STAIRS 2016
D. Pearce and H.S. Pinto (Eds.)

doi:10.3233/978-1-61499-682-8-143

A Repair Operator for Global Solutions of Decomposable Problems

Vitor BARBOSA [a,1], Ana RESPÍCIO [b] and Filipe ALVELOS [c]
[a] *Escola Superior de Ciências Empresariais, Instituto Politécnico de Setúbal*
[b] *Centro de Matemática, Aplicações Fundamentais e Investigação Operacional, Faculdade de Ciências, Universidade de Lisboa*
[c] *Departamento de Produção e Sistemas, Universidade do Minho*

Abstract. This paper proposes a new repair operator to be used inside algorithms based on the concept of Search by Column Generation (SearchCol). This concept has revealed to be suitable to address problems represented by models that decompose the problem into several subproblems and in which a global solution can be obtained by combining solutions of the subproblems. SearchCol starts by solving the linear relaxation of the integer programming decomposition model using column generation. Metaheuristics are then used to search for the best global integer solution by combining subproblems' solutions. The new repair operator intents to fix the invalid solutions but ends up has a generator of new subproblems' solutions and allows to change the search space as the metaheuristic explores the search space. The success of the repair operator is verified in a SearchCol based evolutionary algorithm to solve a Bus Driver Rostering Problem.

Keywords. Search, Column generation, metaheuristics, repair operator.

1. Introduction

The theoretical concept of Search by Column Generation was presented as SearchCol in [1]. This framework proposes a new concept of using metaheuristic search combined with column generation [2], to obtain approximate solutions of decomposable optimization problems, as occurs when the Dantzig-Wolfe decomposition [3] is applied over a compact model. Further developments and a more detailed description of the framework concept and its implementation were included in [4].

SearchCol algorithms can be used to address problems which are very hard to solve using exact methods, such as branch-and-bound or branch-and-price, used to solve the compact or the decomposition model of a problem, respectively. For large problems, branch-and-bound can consume all the computational resources, particularly the memory, while the time needed by branch-and-price to explore all nodes can be too large in some problems.

The purpose of the framework SearchCol is to achieve good quality integer solutions for an overall problem defined by a decomposition model using a reduced amount of time when compared with the time required by branch-and-price. An integer global solution for a decomposable problem is made with a solution from each of its subproblems. Several metaheuristics based on the evolution of a single solution [5] are

[1] Corresponding Author; E-mail: vitor.barbosa@esce.ips.pt.

available in the framework to search for the best combination of subproblems' solutions found in the search space. The first population based metaheuristic, an evolutionary algorithm [6], based on the SearchCol concept was proposed in [7] to address the Bus Drivers Rostering Problem (BDRP) [8; 9]. SearchCol was already used to address other problems and with distinct metaheuristics. In [10] the network load balancing problem is addressed and the metaheuristic used is the Greedy Randomized Adaptive Search Procedure (GRASP) with path relinking. The Forest Harvest Scheduling Problem [11], the Two- and Three- Stage Bin Packing Problems in [12] and the Machine Scheduling Problem with Job Splitting in [13].

Independently of the problem and the metaheuristic used in the search for the best integer solution for the overall problem, it may occur that the subproblems' solutions available do not allow to build a global solution of good quality.

In this paper we propose a new repair operator that is integrated in the SearchCol framework to fix invalid global solutions (not respecting the global constraints). This operator allows to change the search space from inside any metaheuristic. The primary objective is to repair invalid global solutions but, in the context of the SearchCol algorithms and how the solutions are represented, if a subproblem's solution is changed, the new solution is stored as if it was obtained during the column generation.

The generation of a new solution for a given subproblem, due to the application of the repair operator, corresponds to an expansion of the search space, as if a new iteration of the column generation is run but with the advantage that it occurs during the metaheuristic search.

In the next section, an overview of the concept of search for decomposable problems is described and the evolutionary algorithm is presented. In section 3 the proposed repair operator is detailed. Section 4 presents results of computational tests to evaluate the impact of the new operator in the solutions of a BDRP decomposition problem. The paper ends with conclusions.

