J. Comp. Int. Sci. (2015) 6(2):61-70
http://epacis.net/jcis/PDF_JCIS/JCIS-0095.pdf
jcis@epacis.net

Pan-american Association of Computational Interdisciplinary Sciences

Particle swarm optimisation algorithms for solving many-objective problems

Mateo Torres [a], Benjamn Barn [a b], Uri Yael[a]
[a] Universidad Catlica "Nuestra Seora de la Asuncin" - UCA
[b] Universidad Nacional de Asuncin - UNA

Received on May 25, 2015 / Accepted on June 03, 2015

Abstract

As the number of conflicting objectives increases, multi-objective optimisation problems (MOPs) become harder to solve; reflecting this difficulty, they even receive the name of *many-objective* problems. Today, these difficult problems are mainly solved with evolutionary algorithms as the Non-dominated Sorting Genetic Algorithm (NSGA). This paper proposes the use of Particle Swarm Optimisation based algorithms for solving many-objective problems presenting evidence regarding the feasibility of the proposal.

Keywords: *Multi-objective optimisation problems, Many-objective optimisation, Particle Swarm Optimisation, Non-dominated Sorting Genetic Algorithm.*

1. Introduction

In multi-objective optimisation problems (MOPs), conflicts among objectives usually prevent from having a single optimal solution but rather a set of trade-off solutions known as Pareto optimal set [1, 2].

In the last years, it has been pointed out that the difficulty of MOPs increases for many-objective problems, i.e. problems having 4 or more objectives [1, 3]. In Pareto-based algorithms, these difficulties are intrinsically related to the fact that as the number of objectives increases, the proportion of non-dominated elements in the populations grows, being increasingly difficult to distinguish among solutions using only the dominance relation [2, 4]. Additionally, several algorithms are based on data structures with complexities that grow exponentially in the number of objectives and other factors [3, 5].

Once an algorithm finds a Pareto set, a decision maker selects a solution from this optimal set. This decision making is beyond the scope of this paper, which is limited to the steps required to obtain a good approximation to the Pareto optimal set. At this stage, visualization of solution alternatives becomes very important. Although several methods have been proposed to this aim [6], there is a lack of a simple and intuitive way to represent solutions in objective space with more than three objectives [1].

As the number of objectives grows, the following phenomena becomes evident:

- Pareto-based algorithms as the well established NSGA are unable to provide the required selection pressure towards better solutions in order to conduct an efficient evolutionary search [1, 2].
- For a given set of solutions, as the number of objectives m increases, the expected growing of the domain space proportion e containing the points that the Pareto dominance classifies as non-comparable is given by the following expression:

$$e = \frac{2^m - 2}{2^m} \tag{1}$$

Note that $\lim_{m\to\infty} e = 1$. This phenomena indicates that Pareto dominance may be inadequate to discriminate among solutions in many-objective problems [1, 4].

In the following sections important concepts for understanding of MOPs are introduced. Also, this work illustrates the feasibility of using MOPSO for solving many-objective problems. Therefore, two multi-objective algorithms are described, first NSGA-II [7] and then, two variants of MOPSO [5, 8]. In Section 5, a complexity analysis on the mentioned algorithms is shown while in Section 6 our experimental tests are described and results are presented. Finally, in Section 7, conclusions and future work are depicted.

2. Multi Objective Optimisation

Problems addressed in this paper are mainly many-objective. Usually, these problems have conflicting objectives and solving them is not a trivial task. Often, when dealing with more than one objective, the need of selecting a solution among a set of non-comparable solutions arises, because a single optimal solution may not exist.

Definition 1 *Multi-objective optimisation problem* [1]: Let F be a set of m objective functions $\{f_1, f_2, \cdots, f_m\}$, $f_i : \mathbb{R}^n \Rightarrow \mathbb{R}$, a MOP is defined as:

$$\begin{aligned} \text{Optimise } \textit{(Maximise/Minimise)}\ & y = F(x) = (f_1(x), f_2(x), \cdots, f_m(x)) \\ & x = (x_1, x_2, \cdots, x_n) \in \mathcal{X} \subseteq \mathbb{R}^n \\ & y = (y_1, y_2, \cdots, y_m) \in \mathcal{Y} \subseteq \mathbb{R}^m \end{aligned} \tag{2}$$

subject to

$$x_i^{(L)} \leq x_i \leq x_i^{(U)}\ \forall\ i \in \{1, 2, \cdots, n\} \tag{3}$$

