CoBRA: A cooperative coevolutionary algorithm for bi-level optimization

François Legillon, Arnaud Liefooghe, El-Ghazali Talbi

CoBRA: A Cooperative Coevolutionary Algorithm for Bi-level Optimization

François Legillon
Université Lille 1, LIFL – CNRS
INRIA Lille-Nord Europe
Villeneuve d'Ascq, France
Email: francois.legillon@inria.fr

Arnaud Liefooghe
Université Lille 1, LIFL – CNRS
INRIA Lille-Nord Europe
Villeneuve d'Ascq, France
Email: arnaud.liefooghe@univ-lille1.fr

El-Ghazali Talbi
Université Lille 1, LIFL – CNRS
INRIA Lille-Nord Europe
Villeneuve d'Ascq, France
Email: talbi@lifl.fr

***Abstract*—This article presents CoBRA, a new evolutionary algorithm, based on a coevolutionary scheme, to solve bi-level optimization problems. It handles population-based algorithms on each level, each one cooperating with the other to provide solutions for the overall problem. Moreover, in order to evaluate the relevance of CoBRA against more classical approaches, a new performance assessment methodology, based on rationality, is introduced. An experimental analysis is conducted on a bi-level distribution planning problem, where multiple manufacturing plants deliver items to depots, and where a distribution company controls several depots and distributes items from depots to retailers. The experimental results reveal significant enhancements, particularly over the lower level, with respect to a more classical approach based on a hierarchical scheme.**

***Index Terms*—Evolutionary computation, Algorithm design and analysis, bi-level optimization, vehicle routing**

I. Introduction

Bi-level optimization problems allow to model a large number of real-life applications, with a hierarchical structure between two decision makers. It includes companies which have to face a legislator and security constraints [1], companies trying to predict consumer reaction [2], or a supply chain where a company has to predict its supplier reaction to determine the real cost of its decision [3].

Evolutionary algorithms (EA) are a class of approximate algorithms focusing on finding good-quality solutions for large-size and complex problems, in a reasonable amount of time [4]. While most of the existing literature about bi-level optimization focuses on small-size linear problems (see for example [5], [6]), many real-life applications involve large-size instances and complex NP-hard problems, justifying the use of EAs. Existing EAs for bi-level optimization can be divided into two main classes. On the one hand, *hierarchical algorithms* try to solve the two levels sequentially, improving solutions on each level to get a good overall solution on both levels. Such algorithms include the repairing algorithm [7], which considers the lower-level problem as a constraint and solve it during the evaluation step, or the constructing algorithm [8] which applies two population-based algorithms on a population, one for each level, sequentially until a stopping criterion is satisfied. On the other hand, *coevolutionary algorithms* maintain two populations, one for each level, and try to improve them separately, while periodically exchanging information to keep an overall view on the problem, like in [9]. In cooperative coevolution, different sub-populations evolve a part of the decision variables, and complete solutions are built by means of a cooperative exchange of individuals from sub-populations [10].

This article focuses on a coevolutionary approach. Sub-problems involved in bi-level optimization can be tackled by EAs. Finding a good way to combine two EAs in order to solve a bi-level optimization problem as a whole would give a general methodology for bi-level optimization. First, we introduce a new algorithm, the *Co*evolutionary *B*i-level method using *R*epeated *A*lgorithms (CoBRA). This coevolutionary algorithm is able to face general bi-level optimization problems, possibly involving complex large-size search spaces. Next, we introduce a new measure for performance assessment, the rationality, able to more fully grasp the bi-level aspect of the problems than the Pareto efficiency. Rationality is based on the proximity from the optimum of the lower-level variables while keeping the corresponding upper-level variables fixed. At last, to evaluate the performance of CoBRA against classical hierarchical approaches, we give an experimental analysis on a bi-level transportation problem involving a supply chain, the bi-level multiple depot vehicle problem introduced in [3]. This analysis includes the problem modeling, the instantiation of CoBRA to solve it, and the study of the results with respect to fitness values and rationality metrics.

The paper is organized as follows. Section II gives the necessary background on bi-level optimization. Section III presents the new coevolutionary algorithm proposed in the paper for bi-level optimization, namely CoBRA. In Section IV, we discuss the issue of assessing the performance of approximate algorithms in bi-level optimization. The bi-level transportation problem under investigation in this paper is presented in Section V. The experimental analysis of CoBRA is given Section VI. At last, the final section concludes the paper and gives directions for further research.

