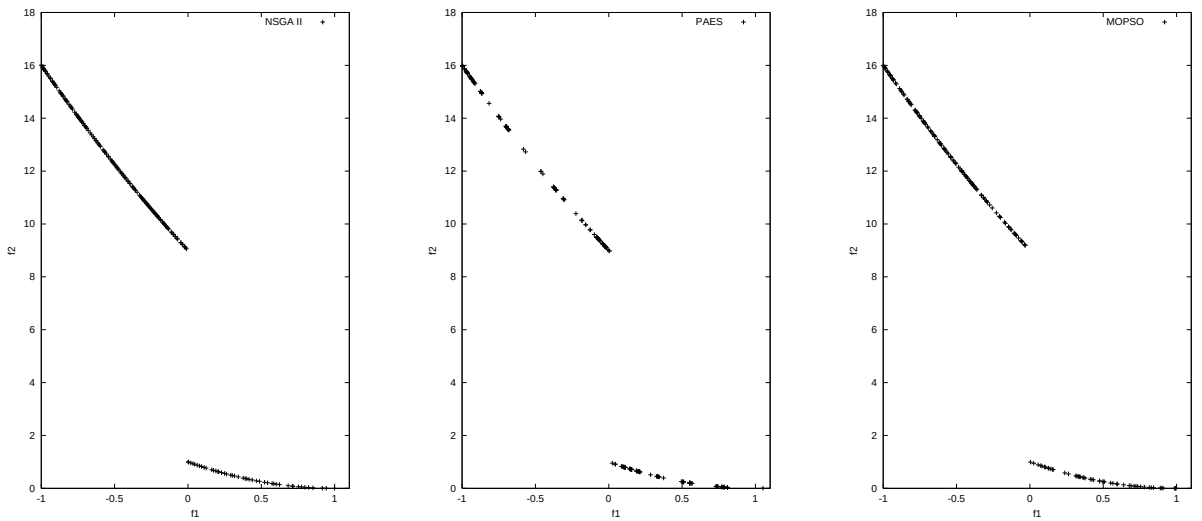

Figure 2: Pareto fronts produced by the NSGA II (left), PAES (middle), and MOPSO (right) for the second test function

where: $-5 \leq x \leq 10$.

This problem also has a disconnected Pareto front. Figure 2 shows the Pareto fronts produced by the NSGA-II, PAES and MOPSO for the second test function. For this example, the three algorithms performed 1200 fitness function evaluations. The average running time of each algorithm (over 20 runs) were the following: 0.812 seconds for the NSGA II, 0.339 seconds for PAES and only 0.046 seconds for our MOPSO. The average values of the metric M_1^* were the following: 0.001594 for the NSGA II (with a standard deviation of 0.000122), 0.070003 for PAES (with a standard deviation of 0.158081), and 0.00147396 for MOPSO (with a standard deviation of 0.00020178).

4.3 Test Function 3

Our third example was proposed by Deb [3]:

$$\text{Minimize } f_1(x_1, x_2) = x_1 \tag{8}$$

$$\text{Minimize } f_2(x_1, x_2) = g(x_1, x_2) \cdot h(x_1, x_2) \tag{9}$$

where:

$$g(x_1, x_2) = 11 + x_2^2 - 10 \cdot \cos(2\pi x_2) \tag{10}$$

$$h(x_1, x_2) = \begin{cases} 1 - \sqrt{\frac{f_1(x_1,x_2}{g(x_1,x_2}} & \text{if } f_1(x_1, x_2) \leq g(x_1, x_2) \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

and $0 \leq x_1 \leq 1$, $-30 \leq x_2 \leq 30$.

This problem has 60 "local" Pareto fronts to which the population could be easily attracted (i.e., the problem is multifrontal [2]).

Figures 3 shows the Pareto fronts produced by the NSGA-II, PAES and MOPSO for the third test function. For this example, the three algorithms performed 3200 fitness function evaluations. The average running time of each algorithm (over 20 runs) were the following: 2.1165 seconds for the NSGA II, 1.641 seconds for PAES and only 0.098 seconds for our MOPSO. The average values of the metric M_1^* were the following: 0.094644 for the NSGA II (with a standard deviation of 0.117608), 0.259664 for PAES (with a standard deviation of 0.573286), and 0.0011611 for MOPSO (with a standard deviation of 0.0007205).

5 Discussion of Results

In the test functions used (including those not included in the paper), MOPSO performed reasonably well[5] in terms of the metric adopted and it required lower computational times than any of the two other algorithms tried. The graphical representation of the results obtained by each method also seem to indicate a good behavior of the proposed algorithm (see Figures 1, 2, and 3).

Finally, we want to mention briefly what are the parameters used by MOPSO and some of the suggested guidelines to fine tune them:

- **Number of particles**: This is equivalent to the population size of a genetic algorithm. We recommend using between 20 and 80 particles.
- **Number of cycles**: This parameter is related to the number of particles. The relationship tends to be inversely proportional (i.e., to larger number of particles, smaller number of cycles and viceversa). We recommend to use between 80 and 120.
- **Number of divisions**: It allows us to determine the number of hypercubes that will be generated in objective function space. We recommend to use between 30 and 50 divisions.
- **Size of the repository**: This parameter is used to delimit the maximum number of nondominated vectors that can be stored in the repository. The value of this parameter will determine the quality of the Pareto front produced.