$$g(x) = (g_1(x), g_2(x), \cdots, g_k(x)) \leq 0 \tag{4}$$

x is a vector of n decision variables, while y represents an m-dimensional objective vector. Constraints (3) represent $2n$ variable bounds that help to define the decision variable space or decision space $\mathcal{X}$. Objective functions constitute a multi-dimensional space called the objective space, termed as $\mathcal{Y}$. Vector g is composed of k constraint functions (4) which shape the feasible region. Solutions that do not satisfy constraint functions and/or variable bounds are called infeasible solutions, while solutions that meet all constraints (3) and (4) are feasible solutions. The set of all feasible solutions $\mathcal{X}_f$ is known as the feasible region. The domain of each f_i is $\mathcal{X}_f$. For each solution $x \in \mathcal{X}_f$ there exists a point y in the objective space. This $\mathcal{X}_f$ defines the feasible objective space $\mathcal{Y}_f$:

$$\mathcal{Y}_f = F(\mathcal{X}_f) = \bigcup_{x \in \mathcal{X}_f} \{F(x)\} \tag{5}$$

As previously stated, problems with $m \geq 4$ are called many-objective; otherwise, they are simply referred to as multi-objective optimisation problems.

In a feasible set of solutions $\mathcal{X}_f$, given 2 solutions $u, v \in \mathcal{X}_f$, It is said that u Pareto dominates, or simply dominates v (denoted as $u \succ v$) if it is not worse in any objective and it is strictly better in at least one objective [1, 2].

Definition 2 *Pareto Optimal set.* For a given MOP the Pareto Optimal set, denoted as $\mathcal{P}^*$, is defined as the set of non-dominated feasible solutions, i.e.:

$$\mathcal{P}^* = \{x \in \mathcal{X}_f | \nexists\ x' \in \mathcal{X}_f \text{ such as } x' \succ x\}$$

Definition 3 *Pareto Optimal Front.* For a given MOP the Pareto Optimal Front, denoted as $\mathcal{PF}^*$, is defined as the image in objective space of the Pareto set $\mathcal{P}^*$, i.e.:

$$\mathcal{PF}^* = \{y = F(x) \in \mathcal{Y}_f | x \in \mathcal{P}^*\}$$

Algorithm 1: Fast Non-dominated Sort, responsible of the $m(2N)^2$ complexity of the algorithm.

```
input : P
foreach p ∈ P do
    S_p = ∅; n_p = 0;
    foreach q ∈ P do
        if p ≻ q then
            Add q to set S_p of solutions dominated by p;
        else if q ≻ p then n_p = n_p + 1
    if n_p = 0 then p_rank = 1; F_1 = F_1 ∪ {p}
i = 1;
while F_i ≠ ∅ do
    Q = ∅                                   // Used to store members of the next front;
    foreach p ∈ F_i do
        foreach p ∈ S_p do
            n_q = n_q − 1;
            if n_q = 0 then q_rank = i + 1; Q = Q ∪ {q}
    i = i + 1; F_i = Q;
```

3. NSGA-II

The reference algorithm in current literature for solving MOPs with evolutionary algorithms is undoubtedly the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [1, 7]. This algorithm depicts a fast non-dominated sort procedure to classify individuals in a set of candidate solutions called population in several non-dominated fronts (Algorithm 1).

Additionally, NSGA-II presents a crowding distance value d_i to preserve diversity. It implements a crowded-comparison approach using another method called *crowding distance assignment*. It depends in a new operator called *crowded-comparison operator* ($\prec_n$) which defines a partial order as follows:

$$i \prec_n j \iff (\mathcal{F}_i < \mathcal{F}_j) \vee (\mathcal{F}_i = \mathcal{F}_j \wedge d_i > d_j) \tag{6}$$

where $\mathcal{F}_i$ is the front containing solution i [7].

In evolutionary algorithms as NSGA-II, a set of candidate solutions is known as *population*. In the main loop of NSGA-II, a parent population P_t and another offspring population are combined into a set R_t and then separated in several non-dominated fronts $\mathcal{F}_i$. Afterwards, every generated front is assigned a crowding distance until the new population P_{t+1} grows to the population size limit N. Eventually, elements of the next unused front are sorted to complete the population. Finally, a new offspring population Q_{t+1} is generated [7].

Regarding complexity, according to the analysis given in by Deb et al. [7], the complexity of the operations in one iteration are:

1. *non-dominated sorting*: time complexity on the order of $m(2N)^2$ because of the two nested loops used to divide the population in non-dominated fronts, and the comparison of solutions (Algorithm 1).

2. *crowding distance assignment*: since it has to sort m times, and the complexity is given by the sorting algorithm, the overall time complexity is on the order of $m(2N)\log(2N)$.