II. Bi-level Optimization

In this section we introduce a general bi-level optimization problem, and give a quick overview of state-of-the-art EAs for bi-level optimization.

A. General Principles of Bi-level Optimization

Bi-level optimization problems may be defined by the tuple (S, F, f) where S represents the set of feasible solutions (*i.e.* the search space), F the objective function of the upper level, and f the objective function of the lower level. For any $x \in S$ we separate the upper-level variables and the lower-level variables, respectively in x_u and x_l.

We define, for every x_u fixed, the set of rational reactions $R(x_u)$ as the set of x_l optimal in f.

$$R(S, f, x_u) = \begin{cases} \min_{x_l} f(x = (x_u, x_l)) \\ \text{s.t. } x \in S \end{cases}$$

Solving the bi-level optimization problem consists in finding the solution $x \in S$ which is optimal with respect to f for x_u fixed and, respecting this constraint, optimal with respect to F.

$$BP(\mathcal{S}, \mathrm{F}, \mathrm{f}) = \begin{cases} \min \mathrm{F}(x) \\ x \in \mathcal{S} \\ \text{s.t.} \begin{cases} x = (x_u, x_l) \\ x_l \in R(\mathcal{S}, \mathrm{f}, x_u) \end{cases} \end{cases}$$

Those problems induce a hierarchy between two decision makers:

- The *leader*, who chooses the upper part of the decision variables, x_u, and tries to optimize F.
- The *follower*, who chooses the lower part of the decision variables, x_l, and tries to optimize f.

The leader decides first. Then, the follower, knowing the leader decision, has to decide, in the view of optimizing its own objective function f, without regarding the upper-level objective function F. To optimize his choice, the leader then has to predict the follower *reaction*. This hierarchy can conduct to a higher complexity than both sub-problems. For instance, a NP-hard bi-level optimization problem can be obtained from two linear problems [11].

This definition of bi-level optimization corresponds to the optimistic case, where the leader can "choose" the (x_u, x_l) couple in the set of $(x_u, x_l) \in S$ where $x_l \in R(x_u)$: the reaction has to be optimal, but if several reactions are optima (*i.e.* $|R(x_u)| > 1$) the leader has the last word. There exists a pessimistic case [12] which is not treated in this paper, where x_l is chosen as the leader worst-case scenario in the set of rational responses.

B. Evolutionary Approaches for Bi-level Optimization

EAs are approximate algorithms which allow to tackle large-size problem instances by delivering satisfactory solutions in reasonable time [4]. Due to their complexity, most bi-level optimization problems are tackled by approaches which involve a model reformulation masking the bi-level aspect of the problem (see [5], [6], [13], [14]), or which are based on heuristics.

EAs for bi-level optimization generally involve a hierarchical scheme, where a level is firstly treated with a heuristic, giving good-quality solutions. Then, starting with those solution, the algorithm tries to solve the other problem. We can distinguish two main families in this hierarchical scheme:

- Constructive algorithms, that search for good-quality solutions for the lower level, and then try to improve the upper-level fitness value, without degrading the lower level [8].
- Repairing algorithms, that search for good upper-level fitness value, and then try to improve the lower level in order to obtain rational solutions [7].

In this paper, we focus on coevolutionary approaches, a subgroup of meta-heuristics extending the evolutionary scheme. Coevolutionary algorithms consists in associating several EAs and applying variations operators, such as mutation and crossover, to distinct populations. A coevolution operator is then regularly applied between sub-populations to keep a global view on the whole problem.

For instance, Oduguwa and Roy proposed BiGA [9], a coevolutionary algorithm for bi-level optimization. BiGA starts by initializing two distinct sub-populations using a heuristic, pop_u for the upper level and pop_l for the lower level. Then the upper-level part of the solutions is copied from pop_u to pop_l. During a given number of generations, a selection process, based on the respective level fitness values, is applied to both sub-populations, followed by a variation step. Then, sub-populations are evaluated, sorted, and coevolved, by copying the upper (respectively lower) variables to the lower (respectively upper) sub-population. At last, an archiving process occurs to keep the overall best solution, before looping again to the selection step.

III. CoBRA, a cooperative Coevolutionary Algorithm for Bi-level Optimization

In this section we introduce CoBRA, a new EA to tackle bi-level optimization problems. We give here the general parts and principles of CoBRA. The reader is referred to Section VI for a specific implementation of CoBRA with all the problem-specific algorithmic components included.

