A Collaborative Combination between Column Generation and Ant Colony Optimization for Solving Set Packing Problems

Aurélien Merel[1,2], Xavier Gandibleux[1], Sophie Demassey[2]

[1] Laboratoire d'Informatique de Nantes-Atlantique (LINA)
Université de Nantes – UFR Sciences
2 rue de la Houssinière, BP 92208, 44322 Nantes Cedex 3, France.
aurelien.merel@univ-nantes.fr, xavier.gandibleux@univ-nantes.fr

[2] Laboratoire d'Informatique de Nantes-Atlantique (LINA)
École des Mines de Nantes
4 rue Alfred Kastler, BP 20722, 44307 Nantes Cedex 3, France.
sophie.demassey@mines-nantes.fr

Abstract

The Set Packing Problem (SPP) is a well-known NP-hard combinatorial optimization problem. This paper addresses the SPP through a practical case study, namely the Railway Infrastructure Capacity (RIC) problem. SPP instance sizes yielded by this problem contain up to 20,000 variables and 350,000 constraints and consequently appear intractable in reasonable computational time by pure exact algorithms. As an operational response, approximate methods based on Ant Colony Optimization (ACO) and Greedy Randomized Adaptive Search Procedure (GRASP) have been developed. However, even though ACO has given the best results so far, it still requires a significant computational time on some instances.

This paper presents a hybridization coupling an ACO metaheuristic with a Column Generation (CG) procedure. CG is used as a preprocessing to provide a reduced SPP instance to ACO as well as a valuable upper bound on the solution quality. Experiments show that the cumulated computational time of ACO with its CG preprocessing is in most cases smaller than the computational time of the sole ACO method, yielding a substantial reduction that reaches 85%. Moreover, this improvement is in general not counterbalanced by a degradation of the solution quality.

1 Introduction

1.1 The Set Packing Problem and its Application

The Set Packing Problem (SPP) is a well-known NP-hard combinatorial optimization problem. Given a finite set of items $J = \{1, ..., n\}$, and m subsets of J denoted $T_1, ..., T_m$, a packing is a subset $P \subseteq J$ such that $|T_i \cap P| \leq 1,\ \forall i \in 1..m$ (all items in a subset T_i are mutually exclusive). Each item $j \in J$ has a weight denoted c_j. The SPP aims at finding the packing P that maximizes the sum of its items weights. If all items of J have the same weight, the problem is said to be Unicost SPP (USPP). Otherwise, it is said to be Weighted SPP (WSPP). A generic Integer Programming (IP) formulation of the SPP can be written as follows:

$$\left[\begin{array}{lll} \max & \sum\limits_{j \in J} c_j x_j & \\ \text{s.t.} & \sum\limits_{j \in T_i} x_j \leq 1 & ,\forall i \in 1..m \\ & x_j \in \{0,1\} & ,\forall j \in J \end{array}\right] \tag{1}$$

where each variable x_j is set to 1 if item j is selected in the packing P, and 0 otherwise. The m constraints represent the mutual exclusivity of all items inside each subset $T_i,\ \forall i \in 1..m$.

This paper addresses the IP formulation of the SPP (1), applied to the Railway Infrastructure Capacity (RIC) problem presented by Merel et al. [15]. Given a railway infrastructure and a set of trains, it entails routing the maximum number of trains through the infrastructure while respecting safety constraints. Packing constraints emerge when the same railway track is requested at the same time by several distinct trains. The study presented in this paper is based on RIC problem instances concerning a French infrastructure, namely the Pierrefitte-Gonesse junction which is located near Paris. Tested instances of the RIC problem yield SPP formulations that contain up to 20,000 binary variables and 350,000 packing constraints, implicating severe resolution difficulties.

1.2 Approach Summary

Gandibleux et al. [10, 11] give an overview of existing methods designed to solve the SPP. Exact methods mainly consist in branch-and-cut algorithms based on a polyhedral study of the SPP matrix structure. Very large formulations are however intractable in a sufficiently short time. Consequently, metaheuristics have been developed, notably in the context of the RIC problem. They include a Greedy Randomized Adaptive Search Procedure (GRASP) by Delorme et al. [5] and an Ant Colony Optimization (ACO) algorithm by Gandibleux et al. [10, 11]. The latter has been reported to outperform the former on both random and RIC problem instances. Alidaee et al. [1] propose an approach that consists in transposing the SPP into a generic quadratic formulation solved with a tabu search method and obtain results that are similar to GRASP. These previous studies enlight that ACO has given the best results on our problem, we consequently use this algorithm as a comparison reference in this paper.

