Journal of Artificial Intelligence Research 62 (2018) 133-153 Submitted 06/17; published 05/18

A COP Model for Graph-Constrained Coalition Formation

Filippo Bistaffa FILIPPO.BISTAFFA@IIIA.CSIC.ES
IIIA-CSIC,
Campus UAB, 08913, Cerdanyola, Catalonia, Spain

Alessandro Farinelli ALESSANDRO.FARINELLI@UNIVR.IT
Computer Science Department, University of Verona,
Strada le Grazie 15, 37134, Verona, Italy

Abstract

We consider Graph-Constrained Coalition Formation (GCCF), a widely studied sub-problem of coalition formation in which the set of valid coalitions is restricted by a graph. We propose COP-GCCF, a novel approach that models GCCF as a COP, and we solve such COP with a highly-parallel approach based on Bucket Elimination executed on the GPU, which is able to exploit the high constraint tightness of COP-GCCF. Results show that our approach outperforms state of the art algorithms (i.e., DyCE and IDP^G) by at least one order of magnitude on realistic graphs, i.e., a crawl of the Twitter social graph, both in terms of runtime and memory.

1. Introduction

Coalition Formation (CF) (Sandholm, Larson, Andersson, Shehory, & Tohmé, 1999) is one of the key approaches to establishing collaborations in multi-agent systems. It involves the coming together of multiple agents in order to achieve either their individual or collective goals. In particular, here we focus on the associated optimisation problem, denoted as Coalition Structure Generation (CSG). The aim of CSG is partitioning the set of agents (into disjoint coalitions) with the objective of maximising the sum of the values of the chosen coalitions, provided by the so-called *characteristic function* (see Section 2.2).

Our work is positioned in a strand of literature, namely *Graph-Constrained Coalition Formation* (GCCF), pioneered by Myerson (1977) and Demange (2004), and later developed by Voice, Polukarov, and Jennings (2012a), Voice, Ramchurn, and Jennings (2012b), and Bistaffa, Farinelli, Cerquides, Rodríguez-Aguilar, and Ramchurn (2017a). GCCF focuses on a specific type of constraints that encodes synergies or relationships among the agents and that can be expressed by a graph, where nodes represent agents and edges encode the relationships between the agents. In this setting, edges enable connected agents to form a coalition and a coalition is considered *feasible* only if its members represent the vertices of a connected subgraph. Such constraints are present in several real-world scenarios, such as social or trust constraints, e.g., energy consumers who prefer to group with their acquaintances in forming energy cooperatives (Bistaffa et al., 2017a), or commuters sharing rides with their friends (Bistaffa, Farinelli, Chalkiadakis, & Ramchurn, 2017b), and physical constraints, e.g., emergency responders may join specific teams in disaster scenarios where only certain routes are available. Unfortunately, State of the Art (SoA) algorithms for GCCF are either characterised by exponential memory requirements (Voice et al., 2012b) or require

some assumptions on the characteristic function (Voice et al., 2012a; Bistaffa et al., 2017a), which limit their application.

Now, GCCF is essentially an optimisation problem (aiming at maximising the sum of the coalitional values) subject to feasibility constraints (i.e., coalitions must be feasible and disjoint). Nonetheless, to the best of our knowledge none of the constraint optimisation techniques in the literature (Dechter, 2003) has ever been applied to GCCF.

Against this background, in this paper we propose COP-GCCF, the first approach that models GCCF as a Constrained Optimisation Problem (COP). COP-GCCF does not require to know the structure of the characteristic function. We remark that, in such general case, GCCF is NP-Hard, hence computing an optimal solution requires a significant computational effort. Nonetheless, exploiting the structure of the characteristic function is often very difficult or impossible, since it could be not known and/or hard to formalise. Moreover, even if the structure of the characteristic function is known, it could be difficult to exploit from the algorithmic point of view. As an example, in the work by Bistaffa et al. (2017b) the structure is well-known, but a complex, ad-hoc technique to compute an upper-bound on the characteristic function is needed to exploit such a structure within the branch-and-bound CSG algorithm. This is far from trivial and not easily applicable to scenarios different from ridesharing. Along these lines, having a solution technique that does not require any property on the characteristic function is important.