3. *sorting on* $\prec_n$: time complexity on the order of $2N\log(2N)$. Each sort produced in a front is given by the sorting algorithm.

The overall complexity of the algorithm is therefore on the order of $m(2N)^2$, which is governed by the non-dominating sorting part of the algorithm. It is shown in Sections 5 and 6 that iterating over a population of $2N$ has considerable impact on execution time.

4. MOPSO

4.1. Particle Swarm Optimisation

The *Particle Swarm Optimisation* meta-heuristic (PSO), developed by Kennedy and Eberhart in 1995 [9], is inspired in social behaviour of individuals like bees and birds, which perform a collective zone exploration looking for food or guiding the swarm or flock to desirable locations. PSO basically consists in an iterative algorithm having a group of individuals called *swarm*, in which each individual called *particle* explores the decision space looking for optimal solutions.

In a n-dimensional decision space, each particle i in the swarm knows its current position $x_{i=}[x_{i_1}, x_{i_2}, \cdots, x_{i_n}]$, its current velocity $v_i = [v_{i_1}, v_{i_2}, \cdots, v_{i_n}]$ which led it to its current position, and the best position it has been $p_i = [p_{i_1}, p_{i_2}, \cdots, p_{i_n}]$ called *best personal position*. Additionally, all other particles are aware of the best position found by the entire swarm $g = [g_1, g_2, \cdots, g_n]$ called *global best position*.

At every iteration t, every component j of the position and the velocity is updated for each particle i as follows:

$$v_{ij}^{t+1} = \omega \times v_{ij}^t + C_1 \times \text{rand}() \times (p_{ij}^t - xv_{ij}^t) + C_2 \times \text{rand}() \times (g_{ij}^t - x_{ij}^t) \tag{7}$$

$$x_{ij}^{t+1} = x_{ij}^t + v_{ij}^{t+1} \tag{8}$$

where ω is the *inertia* parameter, C_1 is called the *cognitive* parameter, C_2 is the *social* parameter and *rand()* is a procedure that returns a random value in $[0, 1]$.

Equation (7) is used to update the velocity of the i-th particle from its current velocity, the euclidean distance to its best personal position, and the euclidean distance to the best global position.

In (8), position vector components belonging to the i-th particle are updated using the updated velocity. The use of distances in (7) is to achieve an "attraction" for the particle towards both its personal best position and the global best position. This distances are called *cognitive factor* and *social factor* respectively. Parameters C_1 and C_2 are constants that determine how accelerated is the particle towards the mentioned factors [9], defining the maximum influence of these factors in (7). In a similar fashion, `rand()` is used to define the real influence of such factors. The final outcome is the preservation of diversity.

The inertia parameter ω introduced by Eberhart and Shi [10] is used to command the impact of the previous velocities in the calculations for the next value. ω manages the balance between the global and personal decision space exploration [11]. Big values of ω lean toward exploring new areas (*exploration*), but small values are in favour of searching areas close to the current position (*exploitation*). It is usual to limit the growth of the velocity by defining a maximum value $v_{\max}$ [11]. After calculating the new velocity, it is moderated:

$$\begin{aligned} &\textbf{if } v_{ij} > v_{\max} \textbf{ then } v_{ij} = v_{\max} \\ &\textbf{else if } v_{ij} < -v_{\max} \textbf{ then } v_{ij} = -v_{\max} \end{aligned} \tag{9}$$

Finally, if the position reaches the limit of the feasible decision space $\mathcal{X}_f$, it is normally "bounced" in the opposite direction on the component that reached the limit. This ensures that every solution found in the iteration process will be inside the feasible set of solutions.

4.2. Particle Swarm Optimisation in Multi-Objective Problems

Adapting the PSO algorithm to optimize a multi-objective problem requires a modification of the concept of better personal position and best global position. Since having conflicting objectives result in non-comparable solutions, now the algorithm must handle a best personal *repository* for every particle and a global best *repository* as well. An adaptation to the PSO algorithm to solve MOPs is known as *Multi Objective Particle Swarm Optimisation* (MOPSO) [12]. In this Section two variants of MOPSO are briefly presented.

4.2.1 Moore and Chapman (1999)

Moore and Chapman [5] introduced a method in which every particle stores a best personal repository for every non-dominated positions it has visited. This version is simply referred to as MOPSO in this work. A global best repository is maintained as well, storing the non-dominated positions visited by the entire swarm. To calculate a new velocity, every particle selects a random element from its personal repository to use in (7) as best personal position. Similarly, a random element of the global repository is selected to use as global best position.