A. CoBRA Principles

Most of the literature focuses on linear bi-level optimization problems (*i.e.* formed with two linear sub-problems) or lower-level problems that can be solved to optimality in a reasonable amount of time. They use this property to discard the bi-level aspect of the problem. This article tries to define a more general methodology to solve bi-level optimization problems, with a hard lower-level problem. The problem complexity leads us to consider the use of heuristics in order to obtain good-quality solutions.

CoBRA is a *co*evolutionary *b*ilevel method using *r*epeated *a*lgorithms. Extending Oduguwa and Roy's BiGA [9], it is a coevolutionary algorithm consisting in improving incrementally two different sub-populations, each one corresponding to one level, and periodically exchanging information with the other.

Data: initial population pop
$pop_u \leftarrow_{copie}$ pop;
$pop_l \leftarrow_{copie}$ pop;
while *Stopping criterion not met* **do**
  upper evolution (pop_u) and lower evolution (pop_l);
  upper archiving (pop_u) and lower archiving (pop_l);
  selection (pop_u) and selection (pop_l);
  coevolution(pop_u, pop_l);
  adding from upper archive (pop_u);
  adding from lower archive (pop_l);
end
return *lower archive*

Algorithm 1: Pseudo-code of CoBRA

CoBRA uses coevolution to associate two distinct EAs. Each of those algorithms are supposed to be, in the CoBRA scheme, fast methods that offer good-quality solutions for one of the two levels. Using an algorithm for each level, CoBRA seeks to produce a good-quality solution in terms of bi-level optimization.

B. CoBRA Components

In order to instantiate CoBRA for a particular bi-level optimization problem, generic and problem-specific components have to be defined. Generic components, which can correspond to both sub-problems, consist in choosing the following components:

- Two level-specific EAs, one for each level, that does not change the other level part of the solution.
- A coevolution strategy to decide how populations should exchange information.
- An archiving strategy to record the best solutions on every level, and to prevent the coevolution to change the sub-populations completely in a single generation.
- A stopping criterion to decide when the algorithm should stop.
- Level-specific selection operators, which remove bad-quality solutions from populations before the coevolution step.

Problem-specific components still have to be designed in order to use CoBRA:

- Initialization operators, generally heuristics, which create a base population to begin the search process.
- Variation operators, which are level-specific, which are used by the EAs.
- Evaluation operators, corresponding to the f and F functions from the bi-level optimization model.

Figure 1 illustrates the outline of CoBRA.

C. General Algorithm

CoBRA is a coevolutionary algorithm using a different population, and a different archive for each level (Algo. 1). At each iteration, we apply the EAs, we archive the best solutions obtained, then we apply a selection operator to keep a constant size to the archive and populations. The final iteration step is then to coevolve both sub-populations. Once the stopping criterion is met, CoBRA returns the lower-level archive.

Data: Populations $upPop$ and $lowPop$ of same size, op coevolution operator
Shuffle $upPop$;
foreach i *from 0 to size(upPop)* **do**
  $op(upPop[i], lowPop[i])$;

Algorithm 2: Pseudo-code of the coevolution step

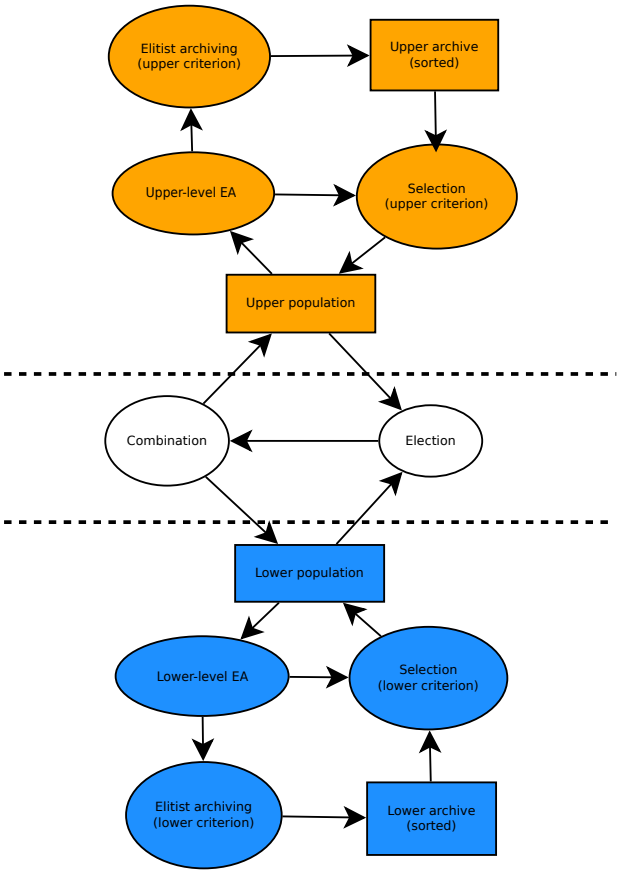

Fig. 1: CoBRA outline.