The RIC problem occurs in the context of strategic railway planning and does not require extremely short computational times. However, it is often embedded in wider studies which require to obtain good quality solutions in reasonable time. From that perspective, ACO spends a computational time that can be considered as too long on some instances, namely up to 20 hours.

To tackle these instances, this paper proposes to hybridize the ACO metaheuristic with a Column Generation (CG) procedure designed for the RIC problem. The CG procedure solves the SPP continuous relaxation to optimality and produces a subset of generated variables. The SPP formulation restricted to this subset is then solved by the ACO metaheuristic. The efficiency of the CG procedure is mainly ensured thanks to the integration of additional techniques, some of them using RIC-specific data.

Computational results show that prepending the CG procedure to ACO often yields a substantial computational time reduction, while the quality of solutions is not degraded.

Section 2 presents some advantages of hybridizing exact LP-based methods with metaheuristics as well as a brief review of existing combinations. The algorithm we propose is explained in section 3 and computational results are presented in section 4. Conclusive remarks and perspectives are given in section 5.

2 Literature Review

2.1 Principle of Hybridization

Metaheuristics and IP-based exact algorithms have peculiar advantages that can be complementary, as put forward by Dumitrescu and Stützle [7]. IP-based and LP-based method often provide valuable bounds and information from the resolution of the problem continuous relaxation. For example, Puchinger et al. [18] propose a memetic algorithm that exploits integer solutions found by a branch-and-cut procedure running in parallel and dual solutions provided by its underlying LP resolutions. On the other hand, metaheuristics can often find solutions of good quality within a much smaller computational time. However, they are typically not designed to prove a solution optimality.

Hybridizations can be classified according to the way the exact algorithm and the metaheuristic interact together. Puchinger [17] suggests a classification into "collaborative" and "integrative" combinations. A collaborative combination means that the exact algorithm and the metaheuristic are executed separately, either sequentially or in parallel. Integrative combination consists in embedding one algorithm into the other.

A wide range of hybridization examples and applications is covered by Blum et al. [2], such as the hybridization of an ACO algorithm with constraint programming techniques and additional examples of interactions between branch-and-bound based procedures and metaheuristics.

2.2 Column Generation and Related Hybridizations

CG is a generic procedure designed to solve LP formulations containing a very large number of variables or for which variables can not be explicitly enumerated. It relies on the fact that considering all variables is not necessary to find an optimal solution. In the context of column generation, the LP formulation is called the "Master Problem" (MP).

A CG procedure is a sequence of iterations, each of them consisting in solving a Restricted MP (RMP), namely the MP restricted to a subset of variables, and a pricing problem which aims at finding new variables (also called columns) to insert into the RMP. If such columns are found, they are inserted and another iteration is started. If no such columns exist, the RMP optimal solution found at the last iteration is considered as the MP optimal solution.

Solving the pricing problem (or subproblem) entails computing the reduced cost of potentially new columns. For maximization problems such as the SPP, one tries to find columns with a positive reduced cost. An efficient pricing algorithm is an essential issue to obtain a high-performing CG procedure, even though the pricing problem is not always easy to solve.

Integrative combinations have been proposed in which a metaheuristic is the slave algorithm responsible for solving the pricing problem. They include the works by Filho and Lorena [9] which proposes a constructive genetic algorithm and by Puchinger [17] which proposes, among others, an evolutionary algorithm. Clements et al. [3] propose a collaborative sequential combination in which a heuristic is used to generate a restricted formulation of a production-line scheduling problem. This restricted formulation is solved afterwards by a CG-based procedure. Although these collaboration configurations are not strictly the same, in both cases the metaheuristic has the responsibility of providing variables to the exact algorithm.

2.3 Ant Colony Optimization Metaheuristics

ACO metaheuristics rely on the cooperation of simple agents ("ants") *via* a "pheromone trail" to find good solutions to an optimization problem. Each ant builds a solution to the problem by performing successive moves in the search space. The pheromone trail reflects the evolution of the whole colony and guides each ant towards a good solution. This principle, initially suggested by Colorni et al. [4] for the traveling salesman problem, has allowed applications to many combinatorial optimization problems such as the set covering problem by Lessing et al. [13].

