An Adaptation of the Parliamentary Metaheuristic for Permutation Constraint Satisfaction

Luis de-Marcos, Antonio García, Eva García, José J. Martínez, José A. Gutiérrez, Roberto Barchino, Jose M. Gutiérrez, José R. Hilera, Salvador Otón

***Abstract*—Inspired by political parties' behavior in parliament's elections of chairman, Parliameantary Optimization Algorithm (POA) has emerged as a new stochastic population-based optimizer. Current research has proven POA efficiency in numerical optimization but it is difficult to find a POA version that deals with combinatorial optimization. In this paper we present a parliamentary algorithm that can solve permutation constraint satisfaction problems along with the results of its experimental testing and comparison with other evolutionary methods. Results demonstrate POA efficiency in this new landscape.**

I. Introduction

PARLIAMENTARY optimization algorithm (POA) [1, 2] is a novel stochastic meta-heuristic inspired in the behavior observed in political parties when trying to gain control over parliaments in head elections. A randomly initialized set of individuals is partitioned in groups and the most fitted individuals of each group are designated as candidates. Then groups engage in an iterative process involving intra-group and inter-group fitness-based competitions. During the former, candidates bias regular members, compelling them to evaluate (move to) new positions closer to candidates' current positions; and thus, exploring the most promising areas of the search space. As for the latter stage, groups stochastically form alliances to merge, but they also compete; less powerful (fitted) groups can be removed. Computation stops when a predefined termination criterion is met, and the best candidate of all groups (best solution) found is returned. POA is, then, a population-based optimizer that evolves a solution over an iterative process. And although, strictly speaking, POA cannot be considered as an evolutionary method, it bears resemblance to some of those methods; and that made, in our opinion, comparisons with them seem quite natural.

POA has been, almost exclusively, used for numerical optimization; where it has demonstrated to be competitive or even outperform other, well known and well studied, stochastic approaches like Genetic Algorithms (GA) [1] and Particle Swarm Optimization (PSO) [2]. Performance improvement over a set of standard test-bench functions can be observed in terms of the quality of the final solution as well as in terms of the number of iterations (calls to the fitness function) required to get such an accurate solution. Although numerical optimization is an always interesting and challenging field with many potential applications, combinatorial optimization is another demanding area with not less importance and possible applications that also requires attention [3].

This paper introduces a POA version designed to deal with permutations and constraint satisfaction problems. Section II describes the algorithm adaptation to the new search space. Section III presents the experiments that were carried out to fine-tune the new optimizer and to test its performance. Section IV presents a comparative analysis of POA's performance with two other evolutionary approaches (PSO and GA). Section V offers further insight in the epistemological foundations of POA and tries to frame it in relation with the evolutionary computation paradigm. And finally, section VI presents conclusions and future work.

II. A Parliamentary Algorithm for Permutation Constraint Optimization

A. Permutation Constraint Satisfaction

Tsang [4] defines a permutation constraint satisfaction problem (permutCSP) as a quadruple *(X,D,C,P)* where $X = \{x_0, x_1, \dots, x_{n-1}\}$ is finite set of variables, D is a function that maps each variable to its corresponding domain *D(X)*, $C_{ij} \subset D_i \times D_j$ is a set of constraints for each pair of values *(i, j)* with $0 \leq i < j < n$, and $P=<v_0, v_1, \dots, v_{n-1}>$ is a tuple of $|X|=n$ values. A solution S of a permutCSP is a complete permutation of P in which all variables x_i in X are assigned a value from its domain D, in such a way that all constraints are satisfied simultaneously. A constraint is satisfied when $(x_i, x_j) \in C_{i,j}$, and (x_i, x_j) it is said to be a valid assignment. If $(x_i, x_j) \notin C_{i,j}$ then the assignment (x_i, x_j) violates the constraint.

To provide an example, consider the problem of ordering five tasks named 1,2,3,4 and 5; the permutCSP whose only solution is the set $S = \{1,2,3,4,5\}$ (all tasks must be ordered) can be defined as:

$$X = \{x_1, x_2, x_3, x_4, x_5\}$$
$$D(X_i) = \{1,2,3,4,5\}\ \forall x_i \in X$$
$$C = \{x_{i+1} - x_i > 0 : x_i \in X, i \in \{1,2,3,4\}\}$$
$$P = < 1,2,3,4,5 >$$

This is an extremely simple example but it should be noted that many popular problems can be modeled according

Manuscript received February 5, 2010.

Luis de-Marcos is with the Computer Science Department, University of Alcalá, Spain (+34918856656; luis.demarcos@uah.es)

Antonio García, Eva García, José J. Martínez, José A. Gutiérrez, Roberto Barchino, Jose M. Gutiérrez, José R. Hilera and Salvador Otón are with the Computer Science Department, University of Alcalá. Spain.

to this definition. Examples include the N-queens problem, the graph coloring problem, the scene labeling problem, temporal reasoning, planning and scheduling, and graph matching. Some of them are known by their complexity, being NP-complete problems, and all of them have important practical applications.