CoBRA involves several differences with BiGA:

1) The main difference is that CoBRA applies a complete algorithm, possibly iterating a certain number of generations, over each main algorithm iteration, instead of just applying variation operators. The evaluation process occurs during those improvement algorithms.
2) The coevolution process is not necessarily elitist: default coevolution strategy (Algo. 2) randomly coevolves solutions with each other.
3) The selection and the archiving processes take place right after the improvement.

The structure of CoBRA makes it easy to use for bi-level optimization problems where both the lower-level and the upper-level parts are hard to solve to optimality. Using those

F	a	b
d	0	1000
e	1	∞
f	300	∞

f	a	b
d	100	99
e	1001	∞
f	99	∞

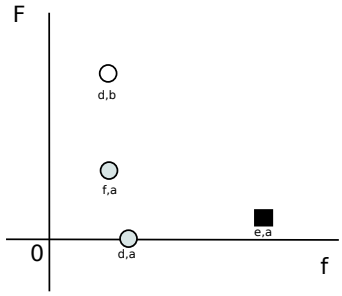

Fig. 2: Example of lower-level and upper-level objective functions whose optimal solution is dominated in terms of Pareto dominance.

heuristics, it allows to tackle the bi-level aspect of the problem as a whole, without much additional work.

IV. Performance Assessment and Bi-level Optimization

In this section, we introduce two new metrics for assessing the performance of heuristics on solving bi-level optimization problems.

A. Motivations

Being a problem with two different objective functions, a natural approach to tackle bi-level optimization problems would be to use a Pareto-based multi-criteria approach. However, bi-level optimization problems have a different structure. Indeed, A Pareto optimal solution could be of bad quality in terms of bi-level optimization.

Bi-level optimization aims at identifying solutions in the form (x_u, x_l) which give good upper-level objective values, while being near the optimum regarding the lower-level objective for a x_u fixed. This leads to the existence of good-quality solutions not being on the Pareto frontier, and solutions on the Pareto frontier not necessarily being good-quality solutions. Fig. 2 gives an example of objective functions giving a bi-level solution corresponding to a dominated solution in the Pareto sense. F and f are respectively the upper and the lower-level objective functions, to be minimized. The leader chooses in $\{$d,e,f$\}$ and the follower in $\{$a,b$\}$. The Pareto frontier would be made of $\{$(d,a),(f,a)$\}$ while the bi-level solution is (e,a). In conclusion, following a classical Pareto approach in this example would lead the decision maker to the wrong solution. A good measure for the quality of a solution needs to take into account the bi-level structure of the problem.

We introduce the notion of *rationality* which corresponds to the difficulty to improve a solution (x_u, x_l), while keeping x_u fixed, according to the lower-level objective function. A rational solution is a solution where the follower reaction is rational, seeking for the optimality of its own objective function. If a solution we consider can easily be improved by the follower in terms of lower-level criteria, it is associated with a bad rationality measure.

Data: AlgoLow, pop, ni number of iterations
$counter \leftarrow 0$;
foreach *gen from 1 to* ni **do**
  $neopop \leftarrow pop$;
  $found \leftarrow false$;
  AlgoLow($neopop$);
  foreach x *in* $neopop$ **do**
    if *(not* $found$*) and (*x *is better than an element of pop)* **then**
      $counter$++;
      $found \leftarrow false$;
    end
  end
end
return $counter/ni$

Algorithm 3: Direct rationality test

B. Rationality

1) Direct Rationality: The *direct rationality* measure corresponds to the difficulty of improving a solution without regarding the actual gap of improvement: we simply consider the "improvability". To evaluate it for a population, we apply a "well-performing" lower-level algorithm for a given number of times, and we count how many times the algorithm actually improves the solution (Algo. 3).

2) Weighted Rationality: The *weighted rationality* is another rationality measure working on the same principle as the direct rationality with the difference that, instead of counting how many times the algorithm was able to improve the solution, we also consider by how much it was improved. Being able to improve a fitness value by 0.001 or by 1000 does not give the same result in terms of rationality, whereas the direct approach would consider both as the same (Algo. 4).