To the best of our knowledge, only few hybridizations of ACO methods with IP-based exact algorithms have been reported. A collaborative sequential combination is suggested by Doerner et al. [6] in the context of a multiobjective combinatorial optimization problem. A single-objective exact IP-based method is used as a preprocessing to a multiobjective ACO algorithm to initialize ACO pheromone trails, in order to increase the number of efficient solutions found by ACO.

Lessing et al. [13], while comparing performances of four ACO algorithms on the set covering problem, emphasize the effect of using additional heuristic information. Heuristic information can be either static or dynamic: the former is computed only once whereas the latter is updated throughout the solution process. Examples of heuristic information include Lagrangian costs, illustrating the exploitability of information provided by LP-based methods. Maniezzo [14] takes advantage of lower bounds computed by LP algorithms to quantify the attractiveness of ACO movements in the search space as well as initialize the pheromone trail, for solving the quadratic assignment problem.

3 A Hybrid Algorithm for the Set Packing Problem

We propose a combination of CG and ACO to reduce the time spent on some instances by the sole ACO algorithm. According to the classification by Puchinger [17], it is a collaborative sequential combination in which the CG procedure is a preprocessing to the ACO algorithm. The proposed algorithm, denoted CG-ACO, can be summarized as follows:

1. Starting with a small initial subset of items $J_0 \subset J$, the CG procedure solves to optimality the continuous relaxation of (1) and returns:
   - a set $J' \subseteq J$ corresponding to all variables generated during the procedure;
   - an optimal solution to the SPP continuous relaxation, denoted $\tilde{x}^* = \{\tilde{x}_j^*\}_{j \in J}$, with $\tilde{x}_j^* \in [0,1],\ \forall j \in J'$ and $\tilde{x}_j^* = 0,\ \forall j \in J \setminus J'$;
2. The ACO algorithm solves the SPP restricted to items of J' and returns a solution $\{x_j^*\}_{j \in J'}$ with $x_j^* \in \{0,1\},\ \forall j \in J'$;
3. Variables associated to items not belonging to J' are set to 0, thus making a solution $x^* = \{x_j^*\}_{j \in J}$ that respects the following constraints:

$$\begin{bmatrix} \sum_{j \in T_i} x_j^* \leq 1 & ,\forall i \in 1..m \\ x_j^* \in \{0,1\} & ,\forall j \in J' \\ x_j^* = 0 & ,\forall j \in J \setminus J' \end{bmatrix} \tag{2}$$

   This solution is thus feasible for the initial SPP (1).

The objective value associated with the solution returned by the CG-ACO algorithm is defined by:

$$z^* := \sum_{j \in J} c_j x_j^* \tag{3}$$

The ACO pheromone trail is initialized with $\tilde{x}^*$, issued by the CG procedure. The continuous optimal objective value z_{LP}^* is an upper bound for z^* and is defined by:

$$z_{LP}^* := \sum_{j \in J'} c_j \tilde{x}_j^* \tag{4}$$

The aim of this approach is to gain performance by providing a much smaller formulation to ACO than the initial full SPP formulation. In other words, we aim at having a set J' much smaller than the initial J while keeping most promising variables in J' so that ACO can still find a good solution. The set J' can be compared to the core as developed by Huston et al. [12] in the generic case of 0/1 IP as it is a mean to solve a reduced problem while keeping a good solution quality. The continuous optimal solution can be seen as an equivalent of the efficiency measure they present.

Subsection 3.1 highlights important improvements brought to the CG procedure to increase its performance. The ACO algorithm is presented in subsection 3.2 as well as the slight modification made to improve its collaboration with the CG procedure.

3.1 Improved Column Generation Procedure

The CG procedure returns an optimal solution for its MP, which is the continuous relaxation of the SPP formulation defined by (1). Particularities and improvements contributing to make it an efficient preprocessing method are summarized here. More details are given by Merel et al. [15].

3.1.1 Subproblem Solution

The RIC problem data implicitly provides all possible columns and allows the pricing problem to be solved in polynomial time. Consequently, in our case, computing the reduced cost for all potentially new columns requires in most cases a neglectable proportion of the CG time, namely between 2% and 10%.