For heuristic and meta-heuristic methods it is a requirement to have a fitness function that represents the goodness of a solution [5]. When dealing with CSP problems and when the problem domain does not provide any predefined function, a common choice is a standard penalty function [6]:

$$f(X) = \sum_{0 \le i < j < n} \mathrm{V_{i,j}}(\mathrm{x_i}, \mathrm{x_j}) \qquad (1)$$

where $\mathrm{V_{i,j}}: \mathrm{D_i} \times \mathrm{D_j} \rightarrow \{0,1\}$ is the violation function

$$\mathrm{V_{i,j}}(\mathrm{x_i}, \mathrm{x_j}) = \begin{cases} 0 \text{ if } (x_i, x_j) \in C_{i,j} \\ 1 \text{ otherwise} \end{cases} \qquad (2)$$

This fitness function works well if the constraint set *C* for the CSP has been accurately defined. In the previous example, the restriction set was defined as $C = \{x_{i+1} - x_i > 0 : x_i \in X, i \in \{1,2,3,4\}\}$. A more accurate definition will be $C = \{x_i - x_j > 0 : x_i \in X,\ x_j \in \{x_1, \dots, x_i\}\}$. Considering the sequence {2,3,4,5,1}, the standard penalty function will return 1 if the first definition of *C* is used, while the returned value will be 4 if it is used the second definition. The second definition is more accurate because it returns a better representation of the number of swaps required to turn the sequence into the valid solution. Moreover, the first definition of *C* has additional disadvantages because some really different sequences (in terms of its distance to the solution) return the same fitness value. For example, sequences {2,3,4,5,1}, {1,3,4,5,2}, {1,2,4,5,3} and {1,2,3,5,4} will return a fitness value of 1. Fortunately, problem of finding the most accurate set of restrictions could be solved programmatically. A function that recursively process all restrictions and calculates the most precise set of restrictions violated by a given sequence can be developed and called over the input sequence. In that way, the user will usually define the minimum necessary number of constraints and an algorithm will compute the most accurate set in order to facilitate heuristic's convergence.

B. Parliamentary Optimization

POA's original description is presented in code listing 1. In what follows we describe the most important details concerning its implementation.

1) Initialization

Initially, a $N \times L$ set of randomly initialized individuals is partitioned in N groups of L members. Best θ of each group are designated as candidates. All other members are considered regular members. N, L and θ are input parameters.

2) Intra-group competition

After initialization, groups engage in an inter-group competition. Regular members are biased towards candidates. Members' new position in the search space is calculated as in equation 3:

$$p' = p_0 + \eta\left(\frac{\sum_{i=1}^{\theta}(p_i - p_0) \cdot f(p_i)}{\sum_{i=1}^{\theta}(p_i)}\right) \qquad (3)$$

In the above formula, p' is the new position, p_0 is the current position, and η is a random number between 0.5 and 2 that gives a stochastic component to members' movement. Members update their position only if the new position has a better fitness value. After that, candidates are reassigned because regular members might have a better fitness. Please note that candidates are not updated.

Finally, groups compute their power as in:

$$power^i = \frac{m \cdot Avg(Q_i) + n \cdot Avg(R_i)}{m + n} \qquad (4)$$

where $\mathrm{Q_i}$ and $\mathrm{R_i}$ are the vectors of fitness of candidates and regular members of the group *i* respectively. And *m* and *n* are weighting constants which are input parameters.

```
Initialize population
  Partition of the population in N groups of L
members
  Pick θ most fitted individuals as candidates
for each group
repeat
  Intra-group competition
    Bias regular members towards candidates
    Reassign candidates
    Compute power for each group
  Inter-group competition
    Merge λ strongest groups with probability pm
    Remove γ weakest groups with probability pd
until(stopping condition met)
return the best candidate
```

Code. 1. Parliamentary Optimization Algorithm (from [2])

3) Inter-group competition

Inter-group competition starts just after intra-group competition terminates. The idea is to model the collaborative and competitive behavior in which political parties engage to gain control over the parliament; for example, to get its best candidate elected as chairman. In real parliaments this usually comprises to create alliances to form more powerful groups. Groups estimate their power based on the quality of their candidates, but the number of members is also an important factor. When they have estimated their power, groups start communication processes in order to collaborate and to achieve common goals. This usually means to gain more power in relation to opposing groups. It is an evolutive process in which more fitted groups gain more and more control and less powerful groups can also eventually collapse and disappear. The real operation of a parliament is extremely complex to describe so it is difficult to model it accurately.

Original POA employs a simple but practical approach to simulate inter-group parliamentary processes. λ more powerful groups can merge into a single group with probability $\mathrm{p_m}$, and γ less powerful groups can disappear with probability $\mathrm{p_d}$. λ, γ, $\mathrm{p_m}$ and $\mathrm{p_d}$ are input parameters. Candidates must be reassigned before merging.

C. POA for permutation constraint satisfaction

Original POA is designed for numerical optimization and thus works in a continuous space. In order to apply POA to solve permutCSPs some of their internal workings have to be redefined so that it can operate in a discrete space in which individuals are also permutation sets. In what follows we describe POA adaptation to solve permutCSPs.

