COLUMN GENERATION APPROACHES FOR THE SOFTWARE CLUSTERING PROBLEM

Hugo Harry Kramer, Eduardo Uchoa
Departamento de Engenharia de Produção - Universidade Federal Fluminense
Rua Passo da Pátria, 156, Bloco E, 4º andar, São Domingos, 24210-240, Niterói, RJ
hugoharry@gmail.com, uchoa@producao.uff.br

Marcia Fampa
COPPE, Universidade Federal do Rio de Janeiro
CP 68530, 21945-970, Rio de Janeiro, RJ
fampa@cos.ufrj.br

Viviane Köhler
Universidade Federal de Santa Maria
CTISM, Santa Maria, Rio Grande do Sul, RS
viviane@redes.ufsm.br

François Vanderbeck
Institut de Mathématiques de Bordeaux, Université de Bordeaux & Inria Bordeaux Sud-Ouest
351 Cours de la Libération, 33405, Talence Cedex, France
fv@math.u-bordeaux1.fr

ABSTRACT

This work presents the application of branch-and-price approaches to the automatic version of the Software Clustering Problem. To tackle this problem, we apply the Dantzig-Wolfe decomposition to a formulation from literature. Given this, we present two Column Generation (CG) approaches to solve the linear programming relaxation of the resulting reformulation: the standard CG approach, and a new approach, which we call Staged Column Generation (SCG). Also, we propose a modification to the pricing subproblem that allows to add multiple columns at each iteration of the CG. We test our algorithms in a set of 45 instances from the literature. The proposed approaches were able to improve the literature results solving all these instances to optimality. Furthermore, the SCG approach presented a considerable performance improvement regarding computational time, number of iterations and generated columns when compared with the standard CG as the size of the instances grow.

KEY WORDS. Software Clustering Problem, Column Generation, Branch-and-Price.

Main area: PM - Mathematical Programming

1. Introduction

In this work we deal with the Software Clustering Problem (SCP), a problem that arises in the context of Software Engineering. Many software systems have to be modified to cope with some demands as, for instance, performance improvement, addition of new capabilities, bug fixing, extension to new platforms, among others. In order to implement these modifications when the system is too big, it is practical to partition it into subsystems which will be distributed to development teams. One issue resulting from such distribution comes from the fact that a modification done by a team can impact the work of other teams. Therefore, it is important to look into ways on how to obtain a partition where the subsystems are highly intra-connected and loosely inter-connected, which in turn will make the system easier to understand, to maintain and to modify.

The problem of determining such subsystems can be seen as a clustering problem, or a graph partitioning problem, which is NP-Hard (Garey and Johnson, 1979). To do so, we rely on the assumption that a source code can be transformed into a language-independent directed graph (Mitchell, 2002), and such graph is called a Modular Dependency Graph (MDG). The objective is to partition the MDG in clusters, where each cluster represents a subsystem, and the quality of the partition must be maximized. The quality of a partition is defined by a Modularization Quality (MQ), which measures the trade-off between inter and intra-connectivity. The definitions of inter and intra-connectivity, MDG, and MQ will be given in Section 2.

Although software clustering has been used for some years before, its goals were first discussed by Gauthier and Pont (1970), and the suggestion of criteria to achieve them were formalized by Parnas (1972).

When software clustering is considered as an optimization problem, most of the works are based on heuristic search algorithms. In Mancoridis *et al*. (1998), the authors dealt with the software clustering as an optimization problem for the first time and present the Bunch clustering tool, which contains an exhaustive enumeration procedure, a Hill Climbing algorithm and a Genetic Algorithm (GA). This work is continued by Doval *et al*. (1999) where the authors address the problem by means of a GA, and by Mancoridis *et al*. (1999) where new features are added to Bunch. These three papers consider a MDG as input and a MQ that measures the trade-off between intra and interconnectivity. A new MQ measure is introduced in Mitchell and Mancoridis (2002), where it is defined as the sum of the Cluster Factors (CF) of the clusters in the partition of MDG. These works are summarized in the PhD thesis of Mitchell (2002). His GA is improved in the DAGC algorithm presented by Parsa and Bushehrian (2005). An algorithm based on learning automata and another one that combines learning automata with a genetic algorithm are proposed by Mamaghani and Meybodi (2009). These algorithms are compared with DAGC and Bunch's Hill Climbing algorithm, and the authors claim to achieve faster convergence to good solutions and more success on avoiding local optimal solutions. In Mahdavi *et al*. (2003b) and Mahdavi *et al*. (2003a) a parallel Hill Climbing algorithm is proposed. A survey on search based algorithms for the problem can be found in Räihä (2010), where the author lists other approaches in which the input is not a MDG and other MQs are used. As far as we know, there are only two works dealing with the SCP with exact algorithms. The first one is the exhaustive enumeration procedure present in Bunch. The other one by Köhler *et al*. (2013), presents some mathematical formulations for the problem, as well as a preprocessing procedure and valid inequalities.