Data: AlgoLow, pop, ni number of iterations
$ratio \leftarrow 0$;
foreach *gen from 1 to* ni **do**
  $neopop \leftarrow pop$;
  AlgoLow($neopop$);
  $ratio$=$ratio$+$(f(best(neopop))/f(best(pop))/ni$;
end
return $ratio$

Algorithm 4: Weighted rationality test

Let us note that those methods are not absolute, in the sense that we have to compare the algorithm using another algorithm, thus introducing a bias. Those measures actually compare the capacity of a heuristic to use their given level-specific EAs, but do not actually compare the overall capacity to tackle the problem. To this end, we have to ensure that none of the tested algorithms is biased toward the algorithm used by the rationality evaluation.

V. A Bi-level Multi-depot Vehicle Routing Problem

In this section we define a bi-level transportation problem, involving two different companies in a supply chain. The leader transports goods from depots to retailers, answering to the retailers demand. The follower manages plants producing goods for the leader. The leader starts by deciding which depots should deliver goods, then the follower decides how to manufacture the goods. Both decisions influencing the overall cost of solutions.

This problem, introduced by Calvete and Galé [3], consists of a bi-level optimization problem where the leader controls a fleet of vehicles to deliver items from several depots to retailers, on the same principle as the classical multi-depot vehicle routing problem (MDVRP). The follower controls a set of plants, and has to produce the items and deliver them to the depots according to the demand of the retailers it serves, thus corresponding to a flow problem. The leader tries to minimize the total distance of his routes and the buying cost of the resources (depending on the lower-level decision). The follower minimizes the production cost and the distance traveled by the produced goods. The follower has to directly transport from plants to depots.

A. Problem Description

Let K, L, R and S denote the sets of plants, of depots, of retailers and of vehicles, respectively. Let E be the edge set between retailers and depots, b_r the demand of retailer r, $\mathrm{c}^a_{i,j}$ the cost of transporting goods from depots or retailers i to j for the leader, $\mathrm{c}^b_{k,l}$ the cost to buy and unload a unit produced in plant k into depot l for the leader, and $\mathrm{c}^c_{k,l}$ the operational cost for plant k to produce and deliver to depot l for the follower.

The upper-level objective function is to minimize the sum of deliver costs from depots to retailers and buying from plants.

$$\mathrm{F}(x,y) = \sum_{s \in S} \sum_{(i,j) \in E} c^a_{i,j} x^s_{i,j} + \sum_{k \in K} \sum_{l \in L} c^b_{k,l} y_{k,l}$$

with x the leader variables representing the routes chosen to deliver retailers, and y the follower variables representing the affectation of plants to depots.

Then, the lower-level objective function is to minimize the sum of costs of producing items in plants and delivering it to depots.

$$\mathrm{f}(x,y) = \sum_{k \in K} \sum_{l \in L} c^c_{k,l} y_{k,l}$$

The leader and follower follow a hierarchical order, where the leader choose routes, creating a demand for the depots corresponding to the retailers to be delivered, and where the follower has to respond to this new demand by associating a part of his plant production to depots.

$$\sum_{k \in K} y_{k,l} \geq \sum_{s \in S_l} \sum_{r \in R_s} b_r, \forall l \in L$$

Several other VRP-related constraints are omitted to improve readability. See [3] for more details about the problem.

TABLE I: Description of the S_1 instances, R corresponding to the number of routes by depot.

Instance	Depot	R	Plants (S_1)	Retailer
bipr01	4	1	4	48
bipr02	4	2	4	96
bipr03	4	3	4	144
bipr04	4	4	4	192
bipr05	4	5	4	340
bipr06	4	6	4	288
bipr07	6	1	6	72
bipr08	6	2	6	144
bipr09	6	3	6	216
bipr10	6	4	6	288

B. Problem Instances

A set of instances[1] was generated to experiment the CoBRA efficiency. S_1 consists of instances created from MDVRP instances following the *modus operandi* described in [3]. We add as many plants as there are depots randomly located on the map. Then, we set their maximal production to ensure that the instance is feasible. c^b and c^c follows the approach described in [3]. Set S_1 contains 10 instances created from the 10 instances provided by Cordeau [15]. Those instance parameters are described in Table I.

VI. Experimental Analysis

In order to evaluate the relevance of CoBRA for bi-level optimization, we conduct in this section an experimental analysis against a classical repairing algorithm. The approach considers the lower-level optimality condition as a constraint, and simply tries to find the best upper-level variables while "repairing" the lower level during the evaluation step. The goal of this section is not to compare a number of coevolutionary algorithms, but rather to show that such approaches are more appropriate than a repairing algorithm for hard-to-solve bi-level optimization problems.