2. Evolutionary Algorithm for decomposable problems

Let P, the overall problem integer programming model defined by equations (1) to (4), represent an optimization problem that was decomposed into a set of similar subproblems:

$$(P)\ \ min\ \sum_{k\in K}\sum_{j\in J^k} c_j^k y_j^k \tag{1}$$

$$\sum_{j\in J^k} y_j^k = 1,\ k \in K \tag{2}$$

$$\sum_{j\in J^k} a_{ij}^k y_j^k \{\leq, =, \geq\} b_i,\ i \in I \tag{3}$$

$$y_j^k \in \{0,1\},\ k \in K, j \in J^k \tag{4}$$

Where K is the set of subproblems defined by the decomposition and J^k is the set of solutions from subproblem $k \in K$. The decision variable y_j^k is used to decide if the solution represented by column with index j for subproblem k is selected to enter the global solution. Constraints (2) state that each subproblem has a solution. Constraints

(3) are linking constraints including variables from different subproblems. Each variable y_j^k is associated with a subproblem's solution with coefficient c_j^k in the objective function (1), a coefficient 1 in the constraint with index k from the group of constraints (2) and coefficient a_{ij}^k in the constraint with index i from the group of constraints (3). The value of a_{ij}^k is the sum of the values of subproblem variables in solution j from subproblem k that are present in linking constraint with index i.

The column generation method [14] is used to solve problems modelled with decomposition models. The method starts with a restricted master problem (RMP) representing the linear relaxation of the problem (P) (without requiring the integrality of the decision variables) and containing a limited set of feasible columns. The method is iterative and in each iteration subproblems are solved to search for new columns attractive to add in the RMP to improve its solution. Whenever these columns are found, they are added to the RPM and the RPM is solved. This cycle continues until no new columns are generated. The optimal linear solution of the RMP is also the linear solution of P. The RMP solution may be fractional, but the solutions obtained from the subproblems are integer and correspond to solutions for a part of the original problem.

Considering the BDRP as an example, the objective of the problem is to define the roster with lower cost assigning all the duties to the drivers and considering all the labour rules in the definition of each driver' work-schedule. In the decomposition model (as presented in [7]) each subproblem contains the constraints and variables considered in the definition of a driver' work-schedule and the problem (P) defines that the combination of a work-schedule for each driver (which are the subproblems' solutions, each one associated with one variable y_j^k) assure that all duties are assigned to a driver (with constraints (3)) and that each driver has a work-schedule defined (with constraints (2)). The RMP is initialized with empty work-schedules and then the column generation is used solving the subproblems to obtain additional work-schedules to include in the RMP until the optimal linear solution is achieved.

The SearchCol suggests that problem (P) can be viewed as the combinatorial problem to select a single subproblem's solution from the set J^k to each subproblem, as defined by constraints (2), and where the linking constraints (3) are respected, minimizing the sum of costs (c_j^k) associated to the subproblems solutions.

A global solution to problem (P) can then be defined as $s = \big(s(1), s(2), \dots, s(k_{max})\big)$, where $s(k)$, $k \in K$, is the index of the solution j selected from set J^k to be the solution of subproblem k.

The SearchCol core idea is to solve any decomposition model using column generation and then, starting from a global solution s' use metaheuristics to search for the best solution s^* by exploring the search space defined by all the subproblems' solutions in $J^k, k \in K$. For the BDRP, a global solution is a roster and is defined by a selection of a work-schedule for each driver. This selection for a particular driver corresponds to solve the subproblem associated to that driver.

If a global solution s' is built taking randomly a solution for each subproblem, it is possible that constraints (3) are not respected. Each solution is evaluated by a pair of values: the infeasibility and the feasibility values. The first represents a measure of the amount of constraints (3) not respected by the solution s'. The feasibility value represents the cost of the solution, as defined in the objective function (1). A solution with a lower infeasibility value is always better. The feasibility value can only be compared in the feasible solutions (infeasibility equal to zero).

A new evolutionary algorithm (EA) [6] adopting the SearchCol concept was proposed in [7]. The resemblance between a global solution for a decomposition model and a chromosome, following the concept of representation of an individual/solution in an EA is evident, as show Figure 1. Each gene contains the subproblem's solution identifier and the gene locus define to which subproblem the solution belongs. The evaluation of the global solutions is also kept to compare individuals, following the concept of fitness function in EAs. For the BDRP, a chromosome represents a roster and each gene the work-schedule of the driver with index equal to the gene locus.