1) Redefining biasing

It can be easily inferred that initialization and inter-group competition do not require any special modification to work in this new landscape. Problems arise in intra-group competition and they are related with regular members biasing towards candidates. A linear fitness-proportional weighting towards all candidates, as in equation 3, is no longer possible to move regular members. We propose to use a fitness proportional selection to stochastically permute each position in the tuple towards one candidate or, alternately, keep its current value. Fitness proportional selection was originally introduced for genetic algorithms [7] and can be employed in the different selection processes: parent selection and survivor selection. Several important drawbacks have been explained for this method, including premature convergence in unbalanced population distributions [8]; so alternative methods like ranking selection and stochastic universal sampling (SUS) [9] have been introduced. Quick convergence can be a more desirable feature in this new definition of POA because selection will be performed for each dimension of the solution, as it is going to be explained in what follows.

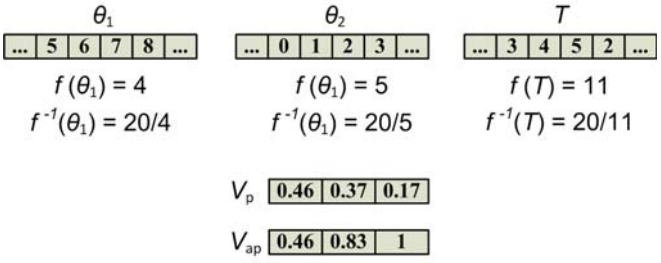
Let T be a candidate solution tuple, the probability for each position, t_j T, to move towards a candidate c θ can be calculated according to equation 5. All candidates and also the current individual form the pool of elements which are considered for the selection. Probabilities are computed once and then random numbers are generated for each dimension to determine the bias. If the movement of any position is finally required, the position will be set to the value of the same position in the candidate selected by swapping values. A similar approach has been described and successfully employed for PSO [10].

$$t_j^{swap \to c} = \frac{f(c)}{\sum_{i=0}^{\theta} f(p_i) + f(T)} \qquad (5)$$

Considering that a permutCSP with the fitness function previously introduced is a minimization problem, it is necessary to use an inverse weighting function in order to assign larger probabilities to individuals with lower (better) fitness. This inverse fitness function was calculated as in equation 6. Then equation 5 will return a fitness proportional probability if every call to f is substituted for a call to f^{-1}.

$$f^{-1}(X) = \frac{\sum_{i=0}^{\theta} f(p_i) + f(X)}{f(X)} \qquad (6)$$

To facilitate implementation, probabilities were stored in an intermediate vector (V_p) and another vector with aggregated probabilities (V_{ap}) was also created to evaluate random numbers. The whole process is exemplified in figure 1 which presents a part of the biasing process in a hypothetical job scheduling problem. Two candidates (θ_1 and θ_2) are considered and the current individual T is also displayed. The numbers on every individual represent the name of the task. Sample fitness values are considered to calculate inverse fitness values and both probability vectors. The final biasing is exemplified in the table showed in the lower part of the figure. Three random numbers are drawn to decide the swap for three positions, the candidate selected and the resulting state of the individual T are displayed.



Position	Random number	Biasing towards	T
T_i	0.1	θ_1 ... 5 6 7 8 ...	... 5 4 3 2 ...
T_{i+1}	0.9	none	... 5 4 3 2 ...
T_{i+2}	0.7	θ_2 ... 0 1 2 3 ...	... 5 4 2 3 ...

Fig. 1. An example of member biasing for permutation problems.

Please note that in this discrete version regular members always update their position. In original POA, positions are only updated if a better fitness is found. Our aim was to improve individuals' mobility in the new landscape. It shall be noted that many discrete spaces may contain plateaus: large areas in which all solutions contain values with similar fitness values. If regular members are compelled to move in any case then wider search areas will be explored at no especial higher expense (new positions are always evaluated). It shall be also noted that the random bias parameter (η) is not longer required. The stochastic component that this parameter originally added is replaced in our bias redefinition by the probabilistic fitness-based selection.

2) Considering mutation

Mutation was introduced in evolutionary algorithms since its very beginning [11, 12] as a way to model a similar process that occurs in nature. The underlying idea is to introduce a mechanism to increase population diversity by randomly altering an individual gene feature that can make it uniquely different from each of its antecessors. Mutation has proven to be so important that a set of evolutionary algorithms rely heavily or almost exclusively in mutation procedures to evolve solutions. Evolution strategies [13] are the most known example.

A simple mutation mechanism was introduced in our version of POA as a mean to increase population diversity. Just before regular members, biasing swap mutation is implemented. Mutation rate for each individual is

implemented as an input parameter (*p*). Individual-level mutation (and not gene-level mutation) is used.

3) Duplication elimination policy

To avoid genetic drift (quick convergence to the same or very similar individual for all the population), which was observed in the initial stages of development, a duplicate elimination policy was introduced. Just after biasing and mutating, each individual is compared with the previous elements in its group. If it is equal to any of these them, a swap mutation is enforced until it differs. That implementation can have huge computational costs when POA faces huge populated groups and it will also hinder any future distributed POA approach because full information about the group is required to implement it. Thus, a new boolean parameter was introduced to enable or disable it.