The remainder of the paper is outlined as follows: in Section 2, the formal definitions of MDG, the used MQ and the SCP are given. Section 3 presents the compact mathematical formulation for the problem found in Köhler *et al*. (2013), and a reformulation obtained by means of Dantzig-Wolfe decomposition (Dantzig and Wolfe, 1960). The CG approaches to solve this reformulation are detailed in Section 4. The computational experiments performed to evaluate the proposed approaches in comparison with those from the literature are described and discussed in Section 5. Lastly, Section 6 presents the conclusions of the work.

2. Problem definition

In this problem, one has a directed MDG $G(V, E)$ consisting of a set $V = \{1, ..., n\}$ of nodes representing the modules of a software and a set $E \subseteq \{(u, v)|u, v \in V\}$ representing the relationships between modules. Each relationship $(u, v) \in E$ has an associated positive weight given by c_{uv} . The goal is to find a partition of V into non-empty clusters $V_1, ..., V_K$ such that

$$\bigcup_{k=1}^{K} V_k = V, \qquad V_k \cap V_l = \emptyset \qquad (k, l = 1, \ldots, K, k \neq l). \tag{1}$$

When the number of clusters K is already given as part of the input, the clustering is called *non-automatic*. Otherwise, when the best value of K must be discovered by the method, the clustering is called *automatic*. The SCP consists in finding a partition with maximal quality. Firstly, before defining the quality measures of a partition, the definitions of intra and inter-connectivity must be given. The intra-connectivity of a cluster measures how such cluster is cohese or dense, i.e., how much the components of the cluster are connected. The intra-connectivity A_k of cluster k is given by:

$$A_k = \frac{\mu_k}{|V_k|^2}, \tag{2}$$

where μ_k is the number of edges with both ends inside cluster k, and $|V_k|^2$ is the maximum possible number of edges inside cluster k, including loop edges. The inter-connectivity between two different clusters measures how they are connected, i.e., how much the modules of these clusters are related. The inter-connectivity $E_{k,l}$ between two clusters k and l is defined as follows:

$$E_{k,l} = \begin{cases} 0, & \text{if } k = l \\ \dfrac{\varepsilon_{k,l}}{2|V_k||V_l|}, & \text{if } k \neq l, \end{cases} \tag{3}$$

where $\varepsilon_{k,l}$ is the number of arcs with an end in cluster k and the other end in cluster l, and $2|V_k||V_l|$ is the maximum possible number of arcs between those clusters.

Given these definitions, a first partition quality measure, called BasicMQ (Mancoridis *et al.*, 1998), is defined as the difference between the average intra-connectivity and the average inter-connectivity, given by the following expression:

$$BasicMQ = \begin{cases} \dfrac{1}{K}\displaystyle\sum_{k=1}^{K} A_k - \frac{1}{|P|}\sum_{(k,l)\in P} E_{k,l}, & \text{if } K > 1 \\ A_1, & \text{if } K = 1, \end{cases} \tag{4}$$

where K is the number of clusters in the partition and P is the set of pairs of clusters $\{(k, l)|k, l = 1, \ldots, K, k \neq l\}$ in the partition. The problem with BasicMQ is that it can not be used to measure the quality of partitions obtained from graphs with weighted arcs. To overcome this drawback, Mitchell (2002) proposed a new MQ, called TurboMQ, which is defined as follows.

$$\text{TurboMQ} = \sum_{k=1}^{K} CF_k. \tag{5}$$

According to the expression above, the quality of a partition is given by the sum of the cluster factors of the clusters that composes the partition. The cluster factor CF_k of a cluster k is defined by:

$$CF_k = \begin{cases} 0, & \text{if cluster } k \text{ is empty} \\ \dfrac{\mu_k}{\mu_k + \frac{1}{2}\varepsilon_k}, & \text{otherwise,} \end{cases} \tag{6}$$

where μ_k is the sum of weights of arcs with both ends inside cluster k, and ε_k is the sum of weights of the arcs with exactly one end in cluster k. In the present work, as well as in Köhler *et al*. (2013), TurboMQ is considered as the partition quality measure to be maximized.