To achieve this objective, within COP-GCCF we exploit the *structure of the graph* so as to achieve a model of manageable complexity. Specifically, we propose a COP formalisation that builds a hierarchy of agents resulting in a linear number of constraints (*wrt* the number of agents). COP-GCCF is based on constraints represented as *incomplete* tables (i.e., unfeasible assignments are not represented in memory), so as to exploit the high constraint tightness inherent in GCCF. This allows us to reduce the number of rows in each constraint function from exponential to linear *wrt* the number of variables in the scope. We employ a GPU version of Bucket Elimination (BE) provided by Bistaffa, Bombieri, and Farinelli (2016), since, to the best of our knowledge, it is the only one able to exploit incomplete tables so as to obtain an approach with manageable memory requirements (see Section 3.4). In fact, SoA COP solution algorithms (Marinescu & Dechter, 2007; Fioretto, Le, Pontelli, Yeoh, & Son, 2015) that do not exploit incomplete tables could only solve problems with up to 5 agents in our tests. Our work confirms that BE can be a practical solution approach if the problem is appropriately modelled as a COP, as also shown by Larrosa, Morancho, and Niso (2005). Finally, we exploit highly-parallel architectures (i.e., GPUs). This choice is motivated by the successful use of cloud-based (Malapert, Régin, & Rezgui, 2016) and GPU-based (Greengard, 2016) parallel approaches to speed-up the solution of problems that exhibit a high level of parallelism, particularly in the field of AI.

In more detail, we advance the SoA in the following ways:

- We propose COP-GCCF, the first COP model to solve GCCF, which requires a linear number of constraints to formalise such problem. We establish a new link between GCCF and COPs, which opens a new line of research focusing on the use of COP methods for GCCF.
- We evaluate COP-GCCF both on realistic and synthetic graphs. Results show that our approach does not provide any advantage with dense graphs, but it outperforms

SoA algorithms (i.e., DyCE and IDP^G) on sparse graphs, both in terms of runtime and memory. COP-GCCF is at least one order of magnitude faster than counterpart approaches using Twitter (Kwak, Lee, Park, & Moon, 2010) as a realistic graph topology. Results confirm that, even without exploiting the structure of the characteristic function, COP-GCCF improves upon SoA GCCF solution algorithms, by correctly exploiting the structure of the graph.

2. Background

In what follows, we first define COPs in Section 2.1 and GCCF in Section 2.2. Then, in Section 2.3 we position our work *wrt* the existing literature.

2.1 COPs

A COP is defined upon a Constraint Network (CN), a theoretical model that encodes a knowledge-base theory as several functions or relations over subsets of discrete variables (e.g., clauses for propositional satisfiability, constraints, or conditional probability matrices for belief networks). In this paper we adopt the definitions provided by Dechter (2003).

Definition 1 (constraint network)**.** *A* constraint network *consists of a set $X = \{x_1, \ldots, x_n\}$ of n discrete variables such that $x_1 \in D_1, \ldots, x_n \in D_n$, where D_i represents the domain of the variable x_i, together with a set of m constraints $\{C_1, \ldots, C_m\}$.*

Definition 2 (constraint)**.** *A* constraint *C_i is a relation defined on a set $X_i = \{x_{i_1}, \ldots, x_{i_h}\}$ of h discrete variables, called the* scope *of the constraint, such that $X_i \subseteq X$. Such a relation denotes the variables simultaneous legal assignments. Non-legal assignments are denoted as* unfeasible*.*

A particular CN corresponds to a Constraint Satisfaction Problem (CSP), which can be generalised obtaining a COP.

Definition 3 (constraint satisfaction problem)**.** *Given a CN, the corresponding* constraint satisfaction problem *requires to find a variable assignment $\bar{a}^* = (a_1^*, \ldots, a_n^*)$ satisfying all the constraints in the CN.*

Definition 4 (constraint optimisation problem)**.** *A* constraint optimisation problem *is a CN augmented with a set of functions. Let $F_1, \ldots, F_l$ be l real-valued functional components defined over the scopes $Q_1, \ldots, Q_l$, $Q_i \subseteq X$, let $\bar{a} = (a_1, \ldots, a_n)$ be an assignment of the variables, where $a_i \in D_i$. The* global cost function *F is defined by $F(\bar{a}) = \sum_{i=1}^{l} F_i(\bar{a})$, where $F_i(\bar{a})$ means F_i applied to the assignments in $\bar{a}$ restricted to the scope of F_i. Solving the COP requires to find $\bar{a}^* = (a_1^*, \ldots, a_n^*)$, satisfying all the constraints, such that $F(\bar{a}^*) = max_{\bar{a}} F(\bar{a})$ (or $F(\bar{a}^*) = min_{\bar{a}} F(\bar{a})$, in case of a minimisation problem).*

Cost functions are usually encoded as *tables* (Dechter, 2003), in which each row represents a variable assignment and its resulting value.