Code fragment 2 presents the final version of POA to deal with permutation problems. Differences with the original implementation are showed in bold.

```
Initialize population
  Partition of the population in N groups of L
members
  Pick θ most fitted individuals as candidates
for each group
repeat
  Intra-group competition
    Bias regular members towards candidates
    Mutate members with probability p
    if de is true Eliminate duplicates
    Reassign candidates
    Compute power for each group
  Inter-group competition
    Merge λ strongest groups with probability pm
    Remove γ weakest groups with probability pd
until(stopping condition met)
return the best candidate
```

Code. 2. POA for combinatorial optimization (permut-POA)

III. Experimentation & Tests

The permut-POA was implemented using the object oriented paradigm in C#. The next natural step comprised its tuning and test. This section first presents the selection and design of test cases that were used to experiment with permut-POA and to fine tune permut-POA parameters. This will lay the ground to present the results.

A. Test Cases

To test the performance of this novel POA version, a set of 100 random test cases was used. Among the different problems common in literature, we decided to choose a generic task scheduling problem. Scheduling has many important practical applications along with a large set of backing literature [3]. There are also other problems, like the n-Queens, travel-salesman and knapsack, which are not less important. These are known and widely studied for their complexity (all of them are NP-complete), but we wanted to select a set of test cases that were not necessarily so difficult to solve and that, then, could be found in most common circumstances. A generic task scheduling problem simply comprises: (1) a set of tasks which must be performed; and (2), a set of constraints that state conditions in relation to tasks arrangement in the form of: task X cannot be started if task Y has not been completed previously. Each test case comprised 25 tasks and 50 constraints. The number of tasks was mainly an arbitrary choice based on experimental tests that determined a problem size that can be solved fast (usually less than 0.5 seconds) and thus susceptible to be repeated a huge number of times to gain statistical significance. As for the number of constraints our choice was based on works related with phase transition, mainly [14], which study the point in which randomly generated binary constraint satisfaction problems turn from being soluble to being insoluble. Constraints were randomly generated ensuring that each test case has at least one solution. To do that, the dependency graph of each test case was generated to check for any possible cycle. Cyclic dependencies meant that the problem had no solution. In that case the problem was discarded generating a new one. 100 different test cases were randomly generated using that method and stored in order to be used for experimental testing.

Each test case may have many potential solutions and the algorithm computation finished when one was found. It is difficult to calculate the exact number of feasible solutions for every problem but it is possible to make some estimations of the relation among feasible solutions and total solutions. We estimate that the order of this relation is in the range of 10^{14} as a mean for all test cases. A random generated constraint between tasks *A* and *B* in an initially unconstrained task schedule problem will reduce the number of feasible solutions by ½ because all the sequences in which *B* precedes *A* will become non-feasible solutions. An initial analysis of a small sample of the generated test cases showed that a certain number constraints introduced such reduction in the number of feasible solutions, while the rest of constraints have no effect because they just introduce redundancy. We noticed that approximately 35 constraints reduce the solution space while the others have no effect; then it is easy to deduce the aforementioned number considering that the solution space size is approximately $1.5x10^{25}$. This is, in our opinion, a quite challenging problem but we also took into consideration that few may argue that it is not so, or even argue that it is a toy problem. To try to mitigate these possible critics and to test permut-POA scalability additional test cases with 40, 50, 60, 75 and 100 tasks with just one feasible solution were designed. These problems were easy to solve using exact deterministic methods but they can be quite challenging for permutation stochastic population-based methods.

B. Parameter Tuning

POA has many parameters and there are no studies, as far as authors are concerned, about them. Original work on POA offers a set of values for parameters without any rationale concerning their selection. This is, in our opinion, a serious drawback for POA. Practitioners that have to select a technique for a particular combinatorial problem will almost

surely choose one in which a substantial work about parameter control (including recommended practices) exist. To try to mitigate that problem, and always focusing on a practitioner's stance, we have conducted a preliminary study to try to set best practices for parameter selection in permut-POA.

Eiben & Smith [7] classification of parameter control techniques (considering how parameters change during time) comprises three categories: Deterministic parameter control, adaptive parameter control and self-adaptive parameter control. The deterministic approach is the easiest way and it is based on a deterministic rule that sets parameters in a fixed and predefined way. If parameters are set before running the algorithm then the process is also known as parameter tuning. Adaptive control techniques consider feedback from the search. And the self-adaptive approach also considers feedback but this time parameters are evolved along with the solution. Parameter tuning may not be the best approach in terms of the algorithm performance but it is the preferred choice many times for two main reasons. The first reason is that is the easiest and then the fastest solution to the problem of setting parameters values (in terms of the cost of development). Many times it is impossible to test all possible cases for all parameters, but using approximation techniques is easy to find a set of parameter values that works well. And the second reason is closely related to that fact: evolutionary algorithms are usually quite flexible and it is easy to find a configuration of parameters that works; and with a little bit of extra work it is even possible to find a good configuration that performs well for a wide range of problems. It is then not difficult, but costly, to find the best configuration. So parameter tuning is always a reasonable choice, in terms of effort, to find a good configuration of parameters. All these arguments lead us to select parameter tuning as the first option to set permut-POA parameters.