3. Mathematical formulations

The SCP can be formulated as a Mixed Integer Linear Programming (MILP) problem. The first application of mathematical models to the problem can be found in Köhler *et al*. (2013). In such work, authors present three formulations, where two of them are MILPs which derive from the application of linearization procedures over the other formulation, where the objective function is a Sum of Linear Fractional Functions (SOLF), which is non-linear and non-convex. The formulation presented in the following is the one obtained by means of the linearization procedure proposed by Billionet and Djebali (2006).

Let $K' = \{1, \ldots, K_{max}\}$ be the set of possible clusters, K_{max} should be set as an upper bound on the maximum number of clusters in the optimal partition. Binary variables x_u^k indicate if a node $u \in V$ is assigned to a cluster $k \in K'$ or not. Binary variables t_{uv}^k indicate whether an edge $(u, v) \in E$ is completely inside a cluster $k \in K'$. Continuous variables r_k represent the Cluster Factor of cluster $k \in K'$. Such variables, along with continuous variables s_u^k result from the linearization procedure, where $s_u^k = r^k x_u^k$ and

$$r^k = \begin{cases} 0, & \text{if cluster } k \text{ is empty,} \\ \dfrac{2\sum_{(u,v)\in E} c_{uv} t_{uv}^k}{\sum_{(u,v)\in E} c_{uv}(x_u^k + x_v^k)}, & \text{otherwise.} \end{cases} \tag{7}$$

Therefore, the problem can be modeled as the following mathematical formulation $\mathcal{F}_1$.

$$(\mathcal{F}_1) \max \sum_{k \in K'} r_k \tag{8}$$

subject to

$$\sum_{k \in K'} x_{uk} = (\leq) 1 \qquad \forall u \in V \tag{9}$$

$$t_{uv}^k \leq x_u^k \qquad \forall (u,v) \in E, k \in K' \tag{10}$$

$$t_{uv}^k \leq x_v^k \qquad \forall (u,v) \in E, k \in K' \tag{11}$$

$$t_{uv}^k \geq x_u^k + x_v^k - 1 \qquad \forall (u,v) \in E, k \in K' \tag{12}$$

$$r^k \leq \sum_{u \in V} x_u^k \qquad \forall k \in K' \tag{13}$$

$$s_u^k \leq r^k \qquad \forall u \in V, k \in K' \tag{14}$$

$$s_u^k \leq x_u^k \qquad \forall u \in V, k \in K' \tag{15}$$

$$s_u^k \geq r^k + x_u^k - 1 \qquad \forall u \in V, k \in K' \tag{16}$$

$$\sum_{(u,v)\in E} c_{uv}(s_u^k + s_v^k) = 2 \sum_{(u,v)\in E} c_{uv} t_{uv}^k \qquad \forall k \in K' \tag{17}$$

$$0 \leq r^k \leq 1 \qquad \forall k \in K' \tag{18}$$

$$0 \leq s_u^k \leq 1 \qquad \forall u \in V, k \in K' \tag{19}$$

$$0 \leq t_{uv}^k \leq 1 \qquad \forall (u,v) \in E, k \in K' \tag{20}$$

$$x_u^k \in \{0,1\} \qquad \forall u \in V, k \in K'. \tag{21}$$

The objective function (8) aims at maximizing the quality of the partition. Constraints (9) assure that each node is assigned to only one cluster. Constraints (10)–(12) guarantee that an edge is inside cluster k only if both nodes linked by it are inside the same cluster. Constraints (13) and (17) define r^k, while constraints (14)–(16) define the relationship between variables r^k and x_u^k given by s_u^k variables. Constraints (18)–(21) define the variables domains.

This formulation suffers from symmetry issues and provides a weak linear programming relaxation dual bound, which is equal to the number of nodes of the instance, resulting in a bad performance when one tries to solve it by means of a MILP solver. Aiming at the improvement of this dual bound and the symmetry elimination, the authors proposed some cuts and valid inequalities.