Definition 5 (complete (resp. incomplete) tables)**.** *A cost function F_i is* complete *if unfeasible assignments are explicitly represented with $-\infty$ ($+\infty$ in case of a minimisation problem) values. In contrast, if unfeasible assignments are not represented at all, F_i is said to be* incomplete*.*

COPs can be solved both with Dynamic Programming (DP) algorithms and with search-based approaches. On the one hand, BE (Dechter, 1999, 2003) is the most important DP algorithm that solves COPs, which has been recently implemented on GPUs (Bistaffa et al., 2016; Fioretto et al., 2015). BE has been successfully employed in practical applications (Larrosa et al., 2005), showing that it can be a viable solution approach if the problem is appropriately modelled as a COP. On the other hand, Marinescu and Dechter (2007) proposed a best-first search approach that adopts BE-based heuristics to guide the traversal of a particular AND/OR search tree. Both the above approaches have been considered as solvers for our model (see Section 3.4).

2.2 The GCCF Problem

The CSG problem (Shehory & Kraus, 1998; Sandholm et al., 1999) takes as input a finite set of n agents $A = \{a_1, \ldots, a_n\}$ and a characteristic function $v : 2^A \rightarrow \mathbb{R}$, that maps each coalition $S \in 2^A$ to its value, describing how much collective payoff a set of players can gain by forming a coalition. A coalition structure CS is a partition of the set of agents into disjoint coalitions. The set of all coalition structures is $\Pi(A)$. The value of a coalition structure CS is assessed as the sum of the values of its composing coalitions, i.e.,

$$V(CS) = \sum_{S \in CS} v(S).$$

The CSG problem aims at identifying CS^*, the most valuable coalition structure, i.e.,

$$CS^* = \arg\max_{CS \in \Pi(A)} V(CS).$$

Given a connected[1] graph $G = (A, E)$, where $E \subseteq A \times A$ is a set of edges between agents, representing their relationships (i.e., friendship), Myerson (1977) considers a coalition S to be feasible if all of its members are connected in the subgraph of G induced by S. That is, if for each pair of players from $a, b \in S$ there is a path in G that connects them without going out of S. Given G, the set of feasible coalitions is

$$\mathcal{FC}(G) = \{S \subseteq A \mid \text{The subgraph induced by } S \text{ on } G \text{ is connected}\}.$$

A GCCF problem (Voice et al., 2012b; Bistaffa et al., 2017a) is a CSG problem together with a graph G, where a coalition S is considered feasible if $S \in \mathcal{FC}(G)$. In GCCF a coalition structure CS is considered feasible if each of its coalitions is feasible, i.e., $\mathcal{CS}(G) = \{CS \in \Pi(A) \mid CS \subseteq \mathcal{FC}(G)\}$. Hence, the goal in GCCF is to identify CS^*, which is the most valuable feasible coalition structure, i.e.,

$$CS^* = \arg\max_{CS \in \mathcal{CS}(G)} V(CS).$$

Voice et al. (2012a) show that GCCF is NP-Hard.

2.3 Related Work

A number of approaches have been developed to solve the GCCF problem. The SoA algorithm that solves general[2] GCCF is based on DP (Yeh, 1986). Specifically, Voice et al.

1. A graph G with c components corresponds to c GCCF problems that can be solved independently.
2. A GCCF problem in which we do not make any particular assumption on the characteristic function.

(2012b) proposed the DyCE algorithm, a modified version of Yeh's approach that represents the SoA algorithm to solve general GCCF. For this reason, we consider DyCE as a sequential benchmark for our GPU approach. GCCF can also be solved as a CSG problem in which unfeasible coalitions have a value of $-\infty$. As such, SoA CSG algorithms based on DP can also be applied. Since we propose a GPU algorithm that solves GCCF, we compare it against IDP^G (Pawlowski et al., 2014), the only GPU version of a CSG algorithm. Existing DP approaches cannot scale over $\sim$25 agents (Michalak, Rahwan, Elkind, Wooldridge, & Jennings, 2016), as they require to store the value of each possible coalition in memory. In addition, DyCE does not take advantage of the presence of the graph to reduce its memory consumption, hence its memory requirements are $\Theta(2^n)$.