TABLE I
VALUES SELECTED FOR EACH PARAMETER

Parameter	Symbol	Values
Number of groups	N	2, **3**, 5, 10
Group size	L	5, **10**, 20
Candidates per group	θ	1, **2**, 3, 5
Member weighting constant	m	0.5, **1**, 1.5, 2
Candidate weighting constant	n	0.01, **0.1**, 0.5, 1
Merge probability	p_m	0, **0.01**, 0.1
Deletion probability	p_d	0, **0.01**, 0.1
Groups to be merged	λ	**2**, 3
Groups to be deleted	γ	**1**, 2
Mutation probability	p	0, 0.01, **0.1**, 0.5, 1
Duplicate elimination policy	de	**0**, 1

The process to tune the parameters can be summarized in what follows. For each parameter a set of representative values was selected for testing (table I). We tried to cover the wider range of (sensible and sense) possibilities. Then, every test value of each parameter was tested keeping all others parameters constant in a kind of pivoting rule. The central value of each parameter was initially selected for the pivot set. The selection was arbitrary between the two central values in the cases in which the number of values was even. Selected pivoting values are highlighted in bold in table I.

26 experiments were then run to solve each of the 100 random test cases of 25 tasks. The number of calls to the fitness function was recorded. Then a General Linear Model (GLM) was used to determine what parameters had a relevant influence in the algorithm performance. Results are summarized in table II.

TABLE II
RESULTS OF THE GENERAL LINEAR MODEL ANALYSIS

Parameter	Symbol	F	p-value
Number of groups	N	292,22	0,000
Group size	L	14,81	0,000
Candidates per group	θ	1,39	0,243
Member weighting constant	m	2,00	0,112
Candidate weighting constant	n	0,36	0,785
Merge probability	p_m	18,91	0,000
Deletion probability	p_d	17,64	0,000
Groups to be merged	λ	3,10	0,078
Groups to be deleted	γ	0,86	0,355
Mutation probability	p	3,23	0,012
Duplicates elimination policy	de	9,43	0,002

Summary of results returned for each parameter. R^2 = 37,18%. p-values bellow 0,05 (CI=95%) suggest that the parameter setting is relevant for the permut-POA performance.

Results suggest that the values set for 6 (out of 11) parameters (N, L, p_m ,p_d, p and de) have a relevant influence in permut-POA performance. All other parameters (θ, m, n, λ and γ) seem to not be so relevant. It should be noted that relevant parameters refer to the population size (N, L) and its evolution (p_m, p_d) along with the last modifications introduced: mutation probability and the population elimination policy.

Finally, for each parameter that was found to be relevant, a one way analysis of variance (ANOVA) was carried out to determine the optimal values among the candidate values. Figure 2 shows, as an example, the results obtained for the de parameter, which enables the duplication elimination policy. It can be observed that a value of 0 (the policy is disabled) clearly improves performance. Duplication elimination policies are usually enabled to avoid quick convergence to local minima because they ensure diversity in the population. Test cases were randomly generated and we conjecture that an average solution space will have many solutions with little, if any, local minima and thus duplicates existence improves convergence ratios.

After completing ANOVA tests, it is possible to set the optimal value for each relevant parameter. For each parameter that was found to be not relevant its initial value was kept. The final optimal values (for our test cases) were N=2, L=10, θ=2, m=1, n=0.1, $p_m = p_d$ = 0.1, λ=2, γ=1 and p=0.1. With these settings the algorithm succeeded in all test cases.

IV. COMPARATIVE STUDY

After finding that permut-POA dealt successfully with all test cases, the next step was to perform a comparative

analysis with other standard evolutionary methods. Genetic algorithms (GA) are a vastly studied subfield in evolutionary computation and it was our first choice. And particle swarm optimization (PSO) is a more recent optimizer, but it has proven its flexibility and efficiency to solve many problems in a wide range of domains [15]. PSO was our second choice. We may say that permut-POA is in its initial and, thus, standard version, so to try to make a fairer comparative, standard versions of GA and PSO were also implemented.

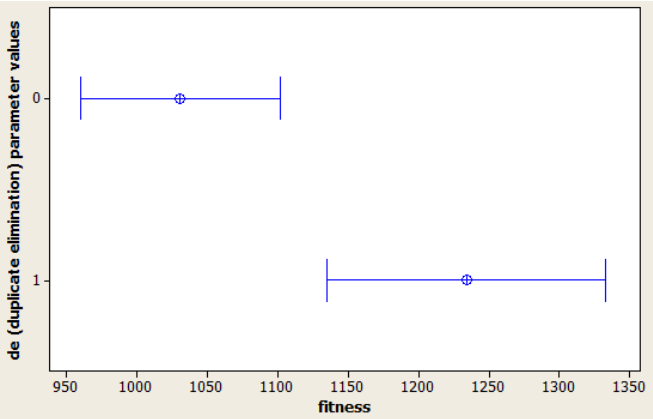


Fig. 2. Interval plot (95% confidence interval for the mean) resulting from running a one way ANOVA for the *de* parameter.