C1	C7	C23	…	C10
SP 1	SP 2	SP 3	…	SP k_{max}

Figure 1. Global solution representation as an EA chromosome.

The initial population of the proposed EA is configured in runtime by defining the number of individuals created by each of the pre-existing global solution generators (included in the framework that implements the SearchCol concept). The algorithm uses the tournament selection by comparing pair of individuals considering its infeasibility and feasibility values, noting that the infeasibility is the first value compared since the objective is to achieve feasible solutions. The feasibility value is only used to compare individuals with the same infeasibility value. Currently, tournaments consider pairs of individuals, but the selection pressure can be increased by adding more individuals in each tournament [15]. The variation operators implemented are the standard crossover (optionally with one or two crossover points) and the mutation operator. The standard crossover [16] is used because the locus of each gene cannot change, since each subproblem's solution is only valid for the corresponding subproblem. The mutation operator allows the replacement of the content of the k-th position gene by another subproblem's solution from the set J^k of solutions for the same subproblem k.

The use of an elite population [17] is also optional in the EA. Its size and the proportion of individuals selected from that population to the mating population are defined by parameters read in runtime. Another option available in the algorithm is the use of a local search [18; 19] procedure to explore the neighbors of the selected individual. The neighborhood of an individual is the set of solutions with a single difference in one and/or two subproblem's solutions. The local search can explore all the neighbors or stop in the first improvement found.

The results obtained by the initial implementation of the EA in [7] show that the solutions achieved by the EA in the search space composed by the added column during the column generation have a large gap to the best known integer solution values. In [20], the EA, already with the elite population and local search options, is compared with two single solution metaheuristics (variables neighborhood search [21] and simulated annealing [22]) and the EA is the metaheuristic obtaining the best solution to the higher number of test instances, showing that the gap observed in the solutions is independent of the metaheuristic used in the search.

3. Repair Operator

Independently of the metaheuristic used in a SearchCol based algorithm, the metaheuristic searches for the best combination of solutions stored in the pools of subproblems' solutions ($J^k, k \in K$) during the column generation. If a new subproblem's solution may improve the global solution but that solution was not generated yet, the metaheuristic cannot provide that improvement in the global solution.

A random global solution for the BDRP, independently of the solution cost, needs to assign all the duties once and only once in order be considered valid. Thus, infeasibility occurs in case of under-assignment: any duties are unassigned; or in case of over-assignment: whenever a duty is assigned more than once (the same duty is assigned to multiple drivers).

To address the infeasibility problem observed, neglecting the subproblem constraints, a possible strategy could be the replacement of a schedule by removing the repeated duties and inserting the missing ones, as illustrated in Figure 2. In this example, the old work-schedule for driver 3 (subproblem's solution), highlighted in grey, and that includes two repeated assignments (duties 1.3 and 3.2), is replaced by a new one where the duties assigned to that driver on days 1 and 3 are changed to the ones that were not assigned (duties 1.1 and 3.1), resulting in a global solution without under and over-assignment.

Even if the described procedure is simple, it is subjected to some restrictions, concretely:

- The described procedure illustrates the repair of a global solution for the BDRP, however the EA must be independent of the problem type;
- Existing subproblems' solutions cannot be changed, since they were used to generate a column in the RMP and they can be part of other global solutions. A change in a work-schedule (subproblem's solution) can result in an update on the cost and in the assigned duties (as in Figure 2), if it occurs, all the global solutions containing that subproblem's solution will have to be re-evaluated.
- The subproblem's constraints must be considered in the assignment of an additional duty in a driver' work-schedule.

Considering the enumerated restrictions on the implementation of a repair operator for a global solution and using the knowledge about the SearchCol framework implementation, some changes were made in the framework in order to include the repair and allow its use inside the metaheuristic.

Since the repair of a global solution of a particular problem must be independent of the metaheuristic (in this case, the EA), a new abstract method was included in the class that implements the decomposition models. With this new method, each particular decomposition can implement its own repair procedures considering the starting solution, all the knowledge about the problem (particularly the constraints), and access the already existing solutions in the pools.

Independently of the problem type, and considering the restriction about the changes on the existing subproblems' solutions, another change was implemented: if one or more subproblem's solutions are changed, the resulting solution is saved in the pool of solutions of the corresponding subproblem. The method used to save a solution checks if the solution is already in the pool of solutions and, if so, it uses the existing one, otherwise the new solution is saved and a new column is added to the RMP as occur during the CG stage.