3.1. Dantzig-Wolfe Decomposition

Let Q be the set of all feasible clusters, where a cluster $q \in Q$ consists of a subset of nodes in V. A binary variable λ_q is used to decide if a cluster $q \in Q$ is part of the solution. The constant a_{vq} takes value 1 if node $v \in V$ is in cluster $q \in Q$ and 0 otherwise. The cluster factor C_q of cluster q is as the definition given in Section 2. The following reformulation aims at selecting a set of feasible clusters which maximizes the sum of cluster factors.

$$(\mathcal{DWM}) \max \sum_{q\in Q} C_q \lambda_q \tag{22}$$

subject to

$$\sum_{q\in Q} a_{vq}\lambda_q = (\leq)1 \qquad \forall v \in V \tag{23}$$

$$\lambda_q \in \{0,1\} \qquad \forall q \in Q \tag{24}$$

The columns in this formulation are associated to the set of all feasible clusters. As the size of instances grow, the number of columns in this model becomes too large, and it cannot be solved directly as it relies on the enumeration of all feasible clusters. To overcome this issue, we propose to solve it by branch-and-price, where the Linear Programming relaxation of $\mathcal{DWM}$ is solved by CG at each node of the branch-and-bound tree. In the CG algorithm, a restricted version of $\mathcal{DWM}$ with a few columns is initially considered and new columns are added by solving a pricing subproblem until the addition of new columns are no more necessary. The pricing subproblem will be presented in the next subsection, while the proposed CG algorithms will be detailed in Section 4.

3.2. Pricing Subproblem

Let $\pi_v, v \in V$ be the dual variables associated to constraints (23). Given the values of these dual variables, the pricing subproblem $\mathcal{SP}$ is:

$$(\mathcal{SP}) \max r - \sum_{u\in V} \pi_u x_u \tag{25}$$

subject to

$$t_{uv} \leq x_u \qquad \forall (u,v) \in E \tag{26}$$
$$t_{uv} \leq x_v \qquad \forall (u,v) \in E \tag{27}$$
$$t_{uv} \geq x_u + x_v - 1 \qquad \forall (u,v) \in E \tag{28}$$
$$r \leq \sum_{u \in v} x_u \tag{29}$$
$$s_u \leq r \qquad \forall u \in V \tag{30}$$
$$s_u \leq x_u \qquad \forall u \in V \tag{31}$$
$$r + x_u - s_u \leq 1 \qquad \forall u \in V \tag{32}$$
$$\sum_{(u,v) \in E} c_{uv}(s_u + s_v) = \sum_{(u,v) \in E} 2c_{uv}t_{uv} \tag{33}$$
$$t_{uv} \in \{0,1\} \qquad \forall (u,v) \in E \tag{34}$$
$$x_u \in \{0,1\} \qquad \forall u \in V \tag{35}$$
$$0 \leq s_u \leq 1 \qquad \forall u \in V \tag{36}$$
$$0 \leq r \leq 1. \tag{37}$$

The solution of $\mathcal{SP}$ is a cluster whose cluster factor is r. It represents a feasible column to $\mathcal{DWM}$ with reduced cost given by the objective function (25). Constraints (26)–(37) have the same meaning as in $\mathcal{F}_1$.

Furthermore, this pricing subproblem can be modified in order to make the CG algorithm capable of adding multiple columns per iteration. This is done by the addition of two constraints. The first one fixes one node $w \in V$ to the solution, and the second one forbids the nodes $v \in V$, such that $v < w$, to appear in such solution. Then, when pricing at each iteration of the CG, a pricing subproblem is solved for each fixed node w. Thus, to diversify the search in the column generation process, this pricing subproblem takes the following form:

$$(\mathcal{SP}_w) \max r - \sum_{u \in V} \pi_u x_u \tag{38}$$

subject to

$$x_w = 1 \tag{39}$$
$$x_v = 0 \qquad \forall v \in V,\ v < w \tag{40}$$
$$(25) - (37)$$

4. Column Generation Approaches

This section describes the proposed CG approaches to deal with SCP. Due to the prohibitive use of formulation $\mathcal{DWM}$ within a MILP solver when instances grow, the CG algorithms are used to solve its linear programming relaxation within a branch-and-price procedure.