A. Permut-GA

Different approaches to GA can be taken to deal with permutation problems. Standard typologies use to distinguish between *order problems* (e.g. job scheduling problem) and *adjacency problems* (e.g. travel sales person problem) [7]. A specific set of recombination and mutation operators already exists for each of these two kinds of problems. Most common mutation operators are swap mutation, insert mutation and scramble mutation [16]. Additionally, inversion mutation [17] was introduced to handle adjacency problems. Usual choices for recombination operators include partially mapped crossover [18] and edge crossover [19] for adjacency problems; and order crossover [20] and cycle crossover [21] for order based problems. GAs that use specific representation and operators for handling permutations can be called permutation GAs or permut-GAs.

We developed a permut-GA for order problems because all our test cases were instances of this type of problems. Order recombination and swap mutation were chosen. Tournament selection was our strategy for parent selection, and a generational model with elitism was our preferred choice for the replacement strategy. Elitism was introduced to keep track of the best individuals in the population in order to mitigate the possible destructive effect that generational replacement introduces. Our decisions were based mainly on the ease of implementation of each strategy.

The permut-GA was tested and tuned using the approach described in section III for the permut-POA. Our permut-GA has four parameters that require tuning: population size (μ), tournament size (k), mutation rate (p) and elitism's size (n). Their final values were μ=20, k=μ/3=7, p=0.5 and n= μ/2=10. After running all tests, we found that permut-GA also succeeded in all test.

B. Permut-PSO

Original PSO [22, 23] is intended to work on continuous spaces. A discrete binary version of the PSO was presented in [24]. This version uses the concept of velocity as a probability of changing a bit state from zero to one or vice versa. A version that deals with permutation problems was introduced in [10]. In this latter version (permut-PSO hereafter), velocity is computed for each element in the sequence, and this velocity is also used as a probability of changing the element, but in this case, the element is swapped establishing its value to the value in the same position in *nbest* (the best position found so far by the current particle and its neighbors). Mutation is also introduced in permut-PSO to avoid stagnation; just after updating each particle´s velocity, if the current particle is equal to *nbest* then two randomly selected positions from the particle sequence are swapped. In [10] is also demonstrated that permutation PSO outperforms genetic algorithms for the N-Queens problem. So we decided to try this version with all its settings.

Each particle shares its information with a, usually fixed, number of neighbor particles to determine *nbest*. Determining the number of neighbor particles (neighbor size) and how neighborhood is implemented has been a subject of deep research in an area that has been called sociometry. Topologies define structures that determine neighborhood relations, and several of them (ring, four cluster, pyramid, square and all topologies) have been studied. It has been proved that fully informed approaches outperform all other methods [25][1]. The fully informed approach prompts using 'all' topology and a neighborhood size equal to the total number of particles in the swarm (i.e. every particle is connected with all other particles).

One important PSO advantage is that it uses a relatively small number of parameters compared with other techniques like genetic algorithms. However, much literature on PSO parameter subject has been written. Among it, [10] presents a configuration setting that works properly for solving permutation problems. So we decided to follow these recommendations, and parameters were set as follows: Learning rates ($c1$, $c2$) are set to 1.49445 and the inertial weight (w) is computed as $w = 0.5 + (rand()/2)$. Permut-PSO was tested and it succeeded in all test cases.

C. Experimental Comparison

In order to carry out a comparative analysis each agent was executed to solve each of the 100 random test cases. The best parameter configuration for each agent was set. The

[1] What this work really concludes is that an all topology is the best option for the canonical PSO, in terms of succeed rate, for a set of standard test functions. This study stresses the importance that the tuple topology/PSO version conforms.

same population size (20) was used in all cases to have a fair comparison. Fitness evolution over time was computed. Time was measured as the number of calls to the fitness function. Figure 3 summarizes mean results. It can be observed that permut-POA clearly outperforms permut-GA and is competitive with permut-PSO. In order to give some statistical relevance to gathered data, we carried out an analysis of variance over the 100 results (total calls to the fitness function) of each agent. p-values returned for a one-way ANOVA were 0.000 when permut-POA was compared with permut-GA, and 0.022 when permut-POA was compared with permut-PSO. Results confirmed the initial hypothesis but also concluded that there is enough evidence that permut-POA performs also better than permut-PSO (97.8% CI). Performance improvement is 14.3% over the mean.

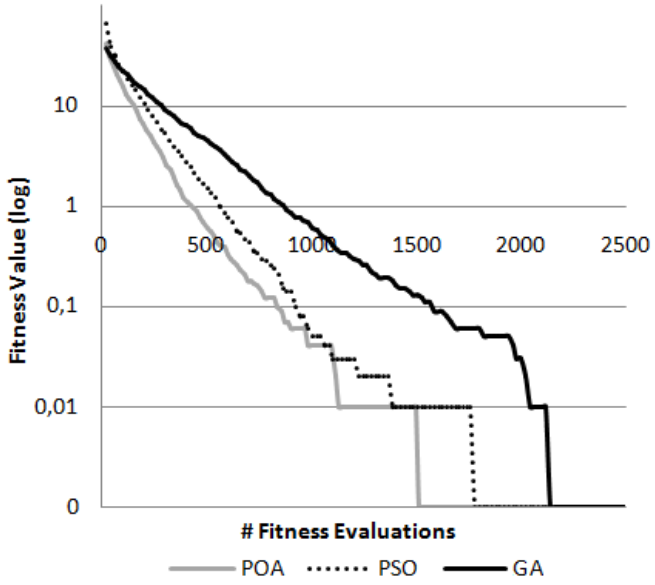


Fig. 3. Experimental results of POA, PSO and GA over 100 randomly generated task scheduling problems (mean values).

