Parallel Differential Evolution Algorithms for Stackelberg-Nash Bilevel Optimization Problems

1st Thiago Tavares Magalhães
National Laboratory for Scientific Computing
Petrópolis, Brazil
thiagotm@lncc.br

2nd Helio J. C. Barbosa
National Laboratory for Scientific Computing[1]
Federal University of Juiz de Fora[2]
{[1]Petropólis, [2]Juiz de Fora}, Brazil
hcbm@lncc.br

***Abstract*—The Bilevel Programming Problems with inter-dependent followers are characterized by three or more optimization problems nested in a hierarchical structure of the type leader-followers whose solution configures a Stackelberg-Nash equilibrium game. Their complexity and structure make many of them almost impracticable for solving via classical deterministic methods. Thus, population-based metaheuristics have been used to handle this class of problems. However, while the meta-models-based implementations are not indicated to tackle the inter-dependent followers BLPs, their nested structure can greatly increment the already high computational cost of the metaheuristics. Thus, we offer and study the three first parallel models for solving Stackelberg-Nash equilibrium BLPs via metaheuristics. All our parallel approaches are scalable and can be efficiently executed in individual multi-cores computing nodes and in high-distributed computational clusters. We reported relevant speedups for different computing architectures, which endorsed the potential of the proposals of this work as relevant tools to enable the study of increasingly larger and complex inter-dependent followers BLPs.**

***Index Terms*—Bilevel Programming, Stackelberg-Nash game, Parallel Computing, Parallel Metaheuristics**

I. Introduction

The Bilevel Programming Problems (BLPs) constitute a useful class of models to describe many optimization problems, involving multiple agents, frequent in the real world. In spite of the additional challenges in handling BLPs, the literature points out a wide set of applications. We can cite works related to network problems [5], economics [6], agriculture [7], biotechnology [8], among others. Surveys about BLP real-world applications can be found in [9] and [10].

However, relating a diverse set of problems in a single hierarchical structure, makes BLPs considerably more complex to handle than the common single-level programming problems. Thus, given some important limitations of the classical methods, BLPs are increasingly handled via metaheuristics, which do not require strong conditions or prior knowledge about the objective functions that describe the problem.

However, the use of iterative and stochastic methods such as the populational metaheuristics to solve BLPs usually leads to notably high computational costs. This barrier is responsible for restricting the literature for solving BLPs of few dimensions, in comparison with the traditional single-level optimization works [26]. Also, we rarely find in the literature examples with a high number of followers being tackled. As an alternative to transpose these challenges, in [2] we suggested high-performance computing models aiming at exploiting both the natural parallelism of the metaheuristics [1] as well as the modern hardware resources available. Efforts of this nature aim at enabling the proper solution of increasingly complex and high-dimensional BLPs. In spite of that, the literature still lacks parallel solutions to handle an important type of BLPs: BLPs with inter-dependent followers, which arise in the Stackelberg-Nash model, briefly described in the following.

We thank the support from CAPES (grant 1577622/2016) and CNPq (grant 312337/2017-5).

II. Bilevel Programming Problems with Inter-Dependent Followers

The Bilevel Programming Problems describe cases that aggregate different agents aiming at different objectives that mutually influence each other in a hierarchical structure. For example, some competing companies making decisions aiming at maximizing each one's profit. Assuming that one of them – the leader –, acts first, all others – the followers – react to the leader's decision. This hierarchical structure of a leader influencing followers and be influenced by them characterizes the Stackelberg Game Model. In addition, there are cases where followers also influence each other. Those will be referred to here as the multiple inter-dependent followers cases. Solving problems of this type implies finding the Stackelberg Equilibrium – between leader and followers – and also the Nash equilibrium – in the lower hierarchical level, among followers. We refer to problems of this nature as Stackelberg-Nash Problems [4]. Denoting by $F(x, y_1, \ldots, y_{nf})$ the objective function to be optimized by the leader, $f_i(x, y_1, \ldots, y_{nf})$ the objective function to be optimized by the i-th follower and $g_i(x, y_1, \ldots, y_{nf})$ the constraint set for the i-th follower, a Stackelberg-Nash model can be written as in the Eq. 1, and an inter-dependent followers BLP arises.

$$\begin{aligned} &\min_{x \in X} F(x, y_1, \ldots, y_{nf}) \\ &\text{subject to } G(x, y_1, \ldots, y_{nf}) \leq 0 \\ &\qquad \min_{y_i \in Y_i} f_i(x, y_1, \ldots, y_{nf}) \\ &\qquad \text{subject to } g_i(x, y_1, \ldots, y_{nf}) \leq 0 \end{aligned} \tag{1}$$

In BLPs, we refer to the leader and follower optimization functions as upper- and lower-level problems, respectively. For the inter-dependent followers cases, any lower-level function f_i depends on the leader x_i variables and also on all lower-level $y_1, \ldots, y_{nf}$. Thus, the followers share information with each other, influencing and being influenced mutually.

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

III. Related Works and Goals

Although many works employ deterministic methods to handle BLPs, the complexity of these problems and some limitations of the traditional methods justify the increasing use of metaheuristics to solve problems of this class. In this context, the literature reports a set of solvers that nest two or more metaheuristics for solving independent-followers [13] [14] [15] and inter-dependent-followers BLPs [22] [23] [24] [25]. Also, there are approaches that hybridize exact and stochastic methods [12], that modify multi-objective algorithms in order to search for the Nash-equilibrium using the Nash domination concept [32], and others that employ meta-models aiming at reducing the high computational cost of the nested-metaheuristics-based methods [16] [17]. However, the literature still is restricted to BLPs with a few followers and low dimensionality, in comparison to the single-level optimization problems [26]. Especially concerning BLPs with inter-dependent-followers, their structure makes it high costly to solve via the aforementioned nested-based methods and the computational complexity of the tasks required to find a set of solutions based on the Nash domination concept increases significantly as the number of followers increase. Also, the complexity of BLPs is a barrier to the achievement of satisfactory results via techniques based on approximations, like the metamodel-based methods.