4.1. Standard Column Generation

The Standard Column Generation algorithm proposed for the SCP is detailed in the Alg. 1 that follows. Alg. 1(a) is the Standard CG algorithm where only one column can be added at each iteration. It begins with the initialization of the Restricted Master Problem (RMP) with a subset of the columns of the linear programming relaxation of $\mathcal{DWM}$ (with artificial columns). At each iteration, the RMP is solved and the dual variables π_v of constraints (23) are obtained. The pricing subproblem $\mathcal{SP}$ is then fed with these dual variables and solved to optimality by a MILP solver. The solution of $\mathcal{SP}$ represents a cluster, which is a column of $\mathcal{DWM}$ with reduced cost

given by objective function (25). If the reduced cost is positive, this column will be added to RMP and the algorithm proceeds to the next iteration. At a given iteration, if the solution of the pricing subproblem provides a column with non-positive reduced cost, it means that this column will not be added to RMP. Furthermore, it means that the addition of columns to RMP is no more necessary and the optimal solution for the linear programming relaxation of $\mathcal{DWM}$ was found. Then, the algorithm stops returning such solution.

```
1: procedure SINGLECOLSTANDARDCG(RMP)
2:     sol ← ∅
3:     sol_SP ← ∅
4:     Initialize RMP
5:     colAdded ← true
6:     while colAdded do
7:         sol ← LPsolver(RMP)
8:         colAdded ← false
9:         Update SP with dual solution values π
10:        sol_SP ← MILPsolver(SP(π))
11:        if f(sol_SP) > 0 then
12:            Add column associated with sol_SP to RMP
13:            colAdded ← true
14:        end if
15:    end while
16:    return sol
17: end procedure
```

(a) Standard CG adding one column per iteration

```
1: procedure MULTICOLSSTANDARDCG(RMP)
2:     sol ← ∅
3:     sol_SP_w ← ∅
4:     Initialize RMP
5:     colAdded ← true
6:     while colAdded do
7:         sol ← LPsolver(RMP)
8:         colAdded ← false
9:         Update SP with dual solution values π
10:        for each node w ∈ V do
11:            sol_SP_w ← MILPsolver(SP_w(π))
12:            if f(sol_SP_w) > 0 then
13:                Add column associated with sol_SP_w to RMP
14:                colAdded ← true
15:            end if
16:        end for
17:    end while
18:    return sol
19: end procedure
```

(b) Standard CG adding multiple columns per iteration

Alg. 1: Standard Column Generation algorithms

In Alg. 1(b), the version of the Standard CG algorithm where multiple columns can be added to RMP at each iteration is presented. The only difference consists in the pricing, where a subproblem $\mathcal{SP}_w$ is solved for each node $w \in V$. Thus, at each iteration of this algorithm, at most n columns can be added to RMP.

4.2. Staged Column Generation

It can be noticed that the Standard CG algorithm proposed for the problem in the previous subsection has its performance highly dependent on the performance of the MILP solver used in the pricing. Therefore, in order to alleviate the computational burden of the Standard CG, a new CG approach is proposed, and is outlined in the following Alg. 2.

As well as the Standard CG algorithms, two versions of the Staged CG algorithm are proposed. The first one adds only one column per iteration and is detailed in Alg. 2(a). It begins with stage s equal to 1, and an enumeration procedure provides a list with all connected clusters containing up to $maxStages - 1$ nodes. At the next step, the RMP is initialized as defined previously in Alg. 1. At each iteration, while $s < maxStages$ (which we call heuristic phase), a solution is obtained for the subproblem by searching for a cluster having at most s nodes with the best reduced cost from the list of clusters. If the reduced cost of this column is positive, it will be added to RMP and the algorithm goes to the next iteration. Otherwise, the column is not added and the stage s is incremented. When the stage s of the algorithm reaches $maxStages$, the heuristic phase is finished and the remaining pricing subproblems are solved to optimality by a MILP solver. Therefore, the iterations are performed in the same way as in the Standard CG until convergence, returning the optimal solution of the linear programming relaxation of $\mathcal{DWM}$.

```
1: procedure SINGLECOLSTAGEDCG(RMP, maxStages)
2:     s ← 1
3:     sol ← ∅
4:     sol_SP ← ∅
5:     clusterList ←EnumProcedure(MDG, maxStages − 1)
6:     Initialize RMP
7:     colsAdded ← true
8:     while colsAdded do
9:         sol ← LPSolver (RMP)
10:        colsAdded ← false
11:        Update SP with values of π
12:        while s < maxStages do
13:            sol_SP ← FindBestCluster(clusterList, s)
14:            if f(sol_SP) > 0 then
15:                Add column associated with sol_SP to RMP
16:                colsAdded ← true
17:            else
18:                s ← s + 1
19:            end if
20:        end while
21:        sol_SP ← MILPSolver (SP(π))
22:        if f(sol_SP) > 0 then
23:            Add column associated with sol_SP to RMP
24:            colsAdded ← true
25:        end if
26:    end while
27:    return sol
28: end procedure
```