In order to deal with such drawbacks, a strand of literature (Voice et al., 2012a; Bistaffa et al., 2017a; Ohta et al., 2009; Ueda, Kitaki, Iwasaki, & Yokoo, 2011) has proposed to avoid storing each coalitional value in memory by exploiting some particular assumptions on the function $v(\cdot)$. Unfortunately, these approaches are not suitable for scenarios in which making assumptions on the characteristic function is not possible. On the one hand, the approach by Voice et al. (2012a) is not applicable if the Independence of Disconnected Members (IDM)[3] property does not hold. On the other hand, the approach by Bistaffa et al. (2017a) has the unpractical worst-case time complexity of $\Omega\left(\frac{n}{\ln(n)}^n\right)$, i.e., the number of coalition structures when G is a complete graph (Berend & Tassa, 2010). Finally, approaches based on *compact* characteristic functions representations (Ohta et al., 2009; Ueda et al., 2011) require strict assumptions on the characteristic function in order to be expressive and succinct. Such assumptions do not hold in many realistic application scenarios (e.g., collective energy purchasing or ridesharing) (Bistaffa et al., 2017a), as well as random functions, which we employ in our experiments. Hence, we do not compare against such approaches.

Constrained CF has also been investigated by Rahwan et al. (2011), but their solution algorithm only applies to *Basic* Constrained CF (BCCF), which cannot be used to represent GCCF (Bistaffa et al., 2017a).

In the field of operations research, the CSG problem has been tackled as the equivalent Set Partitioning (SP) problem (Yeh, 1986) with an Integer Linear Programming (ILP) formalisation (Rahwan, Michalak, Wooldridge, & Jennings, 2015). Such an approach has been shown to be inefficient if solved to optimality, e.g., even an industrial-strength solver such as ILOG's CPLEX was shown to be significantly slower than IDP (Rahwan et al., 2015). This discussion further motivates the choice of IDP^G as a benchmark for our approach.

Column Generation (CG) (Barnhart, Johnson, Nemhauser, Savelsbergh, & Vance, 1998) has been widely adopted in the operations research literature in order to solve very large integer programs, including the SP problem. However, in practical implementations, CG approaches typically do not generate guaranteed optimal solutions (Bredström, Jörnsten, Rönnqvist, & Bouchard, 2014), as we do in this paper. Furthermore, to the best of our knowledge, ILP with CG has been used to (sub-optimally) solve SP problems with up to 500000 variables (Zaghrouti, Soumis, & El Hallaoui, 2014), which, in the case of GCCF, corresponds approximately to a problem with 25 agents, i.e., smaller than the problems we solve in this paper.

3. The IDM property requires that, given two disconnected agents a_i and a_j, the presence of a_i does not affect the contribution of a_j.

Finally, we remark that, despite the constrained optimisation nature of CF, the only work that links CF with COPs is by Ueda et al. (2010). In this work, however, the authors tackle a problem of *different* nature *wrt* our work. In fact, we consider the standard CSG definition adopted in the AI literature, i.e., each feasible coalition is associated to a value by a characteristic function (see Section 2.2). Moreover, we use *one* COP to *solve* the CSG problem (see Section 3). In contrast, in the work by Ueda et al. (2010) the value of each coalition is not given by a characteristic function, but it is the "optimal solution of a Distributed Constrained Optimisation Problem (DCOP) among the agents of the coalition" (Ueda et al., 2010), i.e., one DCOP *for each coalition*. This is clearly different than optimally solve the CSG problem (like we do) and it requires to "solve an NP-hard problem just to obtain the value of a single coalition" (Ueda et al., 2010), i.e., an exponential number of NP-hard problems. To combat this complexity, Ueda et al. (2010) use an *approximate* algorithm that solves CSG with quality guarantees. Since we are interested in computing the optimal solution of the GCCF problem, we do not compare with this approach.

3. COP-GCCF

In this section we discuss COP-GCCF, our COP formalisation of the GCCF problem. Our COP comprises $|\mathcal{FC}(G)|$ *binary* variables, i.e., one per feasible coalition. In our model, $x_S = 1$ does not necessarily mean that S is part of the final coalition structure, since a variable can be activated because it is *required* by another variable (see Section 3.1). Intuitively, only active variables that correspond to *maximal* coalitions are part of the final coalition structure. In the example in Figure 2, if $x_{13} = 1$ in the solution of the COP, then $x_3 = 1$ since x_3 is required by x_{13}, but only $\{a_1, a_3\}$ is part of the final coalition structure.

Formally, $X = \{x_S \mid S \in \mathcal{FC}(G)\}$. The computation of X is equivalent to the enumeration of all the subgraphs of G and can be solved using one of the existing algorithms in the literature (Moerkotte & Neumann, 2006; Voice et al., 2012b). We use SlyCE by Voice et al. (2012b), which also provide a parallelised version, i.e., D-SlyCE.