Thus, in [11] we indicated the feature known as natural parallelism of metaheuristics as a potentially important alternative to enable the development of less execution time demanding optimization algorithms without performing approximations or reducing optimization capability. We offered two parallel computing models based on the Bilevel Differential Evolution (BLDE), a recent nested metaheuristics method initially developed to handle BLPs with independent followers [20]. Reporting promising performances in a single machine, our models were later validated in a multiple-node environment [2]. Those efforts seem to be the first ones in the literature that associated modern high-computing resources with nested metaheuristics aiming at improving the handling of BLPs. In turn, the BLDE algorithm was improved in [21] to enable the solution of BLPs with inter-dependent followers. However, for this class of BLPs no proposal has been reported in the literature in order to provide parallel computing solvers.

Therefore, in this work we design and validate three parallel models also based on the BLDE algorithm, but able to achieve the Stackelberg-Nash equilibrium that characterizes the inter-dependent followers BLPs. Thus, jointly with our previous proposals, we hope that the efforts of which this work is part met an important need in the BLPs literature, employing high-performance computing in order to enable the proper treatment of increasingly complex and large BLPs, for both independent- and inter-dependent followers cases.

IV. Differential Evolution Algorithms for BLPs

A. Canonical Differential Evolution

Proposed by Storn & Price, the Differential Evolution (DE) is a stochastic metaheuristic to solve optimization problems [18]. The DE functioning is based on the evolution of a population of candidate solutions that probabilistically tends to become increasingly fitter to solve a given optimization problem $F(\overrightarrow{x})$. In this context, a candidate solution $\overrightarrow{x}$ is considered promising according to $F(\overrightarrow{x})$ value. We referred to this evaluation value of a solution as *fitness* value. Thus, the DE process occurs over a number gen of generations, employing the DE movement operators known as *mutation* and *crossover* (or *recombination*) and using a selection criterion that maintains promising solutions for more generations in the evolutionary process. For a population of np individuals, the canonical DE algorithm is simply presented in the Alg. 1 and in the Fig. 1.

Algorithm 1: Simplified DE algorithm

```
Create random initial population with np individuals
  x_{i,G} and evaluate all them doing F(x_{i,G});
for G ← 1 to gen do
    for i ← 1 to np do
        Generate trial vector u_{i,G+1} via DE operators;
        Evaluate u_{i,G+1} doing F(u_{i,G+1});
        Compare F(x_{i,G}) and F(u_{i,G+1}) and keep
          only the best in the next generation;
```

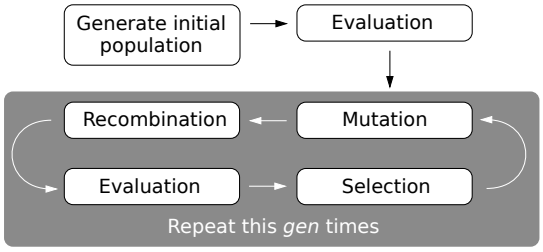

Fig. 1. Simple DE algorithm functioning

Controlled by the parameters F and CR, there are different variants of mutation and crossover in DE [19] that are applied to all individuals in the population, one at a time, for each iteration. The current individual that is undergoing mutation and recombination, x_i, is named *target vector*, whilst the new candidate solution that results from this operation, u_i, is named *trial vector*. In addition, note that the higher gen and np parameters, more function evaluations, mutations, crossovers and comparisons are performed, increasing the computational cost of the DE.

Thus, characterized by its simplicity and robustness, DE constitutes a population-based iterative technique useful to satisfactorily solve a wide range of optimization problems, being increasingly highlighted in the literature. However, the canonical DE can only handle single-level optimization problems.

B. BLDE for Independent Followers Bilevel Problems

In order to enable the solution of BLPs via DE operators, [20] proposed the Bilevel Differential Evolution (BLDE).

Basically this solution consists of hierarchically nested DE algorithms, one handling the upper-level variables of the BLP and others handling the lower-level variables of the followers. Denoting by $< \overrightarrow{x}_{i,G}, \overrightarrow{y}_{i,1,G}, \ldots, \overrightarrow{y}_{i,nf,G} >$ the i-th candidate solution at the G-th iteration composed of upper-level variables $\overrightarrow{x_i}$ and nf $\overrightarrow{y}_{i,j,G}$ lower-level variables, the BLDE algorithm is presented in the Alg. 2.

Algorithm 2: Simplified BLDE algorithm

```
Create random initial population with np individuals;
< x_{i,G}, y_{i,1,G}, ..., y_{i,nf,G} > and evaluate all them
 doing F(x_{i,G}, y_{i,1,G}, ..., y_{i,nf,G});
for G ← 1 to gen do
    for i ← 1 to np do
        Generate trial vector u_{i,G+1} via DE operators;
        for j ← 1 to nf do
            Get the optimum y_{i,j,G+1} for
              f_j(u_{i,G+1}, y_{i,j,G+1}) via follower DE;
        Evaluate < u_{i,G+1}, y_{i,1,G+1}, ..., y_{i,nf,G+1} >
          doing F(u_{i,G+1}, y_{i,1,G+1}, ..., y_{i,nf,G+1});
        Compare F(x_{i,G}, y_{i,1,G}, ..., y_{i,nf,G}) and
          F(u_{i,G+1}, y_{i,1,G+1}, ..., y_{i,nf,G+1}) and
          keep only the best in the next generation;
```

Thus, in comparison with the canonical DE algorithms, the BLDE performs nf complete and self-contained DE executions for each trial vector that arises along the iterative process. It ensures that for each new candidate solution $< \overrightarrow{x}, \overrightarrow{y}_i, \ldots, \overrightarrow{y}_{nf} >$ all lower-level variables $\overrightarrow{y}_i$ are the ones that optimize $f_i(\overrightarrow{x}, \overrightarrow{y}_i)$. Also, evaluating regarding the upper-level problem this new trial with its optimized lower-level variables, the BLDE alters the traditional DE behavior in order to guide the search for solutions that tend to the Stackelberg equilibrium.