(a) Staged CG adding one column per iteration

```
1: procedure MULTICOLSSTAGEDCG(RMP, maxStages)
2:     s ← 1
3:     sol ← ∅
4:     sol_SP ← ∅
5:     clusterList ←EnumProcedure(MDG, maxStages − 1)
6:     Initialize RMP
7:     colsAdded ← true
8:     while colsAdded do
9:         sol ← LPSolver (RMP)
10:        colsAdded ← false
11:        Update SP with values of π
12:        while s < maxStages do
13:            for each node w ∈ V do
14:                sol_SP_w ← FindBestCluster (clusterList, s)
15:                if f(sol_SP_w) > 0 then
16:                    Add column associated with sol_SP_w to
   RMP
17:                    colsAdded ← true
18:                end if
19:            end for
20:            if !colsAdded then
21:                s ← s + 1
22:            end if
23:        end while
24:        for each node w ∈ V do
25:            sol_SP_w ← MILPSolver (SP_w(π))
26:            if f(sol_SP_w) > 0 then
27:                Add column associated with sol_SP_w to RMP
28:                colsAdded ← true
29:            end if
30:        end for
31:    end while
32:    return sol
33: end procedure
```

(b) Staged CG adding multiple columns per iteration

Alg. 2: Staged Column Generation algorithms

The version of the algorithm that allows the addition of multiple columns per iteration detailed in Alg. 2(b) differs only in the pricing from Alg. 2(a). Now, in the heuristic phase, the cluster with best reduced cost found from the list also must satisfy constraints (39) and (40). In the last stage, the pricing is performed in the same way as in Alg. 1(b).

4.3. Branching

The branching scheme in the branch-and-price algorithm is as follows: after solving each node of the branch-and-bound tree using one of the CG algorithms presented, two child nodes are created. At each of these new nodes, disjunctive branching constraints are added to RMP in the form presented by Vanderbeck (2011), which generalizes the Ryan and Foster (1981) scheme. For the SCP, these disjunctive constraints are:

$$\sum_{q \in \hat{Q}} \lambda_q \leq 0 \quad \text{or} \quad \sum_{q \in \hat{Q}} \lambda_q \geq 1, \tag{41}$$

where $\hat{Q}$ is a subset of columns of RMP in which, for two nodes $u, v \in V$, in which $a_{uq} = 1$ and $a_{vq} = 1$. Thus, by adding the disjunctive constraint in the left, λ_q variables with $q \in \hat{Q}$ are forbidden in the solution of RMP, i.e., in such solution, nodes u and v can not belong to the same cluster. By adding the constraint in the right, at least one λ_q variable with $q \in \hat{Q}$ must be in the solution of RMP, which means that in this solution, nodes u and v must be in the same cluster.

After the root node, when using one of algorithms in Alg. 2, the stage parameter s is not reset to 1. This is equivalent to say that, after the root node, each node in the branch-and-bound tree is solved by one of algorithms in Alg. 1, and the pricing subproblems solutions are obtained by a MILP solver.

5. Computational experiments

In order to perform the computational experiments, we consider a set of 45 instances from the literature, where 15 of these instances were proposed by Mitchell (2002). In these instances, there are no weighted edges in the MDGs. The remaining 30 instances were proposed by Mamaghani and Meybodi (2009) with all MDGs originally containing weighted edges. Furthermore, for all the instances, we consider the reduced MDGs resulting from the application of the preprocessing technique proposed by Köhler *et al*. (2013). The preprocessed instances were provided by the authors of the latter work.