A set of coalitions $\mathcal{S}$ is a partition of A if each agent is part of *exactly one* coalition, i.e.,

Property 1. *There are no overlapping coalitions in $\mathcal{S}$, i.e., $\nexists S, S' \in \mathcal{S} : S \cap S' \neq \emptyset$.*

Property 2. *Each agent in A is part of a coalition in $\mathcal{S}$, i.e., $\bigcup_{S \in \mathcal{S}} = A$.*

The above properties enforce constraints that determine the solution space of valid variable assignments for our COP. Within COP-GCCF, we exploit the structure of the graph G in order to express such constraints.

As a first step, we construct a *pseudotree* $PT(G)$ (Petcu, 2007) from G, establishing a partial order among the agents in A.

Definition 6 (pseudotree)**.** *A pseudotree $PT(G)$ of a graph G is a rooted tree with the same nodes as G and the property that adjacent nodes from the original graph fall in the same branch of $PT(G)$. Throughout this paper we assume that $PT(G)$ is constructed by means of a depth-first search (DFS) on G, i.e., $PT(G)$ is always a DFS-tree.*

$PT(G)$ is a graph in which edges are directed from children nodes to parent ones (Figure 1). Back-edges, i.e., edges present in G but not explicitly present in its tree representation, are marked with a dashed line. It is crucial to note that back-edges *cannot* be removed

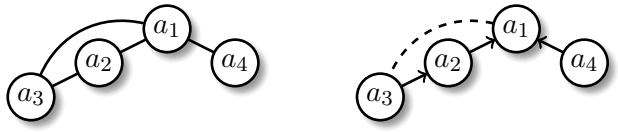

Figure 1: Example graph G and the corresponding $PT(G)$.

from the graph, as they must be considered in order to have a complete and correct GCCF algorithm. Specifically, the information expressed by back-edges is inherently maintained by the variables in the model. As an example, the information of (a_1, a_3) is maintained by x_{13} and x_{134}, which would correspond to unfeasible coalitions without such edge.[4]

Then, we partition the set of variables X in n sets X_i, each corresponding to $a_i \in A$, such that $X_i = \{x_S \mid S \text{ contains only } a_i \text{ or } a_i \text{ with a subset of its descendants in } PT(G)\}$. Each X_i represents the set of *local* variables to the agent a_i. In the above example, $X_1 = \{x_1, x_{12}, x_{13}, x_{123}, x_{14}, x_{124}, x_{134}, x_{1234}\}$, $X_2 = \{x_2, x_{23}\}$, $X_3 = \{x_3\}$, and $X_4 = \{x_4\}$.

As stated above, Properties 1 and 2 must be ensured in order to correctly represent the GCCF problem. Property 1 requires that the activation of a particular variable/coalition *excludes* the activation of incompatible variables, i.e., variables whose concurrent activation would generate overlapping coalitions. Now, since in COP-GCCF we construct n constraint functions F_i, each responsible for the variables in X_i (see Section 3.2), Property 2 can be easily achieved for such variables by allowing only the assignments in which exactly one local variable is activated. On the other hand, Property 1 cannot be directly enforced for variables that are local to different agents, i.e., that belong to different X_i, and hence, to different constraint functions. Notice that introducing additional binary constraints between overlapping variables would result in $\binom{|X|}{2}$ constraints. In contrast, we achieve this exploiting the concept of *required variables*.

3.1 Required Variables

The main idea behind *required variables* is that the formation of a variable/coalition $x_S \in X_i$ can be achieved exploiting the hierarchy induced by $PT(G)$. Intuitively, the agent a_i can negotiate the formation process only with its children nodes, allowing a more succinct representation of the problem and saving computational resources. As an example, x_{1234} (local to a_1) requires the participation of a_2, a_3, and a_4, but a_1 can force the participation of a_3 through a_2. In other words, x_{1234} *requires* x_4 and x_{23}, which indirectly *requires* x_3 through a_2. Formally, we represent such dependencies with the *requires* relation, denoted as $req(PT(G)) \subseteq X^2$, i.e., a set of couples of variables. Figure 2 illustrates such relation corresponding to the above example. Intuitively, if $(x_S, x_{S'}) \in req(PT(G))$, it means that x_S requires $x_{S'}$. In terms of variable assignments, the *requires* relation acts as an implication, i.e., $x_S = 1 \implies x_{S'} = 1$. Notice that, as a consequence, if x_S requires $x_{S'}$ and $x_{S'}$ requires $x_{S''}$, the activation of x_S results in the activation of $x_{S''}$, i.e.,

$$\forall x_S, x_{S'}, x_{S''} \in X : (x_S, x_{S'}) \in req(PT(G)) \text{ and } (x_{S'}, x_{S''}) \in req(PT(G)), x_S = 1 \implies x_{S''} = 1. \quad (1)$$