C. BLDE for Inter-dependent Followers Bilevel Problems

For BLPs with inter-dependent followers, a BLDE able to find the Nash equilibrium between the lower-level variables was developed in [21]. The difference between this approach and the BLDE for Independent Followers BLPs is that for each trial vector, the lower-level variables also are evolved in an additional iterative process that repeats *Nash cycles* times, as presented in the Alg. 3.

Algorithm 3: Simplified Nash equilibrium algorithm

```
Generate trial vector u_{i,G+1} via DE operators;
for k ← 1 to n_cycles do
    for j ← 1 to nf do
        Get the optimum y_{i,j,G+1} for
          f_j(u_{i,G+1}, y_{i,j,G+1}, ... y_{i,nf,G+1}) via
          follower DE;
```

Thus, for a BLP with two followers, for each new trial vector $< \overrightarrow{u}, \overrightarrow{y}_1, \overrightarrow{y}_2 >$ from the upper level population, $\overrightarrow{y}_1$ is first optimized for $f_1(\overrightarrow{u}, \overrightarrow{y}_1, \overrightarrow{y}_2)$ considering a randomly generated $\overrightarrow{y}_2$. Then, this obtained $\overrightarrow{y}_1$ is fixed and $\overrightarrow{y}_2$ is optimized for $f_2(\overrightarrow{u}, \overrightarrow{y}_1, \overrightarrow{y}_2)$. Thus, for *nash cycles* times, the follower variables are alternately optimized in order to provide lower-level variables in Nash equilibrium.

Considering all this structure, we can easily note the reasons that make the interdependent-followers BLPs handling via nested metaheuristics a high computing demanding task. For each trial vector that arises in a given generation, a series of other auto-complete DE algorithms need be performed in order to achieve the Nash Equilibrium between the followers. A simple and sensitive change of the budget parameters at any hierarchical level has the potential to greatly intensify the computational costs.

V. Parallel Models for Inter-dependent Followers Problems

Based on the BLDE for inter-dependent followers problems, we implemented three scalable parallel models to achieve the Stackelberg-Nash Equilibrium. All of them aim at exploring the hierarchical BLP structure and the natural parallelism of the population-based metaheuristics.

A. The Upper-level Parallel Model

The simpler approach is named upper-level parallel model (P.M.1) and divides the upper-level population of trial vectors in a distributed-memory environment. Firstly, the DE movement operators are applied to the upper-level population and a new population of trial vectors – with only the values of the upper-level variables – is generated. Thus, denoting by *n_proc* the number of processing instances, the upper-level population with its respective trial vectors is partitioned into *n_proc* sections of $np/$*n_proc* individuals. More deeply, a master process is designed to distribute the upper-level population sending $np/$*n_proc* individuals to each available processing instance. After this, as each section is assigned to one processing instance, the sections are handled in parallel. Each processing instance evolves the lower-level variables for all the trial vectors received, performing sequentially new DE instances according to the Nash equilibrium routine. The Fig. 2 provides a simple logical flowchart of this procedure.

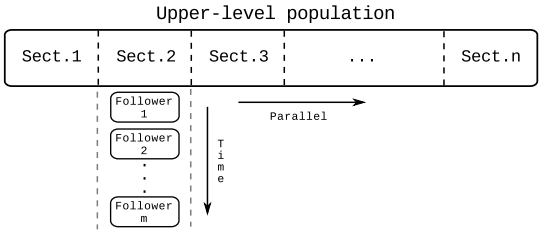

Fig. 2. Upper-level parallel model.

Thus, each trial vector with its optimized lower-level variables is evaluated regarding the upper-level problem and compared with its respective target vector. After handling all its received trial vectors, each process sends the best individuals of its section – among target and trial vectors – back to the

master process, which aggregates all into a complete upper-level population and again applies the DE movement operators.

B. The Upper- and Lower-level Parallel Models

Besides the upper-level parallel model, we implemented two approaches that partition the upper-level population dedicating some hardware resources to enable the concurrent solution of the lower-level problems for each candidate solution. Thus, we named them as upper- and lower-level parallel models. The first one (P.M.2a) works in a completely distributed-memory environment, while the second (P.M.2b) is a hybrid strategy that handles the lower-level variables in a shared-memory environment.

In the P.M.2a the population is partitioned into $\frac{n\_proc}{nf}$ sections of $\frac{np}{n\_proc/nf}$ candidate solutions. Thus, a group of *nf* processing instances are allocated for each section, one per follower, which will enable the parallel optimization of the follower's variables inside each upper-level population partition, differently from what occurs in P.M.1. The Fig. 3 presents a general view of this approach.

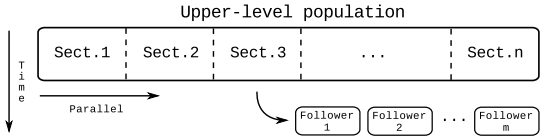

Fig. 3. Upper- and lower-level parallel model.

More deeply, an upper-level partition is assigned to nf distributed-memory processing instances, each one of them receiving a copy of the same $\frac{np}{n\_proc/nf}$ individuals and treating the lower-level variables of one follower. However, since in P.M.2a the followers are handled by different processing instances in a distributed-memory approach, a given follower cannot access the lower-level variables assigned to another, except through message passing. Thus, considering that our solvers are handling Stackelberg-Nash BLPs, at the end of each Nash cycle an all-to-all communication between the processing instances of the same partition will be needed to share the lower-level variables and enable the Nash-equilibrium computing. Also, as the upper-level objective function value depends of all the lower-level variables, an additional all-to-all communication is performed at the end of the last Nash cycle to provide all the evolved lower-level variables for each process by the same section. This occurs in order to enable that each processing instance can evaluate regarding to the upper-level problem, compare and select in parallel $\frac{np}{n\_proc}$ individuals of its respective section. Finally, the master process can receive all individuals and aggregate them into a complete upper-level population, as occurs in P.M.1.