The CG approaches proposed in the previous sections were implemented in C++, using the BaPCod framework[1], and Boost C++ libraries 1.49.0.1. The LP and MILP solver used was the IBM ILOG CPLEX 12.1 64 bits. All executions were performed in a PC Intel Core i5-3210M 2.5 GHz with 6 GB RAM, running under Ubuntu 13.04 64 bits linux OS, kernel 3.8.0-30-generic (x86_64). Only 1 CPU thread was used, a time limit of 600 seconds is imposed to the MILPs, the number of iterations in the CG algorithms is limited to 5000, and the maximum number of stages considered in SCG is 7. Such maximum number of stages was choosen by observing that if one wants to enumerate all connected clusters with more than 6 nodes, the time needed would impact negatively in the overall performance of the SCG algorithms.

We divided the instances in three different groups according to the number of nodes, after preprocessing. The *Small Instances* group contains 15 instances with at most 15 nodes. The 17 *Medium Instances* have MDGs from 16 to 35 nodes. Lastly, the *Larger Instances* group has 13 instances ranging from 36 to 55 nodes. Table 3 shows the comparison of our approaches with the literature in terms of average results. In this table, the second and third columns show the literature results for the exhaustive procedure of the Bunch tool (Mitchell, 2002) and the best MILP approach of Köhler *et al*. (2013), as shown in the latter work. The last four columns show the results found by our approaches, where the fourth and fifth columns contain the results of the Standard CG adding a single column and multiple columns per iteration, respectivelly, and the same results are shown for the Staged CG approach in the sixth and seventh columns.

Comparing the proposed approaches with the literature, one can notice that the results are notably improved in terms of computational time and number of instances solved. Our approaches were able to solve all 45 instances, where 43 of them were solved to optimality at the root node, an improvement of 11 instances when compared with the best results of Köhler *et al*. (2013). Regarding the average computational time, even the straightforward Standard CG adding one column per iteration was able to outperform the literature approaches in the three instance groups.

Observing the results for the CG approaches, it is observed that, for this problem, the pricing in two phases (Staged CG) results in a reasonable improvement in the number of iterations, generated columns and computational time when compared to the Standard CG. Also, the modification made to allow the addition of multiple columns per iteration contributes to obtain better results, specially concerning the number of iterations. In this aspect, the best average results for the three instance groups were found by the Standard CG approach adding multiple columns per iteration. When comparing the average number of generated columns and the average computational time, the best results were obtained by the Staged CG approach adding a single column per iteration.

[1]More info at `https://realopt.bordeaux.inria.fr/?page_id=2`.

Table 3: Average results

	Literature [1]		This work			
	Exhaustive	MILP	Std. CG		Stg. CG	
	(Mitchell, 2002)	(Köhler et al, 2013)	SingleCol	MultiCols	SingleCol	MultiCols
						Small Instances
Avg. iters	–	–	28	**10**	23	14
Avg. columns	–	–	27	35	**18**	23
Avg. time (s)	5732.33	0.60	1.38	0.63	**0.05**	0.08
Inst. solved	8	**15**	**15**	**15**	**15**	**15**
						Medium Instances
Avg. iters	–	–	134	**24**	75	38
Avg. columns	–	–	133	154	**67**	99
Avg. time (s)	11000.00	761.92	73.48	10.98	**3.09**	4.49
Inst. solved	0	16	**17**	**17**	**17**	**17**
						Larger Instances
Avg. iters	–	–	486	**31**	128	44
Avg. columns	–	–	485	406	**121**	179
Avg. time (s)	11000.00	5649.35	1673.44	75.06	**52.27**	95.31
Inst. solved	0	3	**13**	**13**	**13**	**13**

[1]: Intel Xeon 2.67 GHz, 24 GB RAM, Suse Linux, CPLEX 12.2, 11000 s of time limit

For the two instances that were not solved at the root node, the branch-and-price algorithm needed only one branch to solve the instance ciald and two branches to solve the instance star. The gap between the root node relaxation and the optimal solution of instance ciald was 0.06%, and 0.23% for instance star.

6. Conclusions

In this work, a new formulation was obtained for the SCP by means of the Dantzig-Wolfe decomposition which is solved by a branch-and-price method. To solve the linear programming relaxation of this formulation at each node of the branch-and-bound tree, two CG approaches are proposed, a Standard one and a Staged approach. As it can be observed from the computational experiments, the proposed approaches were able to notably improve the literature results, by solving all 45 test instances available.