3.1.2 Dynamic Constraint Aggregation

A significant improvement exploits the presence of many redundant and dominated constraints in the SPP formulation. The majority of them is removed from the formulation before each RMP resolution. This drastically reduces the CG computational time. This technique, known as dynamic constraint aggregation and initially presented by Elhallaoui et al. [8], does not alter the MP optimal solution. Its application to the SPP formulation of the RIC problem is detailed by Merel et al. [16].

As the aggregation algorithm is executed at each CG iteration, it has to be as efficient as possible. Consequently, we make use of RIC-specific data to heuristically achieve a strong problem reduction as quickly as possible.

Dynamically removing constraints implicates that some dual variables values are missing at each iteration in order to solve the subproblem. Consequently, the algorithm features a computation of a full RMP dual solution at each iteration in a way that depicts the saturation of constraints by primal variables so that appropriate column are generated.

3.1.3 Column Insertion Strategy

Among all columns with a positive reduced cost, column insertion strategies select columns aiming at reaching the MP optimal solution in as few CG iterations as possible while preventing the RMP formulation from growing too much. Controlling this growth contributes to keep reasonable RMP solution times and to end up the CG procedure with a relatively small set J', accordingly reducing the subsequent ACO computational time.

Insertion strategies use both generic and RIC-specific criteria to select columns. Generic criteria include a threshold value for the reduced cost and a maximum number of columns that can be inserted at each iteration. RIC-specific criteria include inserting not more than one column per train and diversifying the use of railway tracks. The aggregated structure of the constraint set can also be used as a criterion by favoring columns that preserve existing dominance and redundancy relationships as much as possible, acting to preserve the RMP as small as possible.

3.2 Collaborative Ant Colony Optimization Algorithm for the Set Packing Problem

3.2.1 Original Ant Colony Optimization Algorithm

The considered ACO algorithm is an iterative process proposed by Gandibleux et al. [11] in which each ant builds a complete solution at every iteration. Its parameters such as the number of ants and the stopping criterion are automatically determined. Notably, the stopping criterion is based on a detection of the convergence of the algorithm. This paragraph focuses on the pheromone trail design and management.

Given a SPP formulation (1), this version of ACO implements the pheromone trail as a vector of size n. For each element $j \in J$, x_j is associated to one pheromone level $\phi_j \in [0, 1]$. All pheromone levels are initialized to 1, meaning that no variable is *a priori* favored over any other.

At the end of each iteration, all pheromone levels are multiplied by an "evaporation" coefficient in the range $]0, 1[$, and pheromone levels associated to variables present in the best solution are increased by a "deposit" value. Consequently, variables appearing repeatedly in good solutions are progressively favored. Perturbation mechanisms are also presented by Gandibleux et al. [11] to diversify the solutions explored.

This ACO implementation relies on a $n \times n$ exclusivity matrix describing pairwise incompatibilities between variables. It is built at the initialization according to the SPP matrix and allows to determine in constant time whether two variables are compatible or not. However, the time needed for building this matrix often strongly penalizes the ACO computational time.

3.2.2 Exploitation of the Column Generation Output

In the CG-ACO combination, ACO solves a restricted SPP formulation composed of the $n' = |J'|$ variables present at the end of the CG procedure. The ACO initialization is modified so that initial pheromone levels can use the continuous optimal solution $\tilde{x}^*$. Formally, instead of the default value of 1, the initialization of pheromones is written $\phi_j \leftarrow \tilde{x}^*_j, \forall j \in J'$, every $\tilde{x}^*_j$ being inside the range $[0, 1]$. Indeed, a number of variables can have their initial pheromone value set to 0 because the corresponding $\tilde{x}^*_j$ is equal to 0. The consequence of such an initialization on the ACO integer solution is studied at the end of the next section.

4 Computational Experiments

As the CG procedure subproblem and improvements use RIC-specific data, relevant tests could only be performed on 36 private instances yielded by the RIC problem. As preliminary observations showed that the number of variables n was the main factor impacting the ACO computational time, results focus on a panel of 22 instances with significant enough variations of n. The SPP model developed for the RIC problem implicates a constant number of packing constraints for all instances, namely about 350,000. Considered RIC instances all yield an USPP formulation with all weights equal to 1.