In turn, for P.M.2b the upper-level is sectioned in a distributed-memory environment exactly as occurs in P.M.1. Are assigned np/n_proc upper-level trial vectors for each one of *n_proc* processing instances. However, at the lower-level each process forks into nf processing threads. Thus each thread evolves via DE the lower-level variables corresponding to one follower, in parallel and in a shared-memory environment. As a result, at the end of each Nash cycle, the followers can share information without the need to pass messages. The same occurs after the last Nash cycle to enable that the trial vectors can be evaluated in parallel regarding to the upper-level problem. Briefly, the P.M.2b is a hybrid upper- and lower-level parallel model that employs the shared-memory paradigm as an alternative to provide the concurrently handling of the followers eliminating the all-to-all communications that are part of the P.M.2a.

For all models, we employed the method proposed by Deb for constraints handling, which establishes the following criteria: (*i*) feasible solutions are preferred to infeasible solutions, (*ii*) among feasible solutions the one having better fitness is preferred and (*iii*) among infeasible solutions the one having smaller constraints violation is preferred [27]. Also, regarding to the DE movement operators, for each target vector, we perform a draw to select one of the traditional mutation and recombination configurations *rand/1/bin*, *best/1/bin*, *target-to-rand/1/bin* and *target-to-best/1/bin*. For scope reasons, we do not detail them in this work, but all are exposed in [19].

VI. Experiments

Our three suggested parallel models were implemented using MPI [29] and OpenMP [30] technologies to provide the distributed- and shared-memory environments, when necessary. A set of experiments was designed in order to study the time performances of the parallel models, enabling comparisons between our strategies considering both a single computing multi-core node and also a highly distributed multiple-nodes computing architecture.

Thus, to measure execution times, we submitted the models to the task of handling each one of the Stackelberg-Nash BLPs described in the Eq. 2 to 5. In the following, we give a brief description of these problems, and refrain from providing a detailed discussion about them, in order to maintain the focus on the execution times and parallelism issues. Deeper considerations about the problems can be found in [21].

All problems have two followers, where x is the leader's variables vector, and y, z the decision variable vectors respectively controlled by the followers. For $F1$ [21], $x \in [-5, 5]^2$, $y, z \in [0, 10]$ and the best solution from the literature gives $F1^* = 3$, $f_1^* = 1$, and $f_2^* = 1$. For $F2$ [28], the best solution reported in the literature gives $F2^* = 22$, $f_1^* = 1$, $f_2^* = -1$. For $F3$ [28], the best solution gives $F3^* = 23$, $f_1^* = 0$, $f_2^* = 8$, better than the first reference solution and obtained in [21]. Finally, for $F4$ and $F5$ [22], $x \in [0, 1]$ and $y, z \in [0, 1]$. We named $F4$ the problem described in the Eq. 5 considering a maximization problem at the upper-level, with the optimum solution reported in the literature leading to $F4^* = 0.0185$, $f_1^* = 0$, and $f_2^* = 0$. In turn, as $F5$ we consider the problem denoted by the Eq. 5 in its original formulation, having two

optimum solutions giving $F4^* = 0$, $f_1^* = 0$, and $f_2^* = 0$.

$$\begin{aligned} &\min_x F1(x,y,z) = -(x1+x2)^2 + \frac{1}{2}(y^2+z^2) \\ &\text{s.t. } \min_y f_1(x,y,z) = (y-(x_1+2))^2 + ((x_2+1))^2 \\ &\quad \min_z f_2(x,y,z) = (y-(x_1+1))^2 + ((x_2+2))^2 \end{aligned} \tag{2}$$

$$\begin{aligned} &\max_x F2(x,y,z) = 7x+5y+8z \\ &\text{s.t. } \max_y f_1(x,y,z) = 3y+z \\ &\quad \max_z f_2(x,y,z) = y-z \\ &\quad \text{s.t.} \quad x+y+z \le 3, \quad -x+y \le 0, \quad y+z \le 2, \\ &\qquad x-y-z \le 1, \quad x,y,z \ge 0 \end{aligned} \tag{3}$$

$$\begin{aligned} &\max_x F3(x,y,z) = 7x+5y+8z \\ &\text{s.t. } \max_y f_1(x,y,z) = -2y^2 - 2yz + 3y \\ &\quad \max_z f_2(x,y,z) = -yz - z^2 + 6z \\ &\quad \text{s.t.} \quad x+y+z \le 3, \quad -x+y \le 0, \quad y+z \le 2, \\ &\qquad x-y-z \le 1, \quad x,y,z \ge 0 \end{aligned} \tag{4}$$

$$\begin{aligned} &\min_x F4(x,y,z) = z(x-y-\frac{1}{2})^2 \\ &\text{s.t. } \min_y f_1(x,y,z) = (z-y)^2 + (2y-x)^2 \\ &\quad \min_z f_2(x,y,z) = (y-z)^2 + \frac{(2z-x)^2}{2} \end{aligned} \tag{5}$$

We opted to set the control parameters as close as possible to the values in the literature. We set $F = 0.7$ and $CR = 0.9$. For all the lower-level problems, we established *Nash cycles* and *genf* as $10 \times (10+10)$ in order to maintain 200 generations at the lower-level (10 iterations for each follower, performed for 10 Nash cycles), and $npf = 30$. For the upper-level problems, we configured the leaders with $gen = 32$ and $np = 48$. These values ensure a budget similar to the literature and provide an upper-level population size that can be equally divided for various number of sections, which is important to maintain the load balancing for different configurations of experiments.

Thus, for each problem, we executed the P.M.1 with $\{2, 4, 8, 12\}$ processing instances, the P.M.2a with $\{4, 8, 12\}$ processing instances and the P.M.2b with $\{1, 2, 4, 8, 12\}$ distributed-memory processing instances – which fork into 2 shared-memory processing threads each one. The executions were performed in a cluster architecture composed by computing nodes with 2 CPUs *Intel Xeon E5-2695v2 Ivy Bridge* of 2.4 GHZ and 12 processing cores per CPU. To allow for performance comparisons between different computing architectures, we performed all experiments as follows:

- Single node (*sn*): each test was assigned to a single computing node that executes the complete parallel model. The parallelism occurs intra-node, according to the 24 processing cores that exist per computing node;
- Multiple nodes (*mn*): assigning each distributed-memory processing instance of a parallel model to a different computing node. Each test was performed by a varied number of computing nodes, distributing its upper-level population sections for them.