4. In our examples, each x_S will be named using indexes of agents in S, e.g., x_{a_1,a_3} will be named x_{13}.

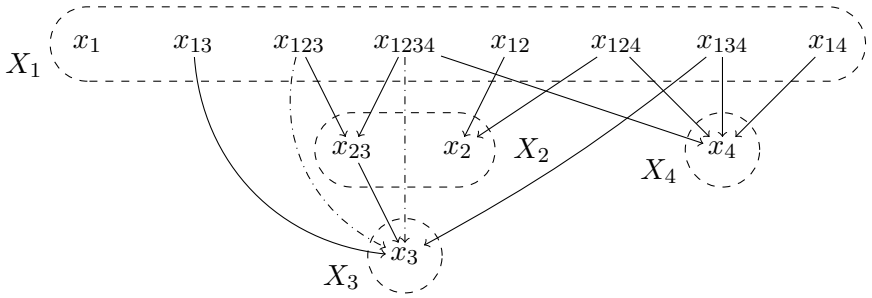

Figure 2: The *requires* relation (indirect requirements drawn as dash-dotted lines).

Due to the above property, *requires* exhibits a property similar to *transitivity*. On the other hand, we construct such a relation in a way such that if $(x_S, x_{S'}) \in req(PT(G))$ and $(x_{S'}, x_{S''}) \in req(PT(G))$, then $(x_S, x_{S''}) \notin req(PT(G))$, hence, strictly speaking, *requires* is *not* a transitive relation. This choice is motivated by the fact that the amount of required variables directly determines the scopes of the constraints within COP-GCCF (see Section 3.2), hence including the additional couple $(x_S, x_{S''})$ in $req(PT(G))$ would be redundant, since the relation between such variables is still expressed by the property in Equation 1. We characterise this property related to transitivity by means of the concept of *indirect requirement*.

Definition 7 (indirect requirement)**.** *A variable $x_S \in X$* indirectly requires *$x_{S''} \in X$ if $\exists x_{S'} \in X$ such that $(x_S, x_{S'}) \in req(PT(G))$ and $(x_{S'}, x_{S''}) \in req(PT(G))$.*

Any required variable $x_{S'}$ activated as a result of the *requires* relation does *not* correspond to the formation of S'. Thus, once the optimal solution of the COP model has been computed (see Section 3.4), such variables must be ignored when deriving the coalition structure.

For any agent a_i and any variable x_S local to a_i representing the coalition S, its required variables are computed with Algorithm 1, which iterates over the children of a_i (line 2) and computes the required variables relative to each child with the recursive routine in Algorithm 2. As an example, we compute the variables required by x_{1234} with such algorithms. We denote as PT_j the agents in the subtree of $PT(G)$ rooted in a_j. The first iteration of the loop in Algorithm 1 refers to the first child of a_1, i.e., a_2. Within the corresponding invocation of RecReq, $S^* = \{a_2, a_3\}$, resulting in the required variable x_{23}. Notice that the inner RecReq call returns $\emptyset$, since $\{a_1, a_4\} \cap PT_3 = \emptyset$. Similarly, the second iteration of the loop (i.e., the one for a_4) yields the required variable x_4. Note that x_{1234} indirectly requires x_3, since x_{23} requires such variable.

Algorithm 1 ComputeReq$(S, a_i, PT(G))$

1: $req_S \leftarrow \emptyset$ {Initialise empty set of required variables}
2: **for all** a_j children of a_i **do**
3: $\quad req_S \leftarrow req_S \cup$ RecReq$(S, a_j, PT(G))$
4: **return** req_S

Algorithm 2 RecReq$(S, a_j, PT(G))$

1: $req_S^j \leftarrow \emptyset$
2: $PT_j \leftarrow$ agents in the subtree of $PT(G)$ rooted in a_j
3: **if** $S \cap PT_j = \emptyset$ **then**
4: **return** $\emptyset$
5: **else**
6: $S^* = \arg\max_{\{x_{\bar{S}} \in X_j \mid \bar{S} \subseteq (S \cap PT_j)\}} \left|\bar{S} \cap S\right|$
7: $\{x_{S^*}$ is the variable in X_j, strictly formed by agents$\}$
8: $\{$in $S \cap PT_j$, with the maximum intersection with $S\}$
9: **for all** a_k children of a_j **do**
10: $req_S^j \leftarrow req_S^j \cup \{x_{S^*}\} \cup$ RecReq$(S \setminus S^*, a_k, PT(G))$
11: **return** req_S^j