All tests were run on a computer equipped with an Intel Core2 Duo CPU at 2.60GHz and 2Gb RAM. LP resolutions inside the CG algorithm are made with the Coin-OR Clp 1.09 solver. Fifty runs were made on each instance for both algorithms.

A comparison is presented between solving full SPP instances with the sole original ACO metaheuristic on the one side, and the CG-ACO combination in which ACO is initialized with the CG continuous solution on the other side. Comparisons are made in terms of computational time and objective value. The upper bound provided by CG is also reported.

4.1 Computational Time

4.1.1 Total Time

Table 1 shows the CPU time required for both methods from their start to their termination with an integer solution returned. Instances are sorted by increasing size. The column n shows the number of variables in the full SPP formulation. Columns "aco" and "cgaco" respectively present computational times for ACO and CG-ACO. The "n'" column shows the number of variables present at the end of the CG procedure out of the n possible variables.

Inst.	n	aco	cgaco	n'	Inst.	n	aco	cgaco	n'
1	876	**17**	28	499	12	5,548	3,174	**537**	1,731
2	1,168	39	**37**	546	13	5,790	2,784	**920**	2,458
3	1,460	81	**49**	662	14	6,139	2,448	**451**	2,343
4	1,737	**85**	103	1,021	15	7,527	5,887	**907**	2,462
5	2,044	159	**51**	691	16	8,092	4,808	**901**	3,134
6	2,631	**227**	228	1,578	17	8,770	6,674	**2,392**	3,134
7	2,920	539	**167**	1,161	18	11,001	14,961	**4,096**	3,892
8	3,468	529	**395**	2,051	19	11,560	12,761	**4,815**	4,992
9	3,796	1,138	**154**	1,144	20	15,028	27,913	**4,692**	5,141
10	4,053	917	**187**	1,161	21	16,663	39,246	**13,561**	6,115
11	4,624	1,133	**494**	2,323	22	21,964	71,751	**24,681**	8,095

Table 1: Average computational time (seconds) out of 50 runs and number of variables generated by CG

It appears from table 1 that the CG-ACO combination runs faster on a large majority of instances. On smallest instances (#1 to #8), computational time is relatively tight between the two methods, whereas

CG-ACO becomes clearly better on largest instances. For instances containing more than 5,000 variables, CG-ACO decreases the computational time by a significant amount varying between 62% and 85%.

These results are illustrated by figures 1 and 2 which present boxplots of the computational time for both algorithms on a selection of instances. Figure 1 confirms the relative tightness of the two algorithms on smallest instances. On the contrary, figure 2 shows the clear advantage of CG-ACO on big instances and the relatively small dispersion of the computational time values.

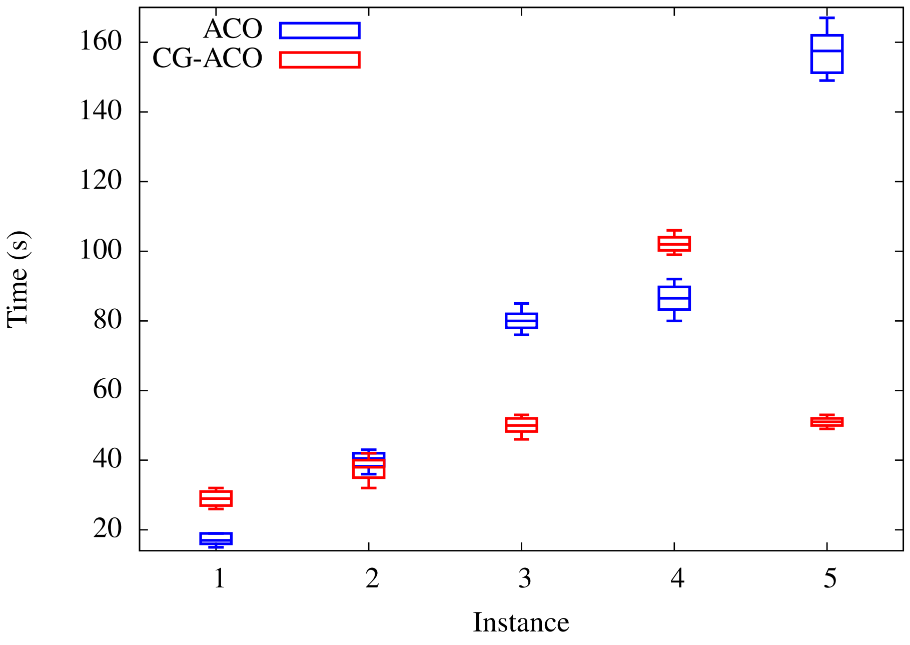

Figure 1: Computational time of ACO and CG-ACO (instances 1 to 5)