In addition, we performed sequential BLDE executions for all problems in order to compute *speedups* and *efficiencies*. The speedup measures the ratio between the sequential and the parallel execution times, while the efficiency measures the ratio between the speedup and the number of processing instances. Each experiment involved 30 independent runs, and the results are displayed in this work in terms of the medians of these multiple executions.

VII. Results and discussion

Firstly, for all problems, Table I presents the execution times and the fitness values of the best solutions obtained by the sequential BLDE implementation. Regarding to the optimum fitness recorded, similar results were achieved by all the parallel configurations. This is important as it indicates the validity of the suggested parallel models as Stackelberg-Nash problems solvers. However, also for this reason, we focus our discussion and the next data on execution time variations.

TABLE I
Experiments for sequential BLDE execution

	Execution Time Data (*s*)				Object. Funct. Values		
Prob	Min	Max	Med.	StdDev.	F	$f1$	$f2$
$F1$	10.01	10.09	10.03	0.02	3.0	1.0	1.0
$F2$	9.58	9.91	9.61	0.12	22.0	1.0	-1.0
$F3$	9.77	9.85	9.78	0.03	23.0	0.0	8.0
$F4$	9.94	10.67	9.97	0.20	0.0185	0.0	0.0
$F5$	9.95	10.11	9.99	0.07	0.0	0.0	0.0

Tables II and III show the execution times for all test problems. The results were arranged in two Tables for size and organization reasons. The *npr* gives the number of distributed-memory processing instances for each configuration noting that for P.M.2b each one of them is also forked into nf shared-memory processing instances. All tables display data from the executions with a single computing node (*sn*) and also data from the configurations that employ multiple computing nodes (*mn*).

Following what is shown in Table I, for this set of experiments the problems do not significantly influence the median execution times. Possibly, the inclusion of specific BLPs – with relevant differences between evaluation costs of leader's and followers' objective functions, for example – could motivate different conclusions. However, for our cases of study, behaviors recorded for a given problem also can be observed in the other ones.

Firstly, we can note the good speedups provided by our parallel models. The maximum one was 14.82, reported when $F3$ was solved via P.M.2b using a configuration with 12 processing instances and multiple computing nodes. In this context, the Figs. 4 and 5 show the executions times demanded by the different configurations of P.M.1 (left) and P.M.2a (right), respectively for single and multiple computing nodes.

On the other hand, Fig. 6 shows the execution times demanded by the P.M.2b, for single (left) and multiple (right) computing nodes. In this case, the curves have two different values for *processes = 1* because the horizontal axis marks

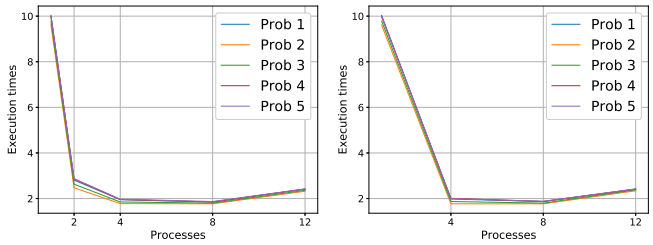

Fig. 4. Execution times for different configurations of P.M.1 and P.M.2a and a single computing node

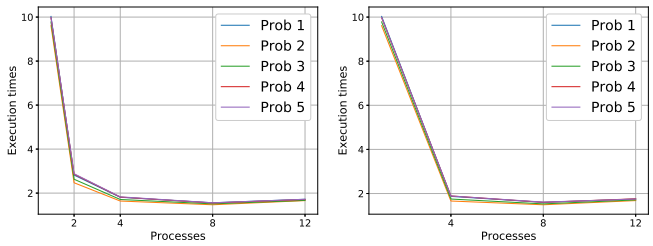

Fig. 5. Execution times for different configurations of P.M.1 and P.M.2a and multiple computing nodes

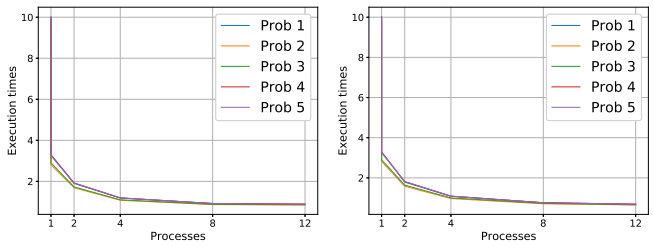

Fig. 6. Execution times for different configurations of P.M.2b, for single and multiple computing nodes

The Figs. 9 and 10, respectively for 4 and 12 distributed-memory processing instances, show the differences between the executions times demanded by our parallel models to handle the problem $F4$ with single (left) and multiple (right) computing nodes. The most notable results concern the P.M.2b, while P.M.1 and P.M.2a achieve more similar performances. In addition, we also could analyze the execution times considering the sum of the distributed- and shared-memory processing instances, comparing P.M.2a in $npr = 8$ with P.M.2b in

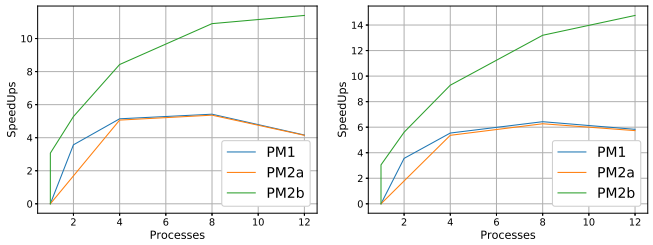

Fig. 7. Speedups for solving Prob $F1$ via all parallel models, for single and multiple computing nodes

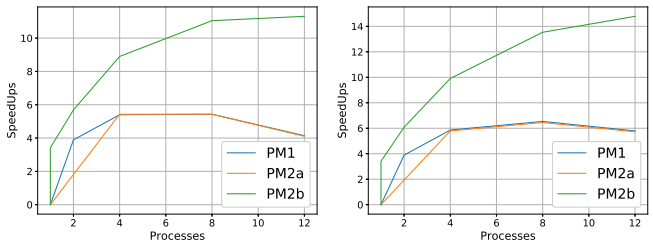

Fig. 8. Speedups for solving Prob $F2$ via all parallel models, for single and multiple computing nodes

$npr = 4$, for example. Even so, P.M.2b obtains the best results. Probably the especially satisfactory performance of this model is a consequence of the shared-memory strategy at the lower-level, which enables the concurrent handling of followers without the undesirable and frequent all-to-all communications that are needed in P.M.2a.