A. Experimental Design

We conduct an experimental analysis by applying both algorithms on the bi-level multi-depot vehicle routing problem (BiMDVRP). Each algorithm is run 30 times with different seed values.

Both algorithms use the same components (*i.e.* the lower-level and upper-level EAs, the stopping criterion, the variation and initialization operators). The repairing algorithm works as follows. It does not use any archiving or coevolution operator, and a different evaluation operator which applies an lower-level based EA before evaluating the solution. Once the stopping criterion is met, we evaluate the population with respect to the following three criteria:

- The upper-level fitness value,
- The direct rationality,
- The weighted rationality.

We report average values over the 30 execution for all metrics.

[1]Benchmark files are publicly available on the paradiseo website in the problems section at the following URL: http://paradiseo.gforge.inria.fr/index.php?n=Problems.Problems.

B. CoBRA for the BiMDVRP

In order to fit CoBRA to the BiMDVRP, several problem-specific components have to be chosen.

1) Solution Representation: With the aim of developing an EA, a solution representation is necessary. Using a generic bi-level representation, we had to decide a representation for each level. For the upper level, we use a permutation: a number is attributed to every retailer and every route (each route being associated with a depot). The route numbers in the permutation determine the routes start, and every retailer between route numbers represent the actual route (Fig. 3). This representation facilitates the solution integrity, and suppresses the need to check the number of routes and the "one visit per retailer" constraint. We use for the lower-level problem a more classical matrix of real numbers M, M_b^a representing the ratio of production sent from a to b. The quantity effectively sent is scaled down at the evaluation step if the sum of a column are over 1, and rounded down if it is not an integer. This indirect representation permits to use of classical algorithms without much adaptation work.

2) Upper-level Problem-related Components: For the MDVRP upper-level problem, we use a combination of three variation operators:

- RBX [16] is a crossover operator copying routes from a parent, and then completing the offspring with routes from the other parents by removing visited retailers.
- SBX [16] is a crossover operator creating a new route, by taking half of a route starting from a single depot in each parent, keeping the order of each half, and then completing the offspring with the other routes and removing visited retailers.
- Or-opt [17] is a mutation operator taking several retailers from a route and putting it in another. This operator changes the number of routes which neither of the SBX and RBX can do.

Operators are applied to uniformly-chosen solutions from the population.

3) Lower-level Problem-related Components: For the lower-level problem, we use a combination of two operators:

- Umut [18] is a mutation operator that adds a parametrized real value $r_{lmut} \in [-0.5, 0.5]$ to each element of the solution matrix with a p_{lmut} probability.
- UXover [19] is a crossover operator choosing elements uniformly for each parent solution matrix and putting it in the offspring population.

4) Stopping Condition: The algorithm uses three stopping criteria, one for each EA and one for the overall algorithm. Both EAs use a generational stopping criterion which continues for a fixed number p_g of generations. The overall algorithm uses a lexicographic continuator which continues until no better solution is found for a fixed parameter p_l of generations, by using a lexicographic comparator (*i.e.* by comparing sequentially the objective values on each level starting with the upper level).

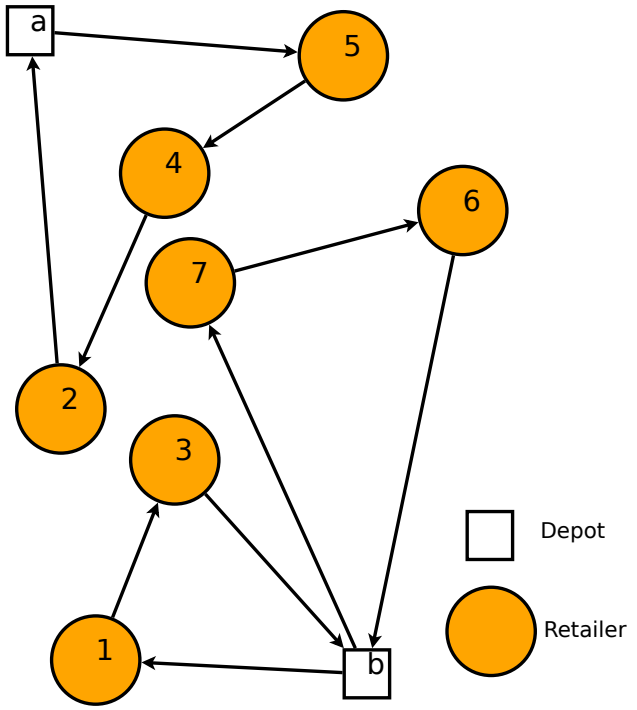

Fig. 3: Example of a VRP with 7 retailers, 2 depots, and 2 routes *per* depot, from the permutation $[5, 4, 2, 9, 7, 6, 10, 1, 3, 8]$. Squares are for depots {a,b}, circles for retailers {1,2,3,4,5,6,7}.