The n' column of table 1 gives a supplementary insight to explain these results, as the best time reductions are often correlated with small $\frac{n'}{n}$ ratios. For example, on instance #18 only 35% of the 11,001 possible columns were generated and the computational time is reduced by 73%. On the contrary, instances #1, #4 and #6 present a ratio $\frac{n'}{n}$ over 55% and CG-ACO gives a poorer performance. These observations mean that when fewer columns have to be generated to reach the continuous optimal solution, a smaller computational time is achieved for both parts of the CG-ACO combination.

The SPP matrix structure also seems to affect the CG behavior, though no precise causal relationship has been determined. On the contrary, the standalone ACO algorithm computational time nearly depends on the sole number of variables, showing a roughly quadratic increase on it.

4.1.2 Distribution of Computational Time between Ant Colony Optimization and Column Generation

In order to have a deeper insight into the CG-ACO behavior, table 2 presents the time share occupied by the CG part of the CG-ACO algorithm, the remaining percentage being used by the ACO part.

Inst.	cg%	Inst.	cg%	Inst.	cg%	Inst.	cg%	Inst.	cg%
1	92%	6	81%	11	75%	16	69%	21	82%
2	96%	7	80%	12	80%	17	69%	22	84%
3	88%	8	78%	13	77%	18	81%		
4	85%	9	78%	14	70%	19	74%		
5	86%	10	74%	15	76%	20	70%		

Table 2: Proportion of total CPU time spent in the CG procedure in CG-ACO

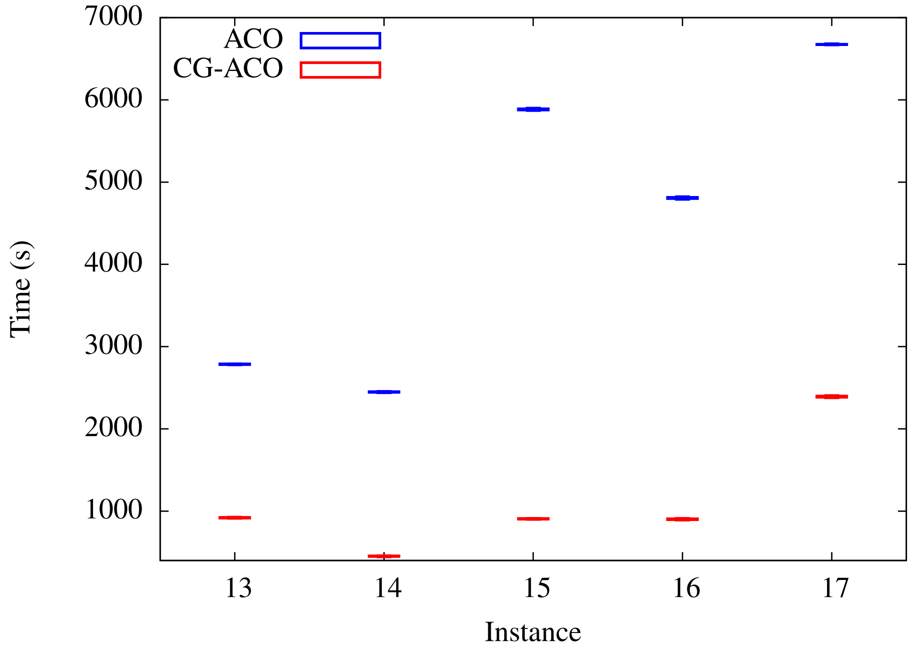

Figure 2: Computational time of ACO and CG-ACO (instances 13 to 17)

CG clearly occupies the major part of the computational time, leading to two conclusions. Firstly, as most of the overall computational times are in favor of CG-ACO, it shows that the CG procedure fulfills well its preprocessing role by drastically reducing the ACO computational time. Secondly, it puts forward how essential it is to design an efficient CG procedure. In our case, improvements brought to the CG procedure such as constraint aggregation and column selection strategies prove to be fruitful.