The crucial feature of required variables is that any two variables that require the same variable *cannot* be enabled simultaneously. Since we cannot activate two variables both local to the same agent, two variables that require variables local to the same agent cannot be both active (Equation 2).

$$\forall x_S, x_{S'} \in X, a_k \in A : \exists x_{S''}, x_{S'''} \in X_k : (x_S, x_{S''}) \in req(PT(G)) \text{ and } (x_{S'}, x_{S'''}) \in req(PT(G)) \implies \neg(x_S \wedge x_{S'}). \quad (2)$$

By enforcing Equation 2, we ensure that no overlapping variables local to different agents are activated at the same time. Proposition 1 proves such property. As background for such proof, we first prove Lemma 1, that ensures that any variable, local to an agent a_i, and involving an agent a_k descendant of a_i, requires a variable local to a_k, possibly by indirect requirement. In the example in Figure 2, x_{13} (local to a_1) corresponds to a coalition containing a_3, which is a descendant of a_1. As a consequence, x_{13} requires a variable local to a_3, i.e., x_3. On the other hand, x_{123} indirectly requires x_3 via x_{23}.

Lemma 1. *Given a variable $x_S \in X_i$, i.e., local to a_i, such that $a_k \in S$, where a_k is a descendant of a_i in $PT(G)$, it exists a variable $x_{S'} \in X_k$, i.e., local to a_k, such that x_S requires $x_{S'}$, possibly by indirect requirement.*

Proof. Let a_j be the child of a_i such that $a_k \in PT_j$, i.e., a_k is a descendant of a_j, and consider the routine call RecReq$(S, a_i, PT(G))$ at line 3 of Algorithm 1 corresponding to a_j. We now show that RecReq$(S, a_j, PT(G))$ either falls into a base case (directly proving this lemma), or into a recursive one. In such case, we show that one of the inner recursive calls still verifies the hypotheses of this lemma, hence recursively applying this proof to such call.

- **Base case ($a_j = a_k$)**: line 6 of RecReq$(S, a_j, PT(G))$ results in a coalition S^* containing a_k as the highest agent in such coalition, considering the hierarchy induced by $PT(G)$. Hence, x_S requires $x_{S'} = x_{S^*}$, local to a_k.
- **Recursive case ($a_j \neq a_k$)**: we distinguish between two cases:

- $a_j \in S$: line 6 of RecReq$(S, a_j, PT(G))$ results in a coalition S^* containing both a_j and a_k, since $a_j \in S$ and $a_k \in PT_j$. Notice that $S \setminus S^*$ at line 10 does not contain a_k. Hence, x_S requires $x_{S'} = x_{S^*}$, which is *not* local to a_k, but to a_j. On the other hand, the recursive procedure that computes the variables required by $x_{S'} = x_{S^*}$ verifies the hypotheses of this lemma, since $a_k \in S^*$ and a_k is a descendant of a_j.
- $a_j \notin S$: line 6 of RecReq$(S, a_j, PT(G))$ results in $\emptyset$. In fact, since $a_j \notin S$, then $a_j \notin S \cap PT_j$, and line 6 only considers coalitions local to a_j (which all contain a_j) *strictly* composed of agents in $S \cap PT_j$, which does not contain a_j. Then, line 9 (which iterates over the children of a_j) contains one iteration referring to a child of a_j who still is an ancestor of a_k. Such iteration recursively calls RecReq with the same S, since $S^* = \emptyset$. Thus, the hypotheses of this lemma are verified.

Notice that both cases in the recursive step refer to agents a_j (all ancestors of a_k) that are gradually closer to a_k. Thus, the base case is eventually executed. □

Proposition 1. *Overlapping variables local to different agents cannot be activated together, i.e., $\forall x_S \in X_i, x_{S'} \in X_j$ such that $a_i \neq a_j$, if $S \cap S' \neq \emptyset$, then $\neg(x_S \wedge x_{S'})$.*