To test the scalability of the different agents, the additional test cases (40 to 100 tasks with just one feasible solution) were used. Mean values for 100 runs were computed. Results are presented in figure 4. Results also confirm many previous supposals. Permut-POA and permut-PSO clearly perform better than permut-GA, and both show a surprisingly similar performance. As an example, for 100 tasks POA required 33,427 fitness calls, while PSO required 34,260 and GA 47,891.

V. Further Thoughts

POA has not been already framed in any group of algorithms or optimizers. Evolutionary computation traditionally focuses on those algorithms, methods and techniques that are inspired by any aspect related with natural evolution. But additionally, some other methods which do not explicitly search for inspiration in evolution, but in other natural beings or processes, have been included within the larger evolutionary computation paradigm. Examples are PSO [22, 23], which was originally influenced by fish schooling, and Ant Colony Optimization (ACO) [26] which is inspired by ant foraging behavior.

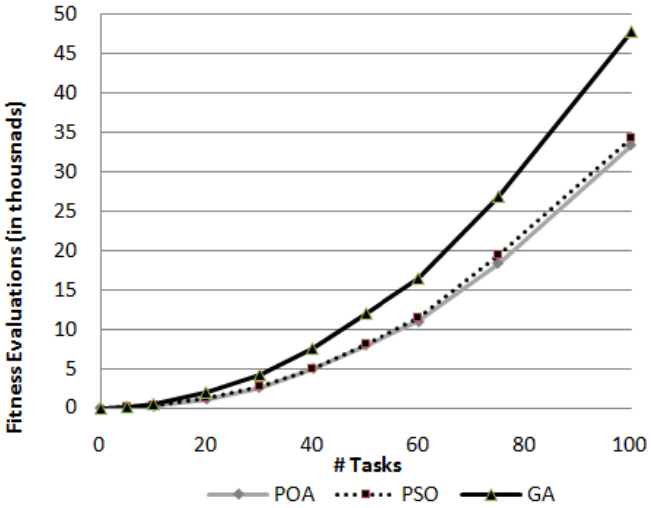


Fig. 4. Agents scalability in experimental task scheduling problems (100 runs, mean values).

Authors do not pretend to be those that frame this novel meta-heuristic under the auspices of any umbrella, only time and colleagues can say (or try to agree) about that. Nonetheless, POAs political metaphor clearly distances it from any natural or biological inspiration, situating it in a social imitation context. What we think is that POAs internal working shows an obvious resemblance with the same internal functions of other evolutionary methods, like those just mentioned. In particular, POA is also a population stochastic optimizer that tries to find a solution to a problem using a fitness-based evolving procedure. But POA emphasizes exclusively on the social dimension of communication that can be observed, for example, in particle swarms. Thus it seems sound to try to compare it with these similar methods to try to solve the same problems.

POA advantages seem to emerge from its ability to grasp parties and politicians optimizing behavior in a simple and easy-to-implement algorithm. POA adjusts the population dynamically maintaining only most fitted clusters of individuals. Simultaneously, POA encourages cooperation among different groups in a kind of promotion of the most fitted individuals, which are compelled to explore most promising areas. It can be said that groups' deletion tries to avoid a waste of resources while groups' merger tries to improve resources allocation. Focus on the social dimension, which is specially evolved in human beings, ignoring any other kind of interaction, seems then to offer important outcomes.

VI. Conclusions and Future Work

POA's ability to grasp politicians and parties behavior has proven its efficiency as an optimization method in numerical

problems. In this paper, we have presented an adaptation of this social metaphor designed to work with permutation sets. Biasing of regular members is redefined as a probability to perform a swap towards a candidate. Results demonstrate the potential of this new method in relation to other well established optimizers. It is also demonstrated that POA outperforms standard versions of genetic algorithms and particle swarms. Improvement is especially relevant when POA is compared with a standard genetic algorithm. POA is, furthermore, easy to develop and seems to scale well. Pointing at its possible drawbacks, it shall be noticed that it has too many parameters. This study just includes an initial analysis concerning parameter tuning that tries to determine which parameters are relevant along with convenient values for them. Experiments suggest that parameters are easy to tune, thus enabling flexible parameters' settings. But, in our opinion, further work is required on this topic to establish possible dependencies between parameters that further facilitate POA implementation and tuning. Results also return a significant resemblance in terms of efficiency when POA is compared with PSO. This may not be so surprising when both metaphors are further inspected. PSO combines local and social exploration, while POA reduces exploration to the social component. POA capacity to dynamically adjust population size is, in authors' opinion, an important feature that may explain POA's observed efficiency when compared with PSO. We think that further research may also be conducted in order to determine possible equivalences between both optimizers, along with hybrid methods devised to exploit both methods strengths. Comparison with other exact and stochastic methods also arises as natural research course to further explore this parliamentary metaphor.