4.2 Solution Quality and Upper Bound

Table 3 presents the best and average solutions found by both methods and the upper bound z^*_{LP} (4) issued by the CG procedure.

		best z^*		average z^*				best z^*		average z^*	
Inst.	z^*_{LP}	aco	cgaco	aco	cgaco	Inst.	z^*_{LP}	aco	cgaco	aco	cgaco
1	28	27	27	26.6	**26.8**	12	34	28	28	27.8	**28.0**
2	28	27	27	27.0	27.0	13	64	**54**	53	**53.9**	52.8
3	31	28	28	27.9	**28.0**	14	83	79	79	78.7	**78.9**
4	54	53	53	**52.9**	52.7	15	62	53	**54**	52.9	**53.9**
5	28	27	27	27.0	27.0	16	109	103	**104**	103.0	**104.0**
6	82	79	79	78.3	**79.0**	17	97	79	**80**	78.1	**79.4**
7	33	28	28	**27.8**	27.4	18	65	53	53	53.0	53.0
8	108	104	104	103.5	103.5	19	126	105	105	**105.0**	104.5
9	32	28	28	27.9	27.9	20	123	**105**	104	**104.6**	104.0
10	63	54	54	53.5	**53.9**	21	97	79	79	**78.9**	78.7
11	109	103	**104**	102.2	**103.5**	22	128	105	105	**104.5**	104.3

Table 3: Best and average integer solutions and upper bound

Knowing that the coefficient of variation never exceeds 3% for every 50-run benchmark on each instance, it appears that ACO and CG-ACO perform similarly, meaning that the CG provides a good selection of variables while removing a majority of them. Such small differences may also be explained by the fact that all SPP variables have unitary weights. In the case of a WSPP, much larger gaps might be observed. An example of such gaps is shown by results presented by Gandibleux et al. [10] on WSPP instances.

Finally, the gap between the upper bound and the best integer solution is very heterogeneous from one instance to another. In the best cases such as instances #1, #2, #4 and #5 the difference is only of one unit, representing a gap of 4%. In those cases, it demonstrates a very good quality of both integer solution and upper bound. Five more instances (#6, #8, #11, #14 and #16) present a gap inferior or equal to 5%. In eight cases, the gap is between 10% and 20%, whereas in the five last cases it is above 20%. These observations can be interpreted as the result of a varying tightness of the SPP continuous relaxation.

4.3 Effects of Pheromone Initialization

A non-neglectable number of variables may have an initial pheromone level of 0 as a consequence of the CG procedure optimal continuous solution. Additional tests were run to check whether or not this phenomemon had a negative consequence on the integer value found by ACO, by initializing all pheromone values to 1 in order to avoid discarding any variable. No significant difference on the CG-ACO integer solution was observed between the two approaches, but removing the pheromone initialization with the continuous solution implicated a slight ACO computational time increase. This initialization consequently seems to be a good information for ACO.

5 Conclusion

We have provided a collaborative sequential hybridization of a CG procedure with an ACO algorithm to solve the SPP applied to the RIC problem. The CG procedure removes the necessity of solving the whole SPP formulation by providing a limited set of promising variables to ACO. It yields a computational time reduction up to 85% without sacrificing the quality of the solutions and provides at the same time a potentially valuable upper bound.

These results show that CG can be a good mean to provide a good subset of variables to a metaheuristic. Moreover, the continuous solution can be exploited as illustrated by the pheromone initialization. Results also show that the performance highly depends on the CG efficiency, which often means that problem-specific data have to be taken into account.

To further assess the benefit of CG as a preprocessing of ACO for the SPP, experiments will be made on more SPP instances, including WSPP instances of the RIC problem. Causal relationships must be identified between the SPP structure on the one side and computational time and gap to the upper bound on the other side. Reducing this gap can be envisaged by using cut generation algorithms as well as trying to improve integer solutions by building solutions with variables not generated by the CG procedure. Finally, the continuous solution could be differentiated from the ACO pheromone vector, so as to take a clearer role of static heuristic information as defined by Lessing et al. [13].

References

[1] B. Alidaee, G. Kochenberger, K. Lewis, M. Lewis, and H. Wang. A new approach for modeling and solving set packing problems. *European Journal of Operational Research*, 186(2):504–512, 2008.