5) Selection Operators: The algorithm uses three selection operators to choose which solution to keep from a generation to the next one, one for each EA, and one for the overall algorithm. We use on both EAs a deterministic tournament, which randomly selects two solutions from the population and keeps the best one. For the overall algorithm we use a survive-and-die replacement politic, which keeps a parametrized proportion of the best solutions n_{sad} from the last generation, and apply a deterministic tournament on the remaining part of the population in order to generate the population for the next generation.

6) Archiving Strategy: The algorithm uses archives to keep record of the best solutions found over all generations. We use a straight-forward archive that keeps the n best-found solutions according the the lower-level fitness value.

7) Numerical Parameters: Following preliminary experiments, the remaining parameters have been set as follows.

- n: the populations size, set to 100,
- r_{lmut}: the uniform mutation adding parameter, set to 0.5,
- p_{lmut}: the uniform mutation probability parameter, set to 0.1,
- p_g: the number of generations each EA is performed, set to 10,
- p_l: the number of generations CoBRA continues without improvement, set to 100,
- n_{sad} the proportion of best solutions that are kept from the previous generation, set to 0.8.

TABLE II: Average upper-level fitness value, direct rationality value and weighted rationality value obtained by CoBRA and the repairing algorithms (Repair) for BiMDVRP instances from S_1. A statistically better result is given in bold.

	upper level fitness		lower level			
			direct rationality		weighted rationality	
instance	CoBRA	Repair	CoBRA	Repair	CoBRA	Repair
bipr01	1830	1847	**0.6**	7.5	**11.2**	3326.8
bipr02	4475	**3331**	**1.3**	5.4	**131.4**	3917.7
bipr03	8723	**5778**	**2.7**	23.5	**264.0**	5293.3
bipr04	11024	**7260**	**1.9**	19.9	**193.4**	5435.4
bipr05	13069	**8490**	**0.6**	5.4	**66.7**	3131.8
bipr06	17229	**11559**	**0.7**	5.9	**68.3**	2160.3
bipr07	3320	**2917**	**0.9**	1.5	**92.5**	150.7
bipr08	8500	**5344**	**2.2**	22.9	**212.8**	233.1
bipr09	13435	**8298**	**2.0**	22.2	**195.4**	235.2
bipr10	18752	**12366**	**0.6**	13.5	**65.9**	133.2

C. Experimental Results

Table II shows numerical results for CoBRA and the repairing algorithm on instances from S_1. The average upper-level fitness values as well as the direct and weighted rationality metric values are reported. A Wilcoxon signed rank-sum test was used to determine if there is a significant difference between both algorithms in terms of fitness and rationality. Clearly, CoBRA obtains a better score in terms of rationality on all the instances. For both algorithms, rationality is not related to the instance size. However, the repairing algorithm is doing better for the upper-level fitness value.

CoBRA has a significant advantage in terms of rationality for all the runs we performed, while having a worse upper-level fitness value. Rationality indicates the quality of the reaction predicted by the algorithm. A bad prediction is likely to lead to a bad solution: once applied to a real-life situation, the follower will have greater chances to chose a better reaction for his own objective function, then degrading the solution quality for the leader. Since CoBRA has a better rationality, it allows to better predict the outcome of the decisions. Thus, we can conclude that CoBRA is more adapted to the bi-level aspect of the problem.

An explanation why the hierarchical algorithm does not select the more rational response would be that, once an irrational solution $x = (x_u, x_l)$ is obtained, through a badly done reparation, which gives a better upper-level fitness value than the more rational response $x' = (x_u, x'_l)$, the overall algorithm will have a tendency to discard x' and keep x. We can conclude that the repairing approach needs either a very efficient lower-level heuristic, an exact lower-level algorithm, or some properties over the problem (such as a strong correlation between the two levels) in order to be able to produce rational responses. This is the reason why the coevolution scheme allows CoBRA to obtain a better rationality. The number of evaluations required by CoBRA stays stable whatever the instance size, while the repairing algorithm requires a number of evaluations about 1000 times higher than CoBRA, and which increases with the problem size. We can conclude that the coevolutionary approach can give a significant enhancement for this problem, without significant loss in terms of computational time.