	Day#.Duty#									
	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	Cost
Driver 1	0	0	1	0	0	1	0	0	1	120
+										
Driver 2	0	1	0	0	1	0	0	1	0	150
+										
Driver 3	0	0	1	1	0	0	0	1	0	100
+										
Driver 3 (new)	1	0	0	1	0	0	1	0	0	110
=										
Assignments	1	1	1	1	1	1	1	1	1	380

Figure 2. Repair procedure example

Figure 3 describes how the new repair operator can be included in any metaheuristic. The metaheuristic only needs to define the criteria to identify if a global solution should be repaired and then apply the repair procedure in the solution to repair. It is easily observed that all the repairing work is done in the decomposition implementation, here represented by the activities *Clean* and *Complete* to symbolize the removal of the over-assigned duties and the try to assign the remaining ones, respectively. After the repairing, the new subproblem's solutions are also saved inside the decomposition implementation, making it invisible to the outside if new columns were generated or not. From the context of the metaheuristic, it only asks the decomposition to repair a global solution and receives a new one repaired, which can be included in the population since all the components (subproblem's solutions) are valid and it was already evaluated to update the feasibility and infeasibility values. If new subproblem solutions were created during the process, they are stored in the pools of solutions and the corresponding columns are added to the RMP.

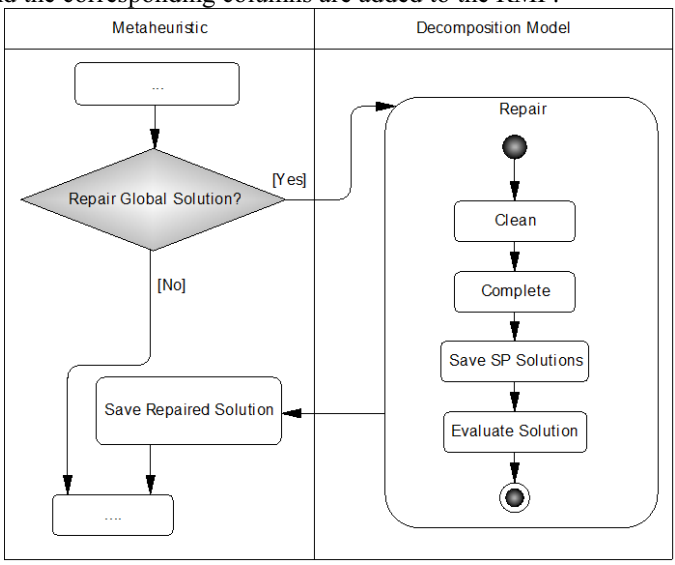

Figure 3. Call the repair procedure from the metaheuristic

For the BDRP problem, the *Clean* stage of the repair procedure consists in removing the repeated assignments, keeping the one with lower cost, if they are distinct; otherwise, the first assignment of each duty is kept and the subsequent are removed. In the *Complete* stage, if there are any duties not assigned, for each of them,

its assignment is tested in all available drivers. If a repeated duty was removed from a driver work-schedule in the same day to which the duty we need to assign belongs, it may be inserted if all the rules of the work-schedule design are verified.

As previously stated, the repair operator is included inside the decomposition model implementation. This allows each problem to use specific knowledge to implement tailored procedures in the *Clean* and *Complete* stages adapted to the problem, considering specific linking constraints sign and how the infeasibility value is obtained.

4. Computational tests

To evaluate the effect of the proposed repair operator, computational tests were made using a set of ten instances of BDRP problem, designated as P80. The P80 instances have 36 drivers available (36 possible subproblems), and 467 duties to assign, in average, for a rostering period of 28 days. Instances details are presented in [9].

The search space was defined by solving the decomposition model using the standard column generation, using only an exact solver to obtain subproblems' solutions. The number of subproblems is reduced to the number of drivers in a global solution obtained by a greedy heuristic. The number of used subproblems in each instance is presented in Table 1.