[2] C. Blum, M.J.B. Aguilera, A. Roli, and M. Sampels, editors. *Hybrid Metaheuristics: An Emerging Approach to Optimization*, volume 114 of *Studies in Computational Intelligence*. Springer, 2008.

[3] D.P. Clements, J.M. Crawford, D.E. Joslin, G.L. Nemhauser, M.E. Puttlitz, and M.W.P. Savelsbergh. Heuristic Optimization: A hybrid AI/OR approach. In *Workshop on Industrial Constraint-Directed Scheduling*, 1997.

[4] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed Optimization by Ant Colonies. In *First European Conference on Artificial Life (ECAL), Paris, France*, pages 134–142, 1991.

[5] X. Delorme, X. Gandibleux, and J. Rodriguez. GRASP for set packing problems. *European Journal of Operational Research*, 153(3):564–580, 2004.

[6] K.F. Doerner, W.J. Gutjahr, R.F. Hartl, C. Strauss, and C. Stummer. Pareto ant colony optimization with ILP preprocessing in multiobjective project portfolio selection. *European Journal of Operational Research*, 171(3):830–841, 2006.

[7] I. Dumitrescu and T. Stützle. Combinations of Local Search and Exact Algorithms. In *Applications of Evolutionary Computing*, volume 2611 of *Lecture Notes in Computer Science*, pages 57–68. 2003.

[8] I. Elhallaoui, D. Villeneuve, F. Soumis, and G. Desaulniers. Dynamic aggregation of set-partitioning constraints in column generation. *Operations Research*, 53(4):632–645, 2005.

[9] G.R. Filho and L.A.N. Lorena. Constructive Genetic Algorithm and Column Generation: an Application to Graph Coloring. In *Fifth Conference of The Association of Asian-Pacific Operations Research Societies Within IFORS, Singapore*, July 2000.

[10] X. Gandibleux, J. Jorge, S. Angibaud, X. Delorme, and J. Rodriguez. An ant colony optimization inspired algorithm for the set packing problem with application to railway infrastructure. In *Proceedings of the Sixth Metaheuristics International Conference (MIC2005), Vienna, Austria*, pages 390–396, 2005.

[11] X. Gandibleux, J. Jorge, X. Delorme, and J. Rodriguez. An ant algorithm for measuring and optimizing the capacity of a railway infrastructure. In N. Monmarché, F. Guinand, and P. Siarry, editors, *Artificial Ants*, volume 1, chapter 9. ISTE Ltd and John Wiley & Sons Inc, 2010.

[12] S. Huston, J. Puchinger, and P. Stuckey. The core concept for 0/1 integer programming. In *Proceedings of the fourteenth symposium on Computing: the Australasian theory - Volume 77*, CATS '08, pages 39–47, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc.

[13] L. Lessing, I. Dumitrescu, and T. Stützle. A Comparison Between ACO Algorithms for the Set Covering Problem. In *Ant Colony, Optimization and Swarm Intelligence*, volume 3172 of *Lecture Notes in Computer Science*, pages 105–122. Springer, 2004.

[14] V. Maniezzo. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS Journal on Computing*, 11(4):358–369, 1999.

[15] A. Merel, X. Gandibleux, and S. Demassey. Assessing Railway Infrastructure Capacity by Solving the Saturation Problem with an Improved Column Generation Algorithm. In *Fourth International Seminar on Railway Operations Modelling and Analysis (RailRome 2011), Rome, Italy*, page 20, February 2011.

[16] A. Merel, X. Gandibleux, S. Demassey, and R. Lusby. An improved Upper Bound for the Railway Infrastructure Capacity Problem on the Pierrefitte-Gonesse Junction. In *Dixième congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision, Nancy, France*, pages 62–76, 2009.

[17] J. Puchinger. *Combining Metaheuristics and Integer Programming for Solving Cutting and Packing Problems*. PhD thesis, Vienna University of Technology, January 2006.

[18] J. Puchinger, G.R. Raidl, and M. Gruber. Cooperating Memetic and Branch-and-Cut Algorithms for Solving the Multidimensional Knapsack Problem. In *Proceedings of the Sixth Metaheuristics International Conference (MIC2005)*, pages 775–780, 2005.