As complexity is concerned, it is clear that, for every iteration, going through every component on the decision space to calculate in the first loop drives on the order of N^2 complexity. Also, comparing non-domination in the best particle selection is on the order of M. Therefore, the overall complexity is MN^2. Note that spatial complexity (or memory needs) increases due to storing a personal repository for each particle resulting in an overall spatial complexity on the order of $nN(N+1)$ (Algorithm 2).

Algorithm 2: Moore y Chapman's MOPSO with complexity on the order of mN^2.

```
input : ω, C1, C2, vmax, N
for i = 1 to N do
    Initialize the i-th particle with a random velocity and position;
    eval(i);
    Update the personal repository if the i-th particle (Pi);
    Update the global repository (G);
while not stop_condition() do
    for i = 1 to N do
        Select best personal position (random element in Pi);
        Select best global position (random element in G);
        Calculate new velocity for the i-th particle (eq. 7);
        Limit velocity components to vmax (eq. 9);
        Calculate a new position for the i-th particle (eq. 8);
        Update position to constraints if necessary;
    for i = 1 to N do
        eval(i);
        Update the personal repository if the i-th particle (Pi);
        Update the global repository (G);
Return non_dominated_solutions(G);
```

4.2.2 Coello and Lechuga (2002)

Coello and Lechuga [8] presented a version of MOPSO in which a fixed size repository stores every non-dominated solution found by the swarm in the exploration process. That repository employs an *adaptive mesh* scheme in which the explored space is represented in bounded regions called *hypercubes*. In this work, this version is referred to as MOPSO_CL.

Each hypercube is given a score equal to dividing a constant number like 10 by the total number of non-dominated solutions contained by it [12]. Such score is later used to select the best global position from the repository, which is updated in every iteration, inserting every non-dominated solution found by the swarm and removing the ones that prove to be dominated by the new solution. If the repository is full while inserting a new solution, a solution is removed from the hypercube containing the largest number of solutions. In Algorithm 3 the procedure for assigning solutions to hypercubes is shown.

Algorithm 3: Assign Solutions to Hypercubes using MOPSO_CL, with complexity on the order of ${n_{\text{div}}}^m$.

```
input : G, ndiv, m
qtyHypercubes = ndiv^M;
H = initialize_hypercubes(qtyHypercubes);
for i = 1 to m do
    max_i = 1 get_max( G_i );
    min_i = 1 get_min( G_i );
for i = 1 to m do
    amp_i = (max_i − min_i) / (ndiv − 1) ;
    ini_i = (min_i − amp_i) / 2 ;
for i = 1 to |G| do
    for i = 1 to |G| do
        j = Calculate index for particle i;
    insert_particle_in_hypercube(H[j],G[i]);
for i = 1 to qtyHypercubes do update_calif(H[i])
```

Regarding the complexity of this version, it is clear from the last loop that it is on the order of ${n_{\text{div}}}^m$.

5. Complexity Analysis

The complexity analysis was performed taking into account the original definition of the algorithms [5, 7, 8]. In what follows, a summary based on algorithms presented in Sections 3 and 4 are illustrated. These results are completely congruent with the ones proposed by the authors:

- NSGA-II: On the order of $m(2N)^2$, as stated in Section 3.
- MOPSO: On the order of mN^2, as stated in Section 4.2.1.
- MOPSO_CL: On the order of ${n_{\text{div}}}^m$, as stated in Section 4.2.2.

Considering that this work is mainly interested in many-objective problems (m may be very large), experiments with MOPSO_CL are not presented due to its exponential growth in m. However, it is worth noting that this complexity is given by the hypercube administration (see Algorithm 3). Therefore, an adaptation of MOPSO_CL is suggested to take advantage of a sparse matrix representation for many-objective problems.

Let the relation $\mathcal{U}$ be:

$$\mathcal{U} = \frac{\texttt{complexity(NSGA)}}{\texttt{complexity(MOPSO)}} = \frac{m(2N)^2}{mN^2} = 4 \tag{10}$$

It is clearly expected that MOPSO's execution time be around 4 times faster than NSGA-II's; therefore, it may be a good idea to use MOPSO to solve many-objective problems. To validate this statement, in the following section our experimental tests are shown.

6. Experimental Results

To validate the complexity analysis presented in Section 5, the following experiments was performed: varying m (dimension of objective space), n (dimension of decision space), N (population size limit) and E (evaluation limit), execution time is compared.

Being t_{MOPSO} and t_{NSGA} execution times for NSGA-II and MOPSO respectively, the improvement ratio τ is calculated as follows:

$$\tau = \frac{t_{\text{NSGA}}}{t_{\text{MOPSO}}} \tag{11}$$

This clearly means that, for a given configuration, if $\tau > 1$, MOPSO is faster than NSGA-II.