Table 1. Number of subproblems by instance

Instance	P80_1	P80_2	P80_3	P80_4	P80_5	P80_6	P80_7	P80_8	P80_9	P80_10
# subproblems	25	20	26	26	26	23	26	32	25	33

The number of columns generated by CG for each instance is presented in Table 2. The number of subproblems' solutions is significantly large to allow for a complete search in the solutions space. Considering the average number of solutions from each subproblem and the number of subproblems used, as an example, for the instance P80_1, the number of combinations to evaluate would be 1732^25.

Table 2. Number of subproblem's solutions to explore

Instance	P80_1	P80_2	P80_3	P80_4	P80_5	P80_6	P80_7	P80_8	P80_9	P80_10
# subproblem solutions	43299	33758	37168	29570	36616	26044	34318	38899	45602	15579
#solution/ subproblem	1732	1688	1430	1137	1408	1132	1320	1216	1824	472

To evaluate the repair operator inside the EA, the EA was applied in P80 with and without the repair operator. The configuration of the EA is presented in Table 3. The only change in both configurations is the use of the repair operator.

Table 4 shows the evolution of the infeasibility value obtained in each instance using the EA configuration without the repair operator. Observing the data, in the initial population, the best solution has in average 54 duties not assigned (under-assignment) and 102 assigned more than once (over-assignment). The EA is able to reduce the number of unassigned duties to less than half (57% reduction), but in average 19 duties still not assigned at the end of the search. Looking at the over-assignment, the performance is worse since the average reduction on the number of duties assigned more than once is 34%, resulting in 62 duties over-assigned in average at the end of the search. The inability of the metaheuristics to replace the over-assigned duties by the missing ones was the primary motivation to develop the repair operator.

Table 3. Evolutionary Algorithm configuration

Initial Population	1 global solution selecting the subproblem´ solutions with the higher value in the optimal solution of the CG; 1 global solution selecting the subproblem´ solution that was optimal last time the subproblem was solved; (7x*Number of Subproblems*) global solutions selecting randomly with the probability of one subproblem´ solution being chosen given by its weight in the CG optimal solution; (3x*Number of Subproblems*) global solutions selecting randomly each subproblem´ solution.
Selection	Binary tournament.
Variation Operators	One point Crossover over 90% of individuals; Mutation operator over 20% of offspring.
Elite Population	Population with the 20-th best solutions; 5% of the matting pool parents are selected from here.
Local Search	Local Search used to explore the neighborhood of a random individual defined by the solutions with a single change (1 distinct subproblem solution), stopping in the first improvement found. The local search is used in a random individual each 10 iterations and in the best solution in the last iterations (when approaching the number of iterations without improvement limit).
Stopping Criteria	300 iterations without improvement in the best solution.
Repair Operator	Used every 100 iterations on the individuals with infeasibility value greater than zero (duties over/under assigned). The first use occurs only after half of the iterations defined in the stopping criteria were run.

Table 5 shows the evolution of the infeasibility value obtained in each instance using the EA configuration with the repair operator. The table shows that the best solutions found in the initial population are slightly worse, with 60 duties not assigned in average and 113 over-assigned. However, the repair operator is totally effective in the removal of the over-assigned duties and also in the re-assignment of the missing ones, as the final results are zero duties not assigned and zero assigned more than once for all the instances.

Table 4. Infeasibility evolution using the EA configuration without repair

			Under-assignment			Over-assignment		
Instance	**Iterations**	**Time (s)**	**Begin**	**End**	**Reduction**	**Begin**	**End**	**Reduction**
P80_1	562	34,3	87	7	-92%	152	73	-52%
P80_2	432	15,8	78	11	-86%	148	61	-59%
P80_3	1032	56,2	77	21	-73%	101	53	-48%
P80_4	982	39,2	60	16	-73%	99	39	-61%
P80_5	562	39,2	52	6	-88%	123	66	-46%
P80_6	962	33,3	7	5	-29%	56	48	-14%
P80_7	982	82,5	71	63	-11%	90	108	20%
P80_8	679	61,5	44	0	-100%	92	30	-67%
P80_9	682	75,7	9	8	-11%	43	41	-5%
P80_10	1022	102,6	54	49	-9%	113	105	-7%
Average	789,7	54,0	53,9	18,6	-57%	101,7	62,4	-34%

Besides the clear effectiveness on the repair function, the operator extends the metaheuristic search. In the configuration without the repair operator, the average number of iterations was 790 and in the configuration with the repair operator the

average number of iterations is 2212, showing that multiple improvements were achieved, not only in the infeasibility value but also in the feasibility (cost of the global solution). The raise on the number of iterations and the additional computational effort of the repair operator results in an average increase of approximately 300% in the EA runtime.