6 Conclusions and Future Work

It is important to indicate that PSO is an unconstrained search technique. Therefore, it is necessary to develop an additional mechanism to deal with constrained multiobjective optimization problems. We have focused this paper only on the generation of nondominated vectors and in the mechanism to maintain diversity, but we have not dealt with constrained test functions[6]. The extension to handle constrained problems is, however, not too difficult to develop and it is currently under way. A sensitivity analysis is also under way, so that we can determine the role of each of the parameters used in the performance of the algorithm (particularly those controlling the flight direction of the particles). Finally, we are also in the process of validating our algorithm using additional metrics and some other minor refinements to the implementation may take place in the process.

Acknowledgements

The first author gratefully acknowledges support from CONACyT through project 34201-A. The second author acknowledges support from CONACyT through a

[5] MOPSO remained as a competitive algorithm, althought not necessarily better than the other two in all cases.

[6] The approach proposed in [10] uses a sophisticated constraint-handling procedure which seems to be very successful to guide the search in multiobjective problems.

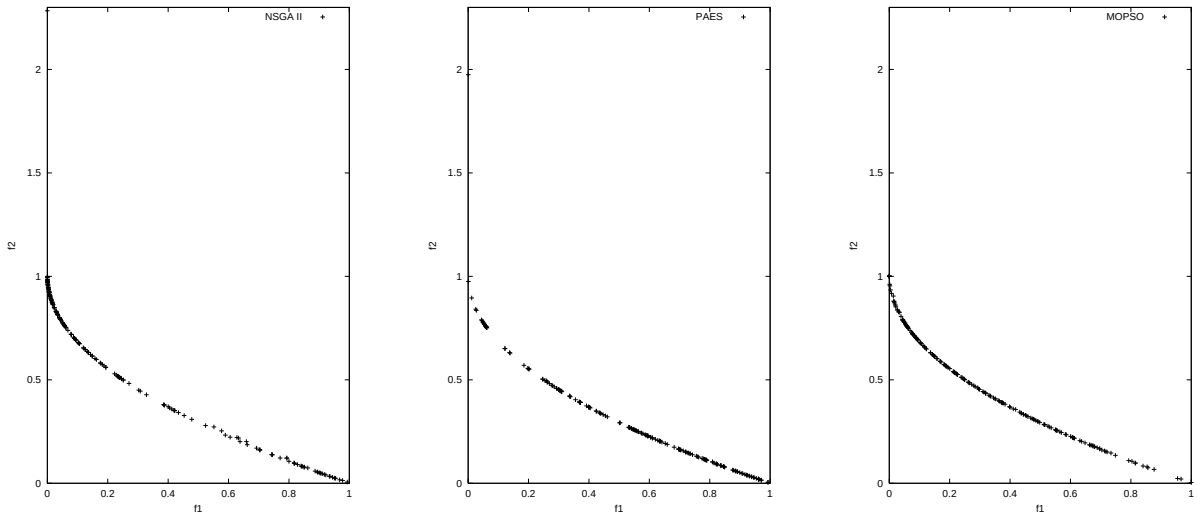

Figure 3: Pareto fronts produced by the NSGA II (left), PAES (middle), and MOPSO (right) for the third test function

scholarship to pursue graduate studies at the Maestría en Inteligencia Artificial of LANIA and the Universidad Veracruzana.

Bibliography

[1] Carlos A. Coello Coello and Gregorio Toscano Pulido. Multiobjective Optimization using a Micro-Genetic Algorithm. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001)*, pages 274–282, San Francisco, California, 2001. Morgan Kaufmann Publishers.

[2] Carlos A. Coello Coello, David A. Van Veldhuizen, and Gary B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, Boston, 2002. ISBN 0-3064-6762-3. (in Press).

[3] Kalyanmoy Deb. Multi-Objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems. *Evolutionary Computation*, 7(3):205–230, Fall 1999.

[4] Kalyanmoy Deb, Samir Agrawal, Amrit Pratab, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. In *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pages 849–858. Springer, 2000.

[5] Kalyanmoy Deb and David E. Goldberg. An Investigation of Niche and Species Formation in Genetic Function Optimization. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 42–50, San Mateo, California, June 1989. George Mason University, Morgan Kaufmann Publishers.

[6] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

[7] James Kennedy and Russell C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, San Francisco, California, 2001.

[8] Joshua D. Knowles and David W. Corne. Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evolutionary Computation*, 8(2):149–172, 2000.

[9] Jacqueline Moore and Richard Chapman. Application of Particle Swarm to Multiobjective Optimization. Department of Computer Science and Software Engineering, Auburn University, 1999.

[10] Tapabrata Ray, Tai Kang, and Seow Kian Chye. Multiobjective Design Optimization by an Evolutionary Algorithm. *Engineering Optimization*, 2002. (In Press).

[11] J. David Schaffer. *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms*. PhD thesis, Vanderbilt University, 1984.

[12] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, 8(2):173–195, Summer 2000.