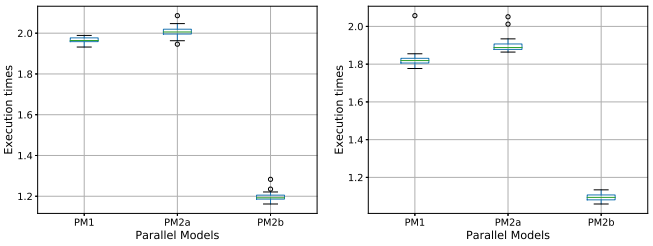

Fig. 9. Execution times for solving Prob $F4$ via all parallel models and 4 processing instances, for single and multiple computing nodes

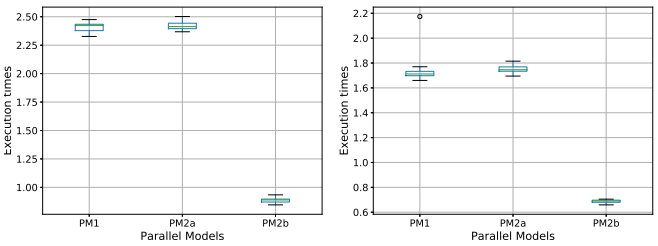

Fig. 10. Execution times for solving Prob $F4$ via all parallel models and 12 processing instances, for single and multiple computing nodes

Also, Tables II to III present several super-linear speedups, especially for the configurations that employ multiple computing nodes for each parallel execution. A similar behavior was

TABLE II
TIME DATA FOR PROBLEMS $F1$ TO $F3$

		Single Node configuration				Multiple Nodes configuration			
P.M.	npr	Med.	Std.	SpeedUp	Effic.	Med.	Std.	SpeedUp	Effic.
Problem $F1$									
PM1	2	2.81	0.10	3.57	1.78	2.82	0.07	3.56	1.78
	4	1.95	0.06	5.14	1.28	1.81	0.03	5.54	1.38
	8	1.85	0.02	5.42	0.68	1.56	0.02	6.43	0.80
	12	2.41	0.03	4.16	0.35	1.72	0.06	5.83	0.49
PM2a	4	1.98	0.01	5.07	1.27	1.87	0.02	5.36	1.34
	8	1.87	0.02	5.36	0.67	1.60	0.10	6.27	0.78
	12	2.42	0.07	4.14	0.34	1.75	0.07	5.73	0.48
PM2b	1	3.27	0.08	3.07	3.07	3.29	0.08	3.05	3.05
	2	1.90	0.02	5.28	2.64	1.79	0.08	5.60	2.80
	4	1.19	0.02	8.43	2.11	1.08	0.04	9.29	2.32
	8	0.92	0.04	10.90	1.36	0.76	0.58	13.20	1.65
	12	0.88	0.02	11.40	0.95	0.68	0.59	14.75	1.23
Problem $F2$									
PM1	2	2.47	0.01	3.89	1.94	2.47	0.02	3.89	1.94
	4	1.78	0.03	5.40	1.35	1.64	0.02	5.86	1.46
	8	1.77	0.02	5.43	0.68	1.47	0.03	6.54	0.82
	12	2.32	0.13	4.14	0.34	1.66	0.10	5.79	0.48
PM2a	4	1.77	0.01	5.43	1.36	1.66	0.02	5.79	1.45
	8	1.77	0.03	5.43	0.68	1.49	0.07	6.45	0.81
	12	2.34	0.03	4.11	0.34	1.68	0.02	5.72	0.48
PM2b	1	2.81	0.08	3.42	3.42	2.82	0.08	3.41	3.41
	2	1.69	0.03	5.69	2.84	1.58	0.03	6.08	3.04
	4	1.08	0.05	8.90	2.22	0.97	0.02	9.91	2.48
	8	0.87	0.03	11.05	1.38	0.71	0.01	13.54	1.69
	12	0.85	0.04	11.31	0.94	0.65	0.06	14.78	1.23
Problem $F2$									
PM1	2	2.63	0.07	3.72	1.86	2.63	0.02	3.72	1.86
	4	1.85	0.02	5.29	1.32	1.71	0.02	5.72	1.43
	8	1.80	0.04	5.43	0.68	1.51	0.02	6.48	0.81
	12	2.36	0.08	4.14	0.34	1.67	0.02	5.86	0.49
PM2a	4	1.87	0.03	5.23	1.31	1.75	0.04	5.59	1.40
	8	1.80	0.03	5.43	0.68	1.53	0.02	6.39	0.80
	12	2.38	0.04	4.11	0.34	1.71	0.18	5.72	0.48
PM2b	1	2.90	0.08	3.37	3.37	2.90	0.10	3.37	3.37
	2	1.73	0.03	5.65	2.82	1.64	0.05	5.96	2.98
	4	1.10	0.06	8.89	2.22	1.00	0.03	9.78	2.44
	8	0.88	0.04	11.11	1.39	0.73	0.11	13.40	1.68
	12	0.86	0.03	11.37	0.95	0.66	0.04	14.82	1.24