VII. Conclusions and Future Works

In this paper, we described *CoBRA*, a new general methodology to solve bi-level optimization problems. We introduced the concept of *rationality* for bi-level optimization problems and new metrics to compare the performance of heuristics for such purpose. We compared CoBRA against a classical hierarchical approach on a bi-level optimization problem of production/transportation. Experimental results showed a significant advantage to CoBRA in tackling the bi-level multi-depot vehicle routing problem against a classical hierarchical approach.

A similar experimental analysis applied to other classes of bi-level optimization problems would allow to better understand the proposed algorithm pros and cons. As a future work, it would be interesting to look up a possible integration of diversification principles into CoBRA. This would give the opportunity for the algorithm to escape from local optima easier. Furthermore, the design of CoBRA is intrinsically parallel, since two sub-populations evolve independently, so that parallel computation would improve its performance in terms of computational time.

References

[1] E. Erkut and F. Gzara, "Solving the hazmat transport network design problem," *Computers & Operations Research*, vol. 35, pp. 2234–2247, 2008.

[2] M. Didi-Biha, P. Marcotte, and G. Savard, "Path-based formulations of a bilevel toll setting problem," *Optimization with Multivalued Mappings*, pp. 29–50, 2006.

[3] H. Calvete and C. Galé, "A Multiobjective Bilevel Program for Production-Distribution Planning in a Supply Chain," *Multiple Criteria Decision Making for Sustainable Energy and Transportation Systems*, pp. 155–165, 2010.

[4] E.-G. Talbi, *Metaheuristics: from design to implementation*. Wiley, 2009.

[5] G. Anandalingam and D. White, "A solution method for the linear static stackelberg problem using penalty functions," *IEEE Transactions on Automatic Control*, vol. 35, no. 10, pp. 1170 –1173, 1990.

[6] G. Eichfelder, "Multiobjective bilevel optimization," *Mathematical Programming*, vol. 123, no. 2, pp. 419–449, 2010.

[7] A. Koh, "Solving transportation bi-level programs with differential evolution," in *2007 IEEE Congress on Evolutionary Computation*. IEEE, 2008, pp. 2243–2250.

[8] X. Li, P. Tian, and X. Min, "A hierarchical particle swarm optimization for solving bilevel programming problems," *Artificial Intelligence and Soft Computing–ICAISC 2006*, pp. 1169–1178, 2006.

[9] V. Oduguwa and R. Roy, "Bi-level optimisation using genetic algorithm," in *2002 IEEE International Conference on Artificial Intelligence Systems (ICAIS 2002)*, 2002, pp. 322 – 327.

[10] M. A. Potter and K. A. D. Jong, "Cooperative coevolution: An architecture for evolving coadapted subcomponents," *Evolutionary Computation*, vol. 8, pp. 1–29, 2000.

[11] O. Ben-Ayed and C. Blair, "Computational difficulties of bilevel linear programming," *Operations Research*, vol. 38, no. 3, pp. 556–560, 1990.

[12] P. Loridan and J. Morgan, "Weak via strong stackelberg problem: New results," *Journal of Global Optimization*, vol. 8, pp. 263–287, 1996.

[13] J. Fliege and L. Vicente, "Multicriteria approach to bilevel optimization," *Journal of optimization theory and applications*, vol. 131, no. 2, pp. 209–225, 2006.

[14] Y. Lv, T. Hu, G. Wang, and Z. Wan, "A penalty function method based on Kuhn-Tucker condition for solving linear bilevel programming," *Applied Mathematics and Computation*, vol. 188, no. 1, pp. 808–813, 2007.

[15] J. Cordeau, M. Gendreau, and G. Laporte, "A tabu search heuristic for periodic and multi-depot vehicle routing problems," *Networks*, vol. 30, no. 2, pp. 105–119, 1997.
[16] J. Potvin and S. Bengio, "The vehicle routing problem with time windows part II: genetic search," *INFORMS Journal on Computing*, vol. 8, no. 2, p. 165, 1996.
[17] I. Or, "Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking," *Northwestern University, Evanston*, 1976.
[18] K. Deb and M. Goyal, "A combined genetic adaptive search (geneas) for engineering design," *Computer Science and Informatics*, vol. 26, pp. 30–45, 1996.
[19] K. Deb and R. Agrawal, "Simulated binary crossover for continuous search space," *Complex systems*, vol. 9, no. 2, pp. 115–148, 1995.