Table 5. Infeasibility evolution using the EA configuration with repair

			Under-assignment			Over-assignment		
Instance	**Iterations**	**Time (s)**	**Begin**	**End**	**Reduction**	**Begin**	**End**	**Reduction**
P80_1	2002	146,5	83	0	-100%	150	0	-100%
P80_2	1802	117,8	74	0	-100%	144	0	-100%
P80_3	2402	90,7	69	0	-100%	113	0	-100%
P80_4	1202	147,5	61	0	-100%	100	0	-100%
P80_5	4102	370,0	10	0	-100%	69	0	-100%
P80_6	1702	69,3	55	0	-100%	117	0	-100%
P80_7	1702	167,1	67	0	-100%	113	0	-100%
P80_8	3302	355,0	48	0	-100%	92	0	-100%
P80_9	2202	110,2	73	0	-100%	115	0	-100%
P80_10	1702	388,9	59	0	-100%	114	0	-100%
Average	2212	196,3	59,9	0	-100%	112,7	0	-100%

The value of the feasibility observed at the end of the search in the configuration is presented in Table 6. The average value of the final population and the value of the best global solution found are presented. The feasibility values of the configuration without repair are not presented since the value is distorted by the cost of the duties over-assigned and do not include the cost of the ones not assigned.

Considering the final best global solution obtained by the configuration with the repair operator, which is an integer and valid solution for the global problem, in Table 6 the value is compared with the lower bound provided by the column generation. The value of the column generation solution is necessarily lower than or equal to the optimal integer value. A positive gap does not imply that the integer solution obtained is not optimal or of poor quality, it only means that its optimality was not proven. Note that if the lower bound is poor, even an optimal solution will have a large gap.

Even if the EA with the repair operator was effective in achieving a clean roster for each of the instances, the gaps observed show that it was unable to achieve the minimum number of drivers used in the optimal solution. In this set of tests, the fixed cost of using an extra driver was set to 10000 cost units, which corresponds to around 4% of the gap. The average gap in the ten instances is 7,1%.

Further developments need to be made in the EA with the repair operator to achieve more compact solutions in order to set free the additional driver that is being used in the current solutions.

To validate the effectiveness of the repair operator, ten tests with each of the instances were made. The results are presented in Table 7 where, for each instance, the values of the best, the average and the standard deviation of the feasibility value are presented. The average count of unassigned and over-assigned in the best global solution from the initial population of each run, the average number of generations and the average search time are also presented. All the runs achieved clean global solutions, without unassigned or over-assigned duties. The best feasibility value obtained in the 10 runs was improved for 5 instances (highlighted in bold), however these solutions are not achieved in all runs.

Table 6. Feasibility values at the end of the search and LB gap

With Repair				
Instance	**Average**	**Best**	**LB**	**Gap**
P80_1	268283	256369	244284	4,9%
P80_2	214288	202750	188623	7,5%
P80_3	281289	269320	255074	5,6%
P80_4	278925	266473	244182	9,1%
P80_5	278114	255306	237007	7,7%
P80_6	247472	236029	223659	5,5%
P80_7	279103	277466	255216	8,7%
P80_8	344251	321166	304580	5,4%
P80_9	268730	256351	236155	8,6%
P80_10	352719	339975	315494	7,80%

Table 7 also presents, in the last column, the time used in the search with the local search and the repair operator. Obviously, the increment on the number of iterations and the additional computational work of the repair operator increases the running time, but with an average search time of 210s all the runs were able to return a complete global solution.