TABLE III
TIME DATA FOR PROBLEMS $F4$ AND $F5$

		Single Node configuration				Multiple Nodes configuration			
P.M.	npr	Med.	Std.	SpeedUp	Effic.	Med.	Std.	SpeedUp	Effic.
Problem $F4$									
PM1	2	2.86	0.01	3.49	1.74	2.86	0.02	3.49	1.74
	4	1.96	0.01	5.09	1.27	1.82	0.04	5.48	1.37
	8	1.85	0.02	5.39	0.67	1.55	0.05	6.43	0.80
	12	2.42	0.04	4.12	0.34	1.71	0.08	5.83	0.49
PM2a	4	2.01	0.02	4.96	1.24	1.89	0.05	5.28	1.32
	8	1.87	0.04	5.33	0.67	1.60	0.05	6.23	0.78
	12	2.41	0.04	4.14	0.34	1.76	0.03	5.66	0.47
PM2b	1	3.30	0.08	3.02	3.02	3.28	0.08	3.04	3.04
	2	1.92	0.02	5.19	2.60	1.81	0.03	5.51	2.76
	4	1.19	0.02	8.38	2.10	1.09	0.02	9.15	2.29
	8	0.92	0.03	10.84	1.36	0.76	0.01	13.12	1.64
	12	0.89	0.02	11.20	0.93	0.69	0.01	14.45	1.20
Problem $F5$									
PM1	2	2.87	0.05	3.48	1.74	2.88	0.02	3.47	1.74
	4	1.97	0.04	5.07	1.27	1.83	0.07	5.46	1.36
	8	1.87	0.95	5.34	0.67	1.56	0.02	6.40	0.80
	12	2.42	0.09	4.13	0.34	1.72	0.04	5.81	0.48
PM2a	4	2.00	0.02	5.00	1.25	1.89	0.05	5.29	1.32
	8	1.88	0.02	5.31	0.66	1.61	0.06	6.20	0.78
	12	2.42	0.13	4.13	0.34	1.75	0.06	5.71	0.48
PM2b	1	3.26	0.08	3.06	3.06	3.26	0.09	3.06	3.06
	2	1.92	0.03	5.20	2.60	1.82	0.12	5.49	2.74
	4	1.20	0.09	8.33	2.08	1.09	0.02	9.17	2.29
	8	0.92	0.04	10.86	1.36	0.76	0.02	13.14	1.64
	12	0.89	0.03	11.22	0.94	0.69	0.01	14.48	1.21

reported in [2]. In a brief analysis, we can expect that less distributed configurations achieve better performances, due to lower communication costs. However, we suggest the low-level computations as the most probable reason for the super-linear speedups and also for the lower efficiencies recorded by our experiments in single computing nodes. For example, to add computing nodes to an architecture also implies the increment of cache and RAM memory, which can be used to improve the performance of many simple calculations in an iterative algorithm as a population-based metaheuristic. If these improvements compensate the increase in communication costs, the most distributed experiment performs better than the less distributed one. This also is one of the factors that can be used to justify the super-linear speedups in metaheuristics.

In the literature, many factors have lead to the reporting of results of this nature for parallel metaheuristics. Studies approaching this issue constitutes a separated research theme and are available in [31].

However, despite all this, our results corroborate the efficiency of our parallel models to solve Stackelberg-Nash problems both for single multi-core machine environments as for highly-distributed cluster architectures. The models offered in this work are scalable and can be executed without adaptations for different computing resources.

VIII. CONCLUSIONS

In this paper, we suggested and studied three new parallel Differential Evolution models based on distributed- and shared-memory to enable more efficient handling of Stackelberg-Nash BLPs.

Our results endorsed the important possibilities for Bilevel Optimization that arise from the association of the high-computing performance resources currently available with the natural parallelism of population-based metaheuristics. We demonstrated that the Stackelberg-Nash equilibrium can be obtained through parallel algorithms, which can compute concurrently or not the lower-level variables that compose an inter-dependent followers BLP.

Our suggested parallel models revealed to be satisfactorily scalable. The communication costs were not relevant enough

to reduce the performances of our implementations when distributed to a high number of separated computing nodes. All models achieved relevant speedups and efficiencies for both less and more distributed configurations.

However, reporting a maximum speedup of 14.82, the P.M.2b improved its speedups whenever the number of processing instances was increased. This approach revealed as a good strategy the implementation of a hybrid strategy, which partitions the upper-level population in a distributed-memory environment and evolves concurrently the lower-level followers in a shared-memory environment. Avoiding communication costs and handling in parallel both BLP levels, this model obtained the best efficiencies even considering the sum of its distributed- and shared-memory processing instances to perform comparisons with the other parallel models.

Thus, this work contributes to the literature providing the needed first parallel models to solve Stackelberg-Nash BLPs via population-based metaheuristics. As the treatment of this class of problems naturally implies in considerably computing costs, the cases handled in this work already demand sufficient processing time to compose the first set of tests that endorses the usefulness of our parallel models. We hope that these tools can bring a new range of possibilities for the field, enabling the tackling of increasingly complex and larger problems with inter-dependent followers.

Also, the discussions exposed here can guide future researches indicating the most promising approaches to improve the efficiency in the treatment of this type of problems. In particular, our hybrid distributed- and shared-memory model constitutes a new scalable and notably efficient alternative to enable the study of Stackelberg-Nash BLPs of more varied magnitudes and complexities, for different computing architectures. Thus, having provided the parallel models, our next steps will include the development of a set of scalable Stackelberg-Nash BLPs, which will enable tests involving cases of a higher dimensionality not common in the literature of this class of problems.

REFERENCES

[1] E. Talbi, *Metaheuristics: from design to implementation*, vol. 74, John Wiley & Sons. 2009.

[2] T. T. Magalhães and H. J. C. Barbosa. "Differential Evolution Algorithms for Solving Bilevel Optimization Problems using Computational Clusters." *ACSE 6th Annual Conf. on Computational Science & Computational Intelligence (CSCI)*, 2019.

[3] C Floudas, A. Christodoulos, and P. M. Pardalos. *State of the art in global optimization: computational methods and applications*, vol. 7, Springer Science & Business Media. 2013.