REFERENCES

[1] A. Borji, "A New Global Optimization Algorithm Inspired by Parliamentary Political Competitions," in *6th Mexican International Conference on Artificial Intelligence. LNAI 4827*, A. Gelbukh and A. F. Kuri-Morales, Eds.: Springer-Verlag, 2007, pp. 61-71.

[2] A. Borji and M. Hamidi, "A New Approach to Global Optimization Motivated by Parliamentary Political Competitions," *International Journal of Innovative Computing, Information and Control,* vol. 5, pp. 1643-1653, 2009.

[3] M. L. Pinedo, *Scheduling: Theory, Algorithms and Systems*. New York: Springer, 2008.

[4] E. Tsang, *Foundations of Constraint Satisfaction*. London: Academic Press, 1993.

[5] J. Robinson and Y. Rahmat-Samii, "Particle swarm optimization in electromagnetics," *Antennas and Propagation, IEEE Transactions on,* vol. 52, pp. 397-407, 2004.

[6] L. Schoofs and B. Naudts, "Ant colonies are good at solving constraint satisfaction problems," in *Proceedings of the 2000 Congress on Evolutionary Computation.*, La Jolla, CA, 2000, pp. 1190-1195.

[7] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Berlin (Germany): Springer-Verlag, 2003.

[8] K. A. de Jong and J. Sarma, "On Decentralizing Selection Algorithms," in *Proceedings of the 6th international Conference on Genetic Algorithms*, L. J. Eshelman, Ed. San Francisco (USA): Morgan Kaufmann Publishers, 1995, pp. 17 - 23.

[9] J. E. Baker, "Reducing Bias and Inefficiency in the Selection Algorithm," in *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and their Application*, Cambridge, USA, 1987, pp. 14 - 21.

[10] X. Hu, R. C. Eberhart, and Y. Shi, "Swarm intelligence for permutation optimization: a case study of n-queens problem," in *Proceedings of the 2003 IEEE Swarm Intelligence Symposium*, Indianapolis, USA, 2003, pp. 243-246.

[11] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence through Simulated Evolution*. Chichester (UK): Wiley, 1966.

[12] J. H. Holland, *Adaptation In Natural and Artificial Systems*. Michigan (USA): The University of Michigan Press, 1975.

[13] I. Rechenberg, *Evolution Strategy: Optimizing Technical System based on Biological Evolution Principles*. Stuttgart (Germany): Fromman-Hozlboog Verlag, 1973.

[14] P. Prosser, "An empirical study of phase transitions in binary constraint satisfaction problems," *Artificial Intelligence,* vol. 81, pp. 81-109, 1996.

[15] R. Poli, J. Kennedy, T. Blackwell, and A. Freitas, "Analysis of the Publications on the Applications of Particle Swarm Optimisation," *Journal of Artificial Evolution and Applications,* vol. 2008, p. 3, 2008.

[16] G. Syswerda, "Schedule Optimisation Using Genetic Algorithms," in *Handbook of Genetic Algorithms*, L. Davis, Ed. Washington (USA): Thomson Publishing, 1991, pp. 332-349.

[17] S. Lin and B. Kernighan, "An Effective Heuristic Algorithm for the Travel Salesman Problem," *Operations Research,* vol. 21, pp. 498-516, 1973.

[18] D. E. Goldberg and R. Lingle, "Alleles, Loci and the Traveling Salesman Problem," in *Proceedings of the 1st International Conference on Genetic Algorithms and their Application*, Cambridge (USA), 1985, pp. 154-159.

[19] L. D. Whitley, "Permutations," in *Evolutionary Computation 1: Basic Algorithms and Operators*, T. Bäck, L. J. Fogel, and Z. Michalewicz, Eds. Bristol (UK): Institute of Physics Publishing, 2000, pp. 274-284.

[20] L. Davis, *Handbook of Genetic Algorithms*. Washington (USA): Thomson Publishing, 1991.

[21] I. M. Olivier, D. J. Smith, and J. H. Holland, "A Study of Permutation Crossover Operators on the Traveling Salesman Problem," in *Proceedings of the 1st International Conference on Genetic Algorithms and their Application*, Cambridge (USA), 1985, pp. 154-159.

[22] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proceedings of the Sixth International Symposium on Micro Machine and Human Science. MHS '95.*, Nagoya, Japan, 1995, pp. 39-43.

[23] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings., IEEE International Conference on Neural Networks.*, Perth, WA, Australia, 1995, pp. 1942-1948.

[24] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *1997 IEEE International Conference on Systems, Man, and Cybernetics. 'Computational Cybernetics and Simulation'.* 1997, pp. 4104-4108.

[25] R. Mendes, J. Kennedy, and J. Neves, "The fully informed particle swarm: simpler, maybe better," *Evolutionary Computation, IEEE Transactions on,* vol. 8, pp. 204-210, 2004.

[26] M. Dorigo and G. D. Caro, "The Ant Colony Optimization meta-heuristic," in *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds. London, UK: McGraw Hill, 1999, pp. 11-32.