Table 7. 10 runs results (EA with repair operator)

	Feasibility			**Average**			
Instance	**Best**	**Average**	**σ**	**Under-assignments**	**Over-assignments**	**Generations**	**Time (s)**
P80_1	256432,7	262464,1	5157,8	74,9	97,6	2528	166,3
P80_2	202724,5	202775	47,1	49,3	76,4	1662	59,2
P80_3	269199,1	274256,1	5266,5	79,8	106,7	1702	100,3
P80_4	**256339,6**	261457,4	5188,4	65,9	109	2522	215,4
P80_5	**245204,4**	252226,7	4822,0	57,3	111,9	2882	266,6
P80_6	235848,7	238944,3	4775,7	68,2	90,5	1652	86,4
P80_7	**267487,2**	276467,5	3156,4	79,1	101,9	1959	145,6
P80_8	321023,6	323155,3	4272,9	83,5	134,3	2182	526,8
P80_9	**246372,3**	252362	5130,8	63	102,4	1802	149,7
P80_10	**329704,3**	333996,3	5117,3	91,7	139	2092	394,3
Average	202724,5	267810,5	36736,4	71,27	106,97	2098,31	211,1

To better observe the effect of the repair operator through the search, Figure 4 and Figure 5 present the evolution of the infeasibility and feasibility values (vertical axis), respectively, for instance P80_2, using a not-to-scale representation of the number of iterations (horizontal axis). In Figure 4, the best solution in the initial population has 46 duties not assigned and 78 over-assigned. The repair operator is used by the first time at iteration 300 after which only one duty is not assigned. In the second repair, in iteration 400, the best solution achieves the zero infeasibility value. Besides the impact on the best solution value, the charts also give evidence of decrease on the population average solution value at each application of the repair operator.

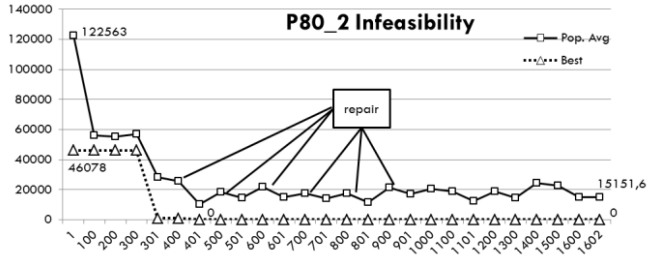

Figure 4. Evolution of the infeasibility

Figure 5 shows the evolution of the cost of the best solution and the average of the entire population. A mark identifies the iteration when the zero infeasibility value was reached and then very small decreases are obtained until the iteration 1100 when the global solution was able to reduce one driver usage (corresponding to 10000 cost units).

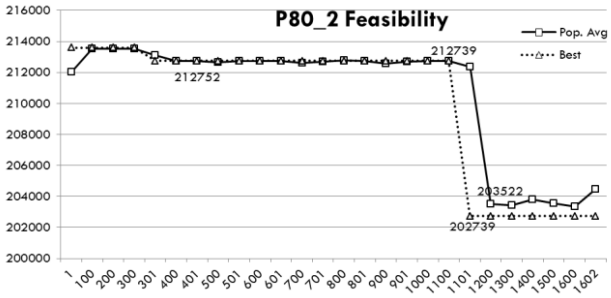

Figure 5. Evolution of the feasibility

5. Conclusion

We proposed and evaluated a new repair operator which was developed to be used by an evolutionary algorithm based on the concept of search by column generation. We used the evolutionary algorithm in the context of the BDRP to evaluate the proposed operator but it can be used by any other metaheuristic available in the SearchCol framework or in any other decomposition problem since each decomposition can define its own procedures to clean and complete invalid global solutions.

Computational tests with a set of BDRP instances show the effectiveness of the repair operator to fix invalid solutions, which do not fully respect the linking constraints, in the search for a global integer solution. The additional runtime results in better quality solutions, especially in valid ones considering all the linking constraints.

The repair procedure can be improved to help the metaheuristic achieving integer solutions of higher quality, since the gaps to best known integer solutions show a slack for improvement and we can study new strategies to reassign the duties in the incomplete solutions.

The usage of the proposed repair operator inside the evolutionary algorithm addressing the BDRP intends to be a proof of concept, from which tailored repair operators may result to address distinct problems or/and be shared by all the metaheuristics available in the SearchCol framework.

Acknowledgement

This work is partially supported by National Funding from FCT - Fundação para a Ciência e a Tecnologia, under the project UID/MAT/04561/2013.

References

[1] F. Alvelos, A. de Sousa, and D. Santos, SearchCol: Metaheuristic Search by Column Generation, in: *Hybrid Metaheuristics,* M. Blesa, C. Blum, G. Raidl, A. Roli, and M. Sampels, eds., Springer Berlin / Heidelberg, 2010, pp. 190-205.