Also, a reasonable improvement is achieved when comparing the Standard and Staged CG algorithms in terms of average iterations, number of columns generated and computation time. The improvement on the number of iterations and generated columns can be credited to the fact that, in the beginning of a CG algorithm, the information given by the dual variables is somewhat irrelevant. Thus, in the Standard CG these dual variables misleads the pricing subproblem to find columns which are very unlikely to be present in the final solution. In general, the columns that will be present in the final solution are added at the last iterations, when the dual variables are stabilized and their information is more relevant for the pricing of new columns. Also, the size of the clusters found in the heuristic phase are very similar to those that will be in the final solution. In addition, at the end of the heuristic phase, the dual variables are reasonably stable and then provide a good information so the columns priced out by the MILP solver in the last stage are more suitable to appear in the final solution.

The Stage CG approach seems to have potential to provide the same improvements in other problems, specially when an efficient oracle is not available to exactly solve the pricing subproblems. Also, the heuristic phase can be seen as a stabilization phase, and the computationally

expensive exact pricing subproblem will be used only when the dual variables give good information on how the columns should look.

References

Billionet, A. and Djebali, K. (2006), Résolution d'un problème combinatoire fractionnaire par la programmation linéaire mixte. *RAIRO. Recherche opérationnelle*, v. 40, n. 2, p. 97–111.

Dantzig, G. B. and Wolfe, P. (1960), Decomposition principle for linear programs. *Operations research*, v. 8, n. 1, p. 101–111.

Doval, D., Mancoridis, S. and Mitchell, B. S. Automatic clustering of software systems using a genetic algorithm. *Proceedings of the Software Technology and Engineering Practice*, p. 73–81. IEEE, 1999.

Garey, M. R. and Johnson, D. S. *Computers and intractability: a guide to the theory of NP-completeness*. Freeman, 1979.

Gauthier, R. and Pont, S. *Designing Systems Programs*. Prentice-Hall, 1970.

Köhler, V., Fampa, M. and Araújo, O. (2013), Mixed-integer linear programming formulations for the software clustering problem. *Computational Optimization and Applications*, v. 55, n. 1, p. 1–23.

Mahdavi, K., Harman, M. and Hierons, R. (2003a), Finding building blocks for software clustering. *Lecture Notes in Computer Science*, v. 2724, p. 2513–2514.

Mahdavi, K., Harman, M. and Hierons, R. M. A multiple hill climbing approach to software module clustering. *Proceedings of the International Conference on Software Maintenance*, p. 315–324. IEEE, 2003b.

Mamaghani, A. S. and Meybodi, M. R. Clustering of software systems using new hybrid algorithms. *Proceedings of the 2009 IEEE International Conference on Computer and Information Technology (CIT'09)*, volume 1, p. 20–25, 2009.

Mancoridis, S., Mitchell, B. S., Rorres, C., Chen, Y. and Gansner, E. R. Using automatic clustering to produce high-level system organizations of source code. *Program Comprehension, 1998. IWPC'98. Proceedings., 6th International Workshop on*, p. 45–52. IEEE, 1998.

Mancoridis, S., Mitchell, B. S., Chen, Y. and Gansner, E. R. Bunch: A clustering tool for the recovery and maintenance of software system structures. *Proceedings of the IEEE International Conference on Software Maintenance*, p. 50–59. IEEE, 1999.

Mitchell, B. S. *A heuristic search approach to solving the software clustering problem*. PhD thesis, Drexel University, 2002.

Mitchell, B. S. and Mancoridis, S. Using heuristic search techniques to extract design abstractions from source code. *Proceedings of the Genetic and Evolutionary Computation Conference*, p. 1375–1382. Morgan Kaufmann Publishers Inc., 2002.

Parnas, D. L. (1972), On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, v. 15, n. 12, p. 1053–1058.

Parsa, S. and Bushehrian, O. (2005), A new encoding scheme and a framework to investigate genetic clustering algorithms. *Journal of Research & Practice in Information Technology*, v. 37, n. 1.

Räihä, O. (2010), A survey on search-based software design. *Computer Science Review*, v. 4, n. 4, p. 203–249.

Ryan, D. M. and Foster, B. A. (1981), An integer programming approach to scheduling. *Computer scheduling of public transport urban passenger vehicle and crew scheduling*, p. 269–280.

Vanderbeck, F. (2011), Branching in branch-and-price: a generic scheme. *Mathematical Programming*, v. 130, n. 2, p. 249–294.