6.1. Environment

3.000 parameter combinations where executed: $N \in \{100, 200, \cdots, 1000\}$ for population size limit, $m \in \{4, 6, 8, 10, 12\}$ for number of objectives, $n \in \{14, 15, 16, 17, 18, 19\}$ for decision space dimension, and finally $E \in \{1000, 2000, \cdots, 10000\}$ for number of executions of the objective functions, used as stop criteria.

Both algorithms were used to solve the same problem: DTLZ1 [13]. Once these executions where finished, correlations for every parameter of the mentioned variables and τ were calculated.

6.3. Execution Times

Table 1: Average execution times with 3.000 executions.

algorithms	**Execution time (seconds)**
NSGA-II	35.087
MOPSO	3.063
τ	11.455

As expected, $\tau > 1$ in every performed test, proving the feasibility of using MOPSO with many-objective problems.

The hardware used in this experiments is an Intel® Core™i7 CPU 970 @ 3.20 GHz with 12 GB RAM and running Fedora 20.

In the rest of this section, correlations for each variable are presented to show the impact of each variable in execution times.

6.3. Correlations

The influence that each variable has over execution times are studied by analyzing the correlations between τ and each variable. Correlations are shown in Table 2.

Table 2: Experimental Correlations

Variable	**Correlation**	**Comment**	**favoured algorithm**
$\rho_{\tau,N}$	0.796	Strong influence of N	MOPSO
$\rho_{\tau,m}$	0.257	Weak influence of m	MOPSO
$\rho_{\tau,n}$	-0.912	Strong influence of n	NSGA-II
$\rho_{\tau,E}$	0.787	Strong influence of E	MOPSO

From the calculations, it is clear that increasing N, m and E lean towards using MOPSO for solving many-objective problems. However, when increasing n NSGA-II still looks as a right choice, although it is worth noticing that large values of n naturally increases E.

7. Conclusions and Future work

With the presented evidence, it is clear that MOPSO is a feasible alternative for solving many-objective problems. Given the considerable difference in execution times in the studied algorithms, it does not seem reasonable to limit the number of executions of the objective function as stop condition.

As future work, authors are working on performing quality measurements (not presented in this work); analysing the complexity of other algorithms; and correlating other variables' impact in quality of the generated approximation solution set.

References

[1] C. von Lcken, B. Barn, and C. Brizuela, "A survey on multi-objective evolutionary algorithms for many-objective problems," *Computational Optimization and Applications*, pp. 1–50, 2014.

[2] K. Deb and D. Kalyanmoy, *Multi-Objective Optimization Using Evolutionary Algorithms*. New York, NY, USA: John Wiley & Sons, Inc., 2001.

[3] D. W. Corne, J. D. Knowles, and M. J. Oates, "The pareto envelope-based selection algorithm for multiobjective optimization," in *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pp. 839–848, Springer, 2000.

[4] M. Farina and P. Amato, "On the optimal solution definition for many-criteria optimization problems," in *Fuzzy Information Processing Society, 2002. Proceedings. NAFIPS. 2002 Annual Meeting of the North American*, pp. 233–238, 2002.

[5] J. Moore and R. Chapman, "Application of particle swarm to multiobjective optimization," *Department of Computer Science and Software Engineering, Auburn University*, 1999.

[6] D. Walker, R. Everson, and J. Fieldsend, "Visualisation and ordering of many-objective populations," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pp. 1–8, July 2010.

[7] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *Evolutionary Computation, IEEE Transactions on*, vol. 6, pp. 182–197, Apr 2002.

[8] C. A. Coello Coello and M. Lechuga, "Mopso: a proposal for multiple objective particle swarm optimization," in *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, vol. 2, pp. 1051–1056, 2002.

[9] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Neural Networks, 1995. Proceedings., IEEE International Conference on*, vol. 4, pp. 1942–1948 vol.4, Nov 1995.

[10] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pp. 69–73, May 1998.

[11] Y. Shi and R. Eberhart, "Parameter selection in particle swarm optimization," in *Evolutionary Programming VII* (V. Porto, N. Saravanan, D. Waagen, and A. Eiben, eds.), vol. 1447 of *Lecture Notes in Computer Science*, pp. 591–600, Springer Berlin Heidelberg, 1998.

[12] J. Lima and B. Barán, "Optimización de enjambre de partículas aplicada al problema del cajero viajante bi–objetivo," *Revista Iberoamericana de Inteligencia Artificial*, 2006.

[13] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, "Scalable Multi-Objective Optimization Test Problems," in *Congress on Evolutionary Computation (CEC 2002)*, pp. 825–830, IEEE Press, 2002.