[2] G. Desaulniers, J. Desrosiers, and M.M. Solomon, *Column Generation*, Springer, New York, 2005.

[3] G.B. Dantzig and P. Wolfe, Decomposition Principle for Linear Programs, *Operations Research* **8** (1960), 101-111.

[4] F. Alvelos, A. Sousa, and D. Santos, Combining column generation and metaheuristics, in: *Hybrid metaheuristics,* E.-G. Talbi, ed., Springer, 2013, pp. 285-334.

[5] E.-G. Talbi, Single-Solution Based Metaheuristics, in: *Metaheuristics*, John Wiley & Sons, Inc., 2009, pp. 87-189.

[6] J.H. Holland, *Adaptation in natural and artificial systems*, MIT Press, 1992.

[7] V. Barbosa, A. Respício, and F. Alvelos, A Hybrid Metaheuristic for the Bus Driver Rostering Problem, in: *ICORES 2013 – 2nd International Conference on Operations Research and Enterprise Systems,* B. Vitoriano and F. Valente, eds., SCITEPRESS, Barcelona, 2013, pp. 32-42.

[8] M. Moz, A. Respício, and M. Pato, Bi-objective evolutionary heuristics for bus driver rostering, *Public Transport* **1** (2009), 189-210.

[9] A. Respício, M. Moz, and M. Vaz Pato, Enhanced genetic algorithms for a bi-objective bus driver rostering problem: a computational study, *International Transactions in Operational Research* **20** (2013), 443-470.

[10] D. Santos, A. de Sousa, and F. Alvelos, A hybrid column generation with GRASP and path relinking for the network load balancing problem, *Computers & Operations Research* **40** (2013), 3147-3158.

[11] I. Martins, F. Alvelos, and M. Constantino, Decompositions and a Matheuristic for a Forest Harvest Scheduling Problem, in: *Operational Research,* J.P. Almeida, J.F. Oliveira, and A.A. Pinto, eds., Springer International Publishing, 2015, pp. 237-260.

[12] F. Alvelos, E. Silva, and J. de Carvalho, A Hybrid Heuristic Based on Column Generation for Two- and Three- Stage Bin Packing Problems, in: *Computational Science and Its Applications – ICCSA 2014,* B. Murgante, S. Misra, A.C. Rocha, C. Torre, J. Rocha, M. Falcão, D. Taniar, B. Apduhan, and O. Gervasi, eds., Springer International Publishing, 2014, pp. 211-226.

[13] L. Florêncio, C. Pimentel, and F. Alvelos, An Exact and a Hybrid Approach for a Machine Scheduling Problem with Job Splitting, in: *Operational Research,* J.P. Almeida, J.F. Oliveira, and A.A. Pinto, eds., Springer International Publishing, 2015, pp. 191-212.

[14] M.E. Lübbecke and J. Desrosiers, Selected Topics in Column Generation, *Oper. Res.* **53** (2005), 1007-1023.

[15] B.L. Miller and D.E. Goldberg, Genetic algorithms, tournament selection, and the effects of noise, *Complex Systems* **9** (1995), 193-212.

[16] M. Mitchell, *An introduction to genetic algorithms*, MIT Press, 1996.

[17] J.A. Vasconcelos, J.A. Ramirez, R.H.C. Takahashi, and R.R. Saldanha, Improvements in genetic algorithms, *Magnetics, IEEE Transactions on* **37** (2001), 3414-3417.

[18] D.S. Johnson, C.H. Papadimitriou, and M. Yannakakis, How easy is local search?, *Journal of Computer and System Sciences* **37** (1988), 79-100.

[19] E.H.E.H.L. Aarts and J.K. Lenstra, *Local Search in Combinatorial Optimization*, John Wiley & Sons, Inc., 1997.

[20] V. Barbosa, A. Respício, and F. Alvelos, Comparing Hybrid Metaheuristics for the Bus Driver Rostering Problem, in: *Intelligent Decision Technologies,* R. Neves-Silva, L.C. Jain, and R.J. Howlett, eds., Springer International Publishing, 2015, pp. 43-53.

[21] N. Mladenović and P. Hansen, Variable neighborhood search, *Computers & Operations Research* **24** (1997), 1097-1100.

[22] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, Optimization by Simulated Annealing, *Science* **220** (1983), 671-680.