[4] H. V. Stackelberg. "Marktform und Gleichgewicht". Springer-Verlag, Berlin, 1934.

[5] L. Brotcorne, M. Labbé, P. Marcotte, and G. Savard. A bilevel model for toll optimization on a multicommodity transportation network. Transportation Science, 35(4):345–358, 2001.

[6] A. Sinha, P. Malo, A. Frantsev, and K. Deb. Multi-objective Stackelberg game between a regulating authority and a mining company: A case study in environmental economics. In Evolutionary Computation (CEC), 2013 IEEE Congress on, pages 478–485. IEEE, 2013.

[7] G. Whittaker. In search of efficient networks using bilevel evolutionary optimization. 2016.

[8] P. Pharkya, A. P. Burgard, and C. D. Maranas. Exploring the overproduction of amino acids using the bilevel optimization framework optknock. Biotechnology and Bioengineering, 84(7):887–899, 2003.

[9] J. F. Bard. Practical bilevel optimization: algorithms and applications, volume 30. Springer Science & Business Media, 2013.

[10] A. Sinha, P. Malo, and K. Deb. A review on bilevel optimization: from classical to evolutionary approaches and applications. IEEE Transactions on Evolutionary Computation, 22(2):276–295, 2018

[11] T. T. Magalhães, E. Krempser and H. J. C. Barbosa. "Parallel Models of Differential Evolution for Bilevel Programming". Proceedings of the 2018 International Conference on Artificial Intelligence. 2018

[12] R. Mathieu, L. Pittard and G. Anandalingam, "Genetic algorithm based approach to bi-level linear programming," *RAIRO - Operations Research - Recherche Operationnelle*, vol. 28, no. 1, pp. 1–21. 1994.

[13] V. Oduguwa and R. Roy, "Bi-level optimisation using genetic algorithm," *IEEE Computer Society 2002 IEEE International Conference on Artificial Intelligence Systems*, pp. 322–327. 2002.

[14] X. Li, P. Tian, and X. Min. "A hierarchical particle swarm optimization for solving bilevel programming problems." *In Springer Berlin Heidelberg Artificial Intelligence and Soft Computing - ICAISC 2006*, vol. 4029, pp. 1169–1178. 2006.

[15] A. Sinha, P. Malo, A. Frantsev, and K. Deb. "Finding optimal strategies in a multi-period multi-leader-follower Stackelberg game using an evolutionary algorithm." *Computers & Operations Research*, cap. 41, pp. 374–385. 2014.

[16] J. S. Angelo, E. Krempser, and H. J. C. Barbosa. "Differential evolution assisted by a surrogate model for bilevel programming problems." *IEEE Evolutionary Computation Congress (CEC)* pp. 1784–1791. 2014.

[17] A. Sinha P. Malo, and K. Deb. "Evolutionary algorithm for bilevel optimization using approximations of the lower level optimal solution mapping." *European Journal of Operational Research*. 2016

[18] R. Storn and K. V. Price, "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces." *Journal of Global Optimization*, vol. 11, pp. 341–359. Dec. 1997.

[19] G. Shanmugavelayutham, C. Jeyakumar. "Convergence analysis of differential evolution variants on unconstrained global optimization functions." *arXiv preprint arXiv:1105.1901*. 2011.

[20] J. S. Angelo, E. Krempser and H. J. C. Barbosa. "Differential evolution for bilevel programming." *IEEE Congress on Evolutionary Computation (CEC)*, pp. 470–477, Jun. 2013

[21] J. S. Angelo and H. J. C. Barbosa. "Differential evolution to find Stackelberg-Nash equilibrium in bilevel problems with multiple followers." *IEEE Congress on Evolutionary Computation (CEC)*, pp. 1675–1682, 2015.

[22] E. D. Amato, E. Daniele, L. Mallozzi, G. Petrone, and S. Tancredi. "A hierarchical multimodal hybrid Stackelberg-Nash GA for a leader with multiple followers game," *in Dynamics of Information Systems: Mathematical Foundations*. Springer, vol. 20, pp. 267–280. 2012.

[23] M. Sefrioui and J. Periaux. "Nash genetic algorithms: examples and applications." *in Proceedings of the 2000 Congress on Evolutionary Computation*, vol. 1, pp. 509–516. 2000.

[24] B. Liu. "Stackelberg-Nash equilibrium for multilevel programming with multiple followers using genetic algorithms." *Computers & Mathematics with Applications*, vol. 36, no. 7, pp. 79 – 89. 1998.

[25] J. F. Wang and J. Periaux. "Multi-point optimization using GAs and Nash/Stackelberg games for high lift multi-airfoil design in aerodynamics." *in Proceedings of the Congress on Evolutionary Computation*, vol. 1, pp. 552–559. 2001.

[26] N. H. Awad, M. Z. Ali, J. J. Liang, B. Y. Qu, and P. N. Suganthan. "Problem definitions and evaluation criteria for the CEC 2017 special session and competition on single objective bound constrained real-parameter numerical optimization." *In Technical Report*. NTU, Singapore. 2016.

[27] K. Deb. "An efficient constraint handling method for genetic algorithms," *Computer methods in applied mechanics and engineering*, vol. 186, pp. 311–338. 2000.

[28] Q. Wang, F. Yang, S. Wang, and L. Yi-Hsin. "Bilevel programs with multiple followers." *Journal of Systems Science and Complexity*, vol. 13, no. 3, pp. 265. 2000.

[29] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: portable parallel programming with the message-passing interface*. MIT press. 1999.

[30] R. Chandra, L. Dagum, D. Kohr, R. Menon, D. Maydan, and J. McDonald. *Parallel programming in OpenMP*. Morgan Kaufmann. 2001.

[31] E. Alba. "Parallel evolutionary algorithms can achieve super-linear performance." *Information Processing Letters* no. 82.1, pp. 7–13. 2002.

[32] A. Koh. "An evolutionary algorithm based on Nash dominance for equilibrium problems with equilibrium constraints." *Applied soft computing* vol. 12 no. 1 pp. 161–173. 2012