Proof. Let a_k be an agent in $S \cap S'$. Notice that a_k always exists since $S \cap S' \neq \emptyset$. As a direct consequence of how $PT(G)$ is constructed, $a_k \in PT_i$ and $a_k \in PT_j$, where PT_h represents the set of agents in the subtree of $PT(G)$ rooted in a_h. Now, since $PT(G)$ is a tree, it follows that either $a_i \in PT_j$ or $a_j \in PT_i$. Without loss of generality, assume that a_i is an ancestor of a_j, i.e., $a_j \in PT_i$, and thus, a_k is a descendant of both a_i and a_j. By applying Lemma 1 to x_S and $x_{S'}$, it follows that x_S and $x_{S'}$ require variables both local to a_k. Finally, Equation 2 ensures that $\neg(x_S \wedge x_{S'})$. □

As an example, our technique ensures that x_{13} and x_{23} cannot be both set to 1, since they both require x_3. In the next section, we show how to construct n constraint functions that implement the above discussed concepts.

3.2 Constructing Constraint Functions

As mentioned in Section 3, COP-GCCF involves n constraint functions F_i, one for each agent a_i. Each F_i is constructed according to the following definition.

Definition 8 (F_i)**.** *Each F_i is responsible for the variables local to a_i, hence we initialise the scope Q_i of each F_i to include X_i. To represent the* requires *relation, we include all the non-local variables that require a variable in X_i, i.e., $Q_i = X_i \cup \{x_S \mid \exists x_{S'} \in X_i : (x_S, x_{S'}) \in req(PT(G))\}$. The scope Q_i of each F_i comprises X_i, i.e., the variables local to a_i, plus all the non-local variables that require a variable in X_i. Each F_i contains $|Q_i|$ feasible assignments, i.e., one for each variable in the scope. The variable assignment in each row is constructed by activating the corresponding variable, namely x_S. If x_S is non-local, i.e., $x_S \notin X_i$, we also activate the variable required by x_S. Then, for each assignment in which a local variable $x_S \in X_i$ is activated, we define the corresponding value equal to $v(S)$, while such value is 0 when a non-local variable is considered. This avoids the duplication of $v(S)$ when x_S is propagated as a non-local variable across the constraint functions.*

Notice that, in our model, we do not explicitly represent unfeasible assignments, i.e., each F_i is an *incomplete* table. This is of utmost importance, since it allows us to reduce the space required by each F_i to a tractable size. Specifically, within COP-GCCF each F_i has $|Q_i|$ rows, as opposed to $2^{|Q_i|}$ rows, if we used complete tables with unfeasible assignments represented as $-\infty$. On the other hand, not all COP solution algorithms support incomplete tables. Henceforth, is it also necessary to use a solver that is able to exploit this feature, as discussed in detail in Section 3.4.

Figures 3–6 show an example of constraint functions (with non-local variables highlighted in grey) corresponding to the example in Figure 1. Notice that, at the moment, our model propagates several variables down the pseudotree. As an example, X_4 contains only one variable, but $Q_4 = X_4 \cup \{x_{1234}, x_{124}, x_{134}, x_{14}\}$. We will show how to reduce this amount in the next section.

	x_1	x_{12}	x_{13}	x_{14}	x_{123}	x_{124}	x_{134}	x_{1234}	Value
Local	1	0	0	0	0	0	0	0	$v(\{a_1\})$
	0	1	0	0	0	0	0	0	$v(\{a_1, a_2\})$
	0	0	1	0	0	0	0	0	$v(\{a_1, a_3\})$
	0	0	0	1	0	0	0	0	$v(\{a_1, a_4\})$
	0	0	0	0	1	0	0	0	$v(\{a_1, a_2, a_3\})$
	0	0	0	0	0	1	0	0	$v(\{a_1, a_2, a_4\})$
	0	0	0	0	0	0	1	0	$v(\{a_1, a_3, a_4\})$
	0	0	0	0	0	0	0	1	$v(\{a_1, a_2, a_3, a_4\})$

Figure 3: Constraint function F_1.

	x_2	x_{23}	x_{12}	x_{124}	x_{123}	x_{1234}	Value
Local	1	0	0	0	0	0	$v(\{a_2\})$
	0	1	0	0	0	0	$v(\{a_2, a_3\})$
Non-local	1	0	1	0	0	0	0
	1	0	0	1	0	0	0
	0	1	0	0	1	0	0
	0	1	0	0	0	1	0

Figure 4: Constraint function F_2.

	x_3	x_{134}	x_{13}	x_{23}	Value
Local	1	0	0	0	$v(\{a_3\})$
Non-local	1	1	0	0	0
	1	0	1	0	0
	1	0	0	1	0

Figure 5: Constraint function F_3.

	x_4	x_{1234}	x_{124}	x_{134}	x_{14}	Value
Local	1	0	0	0	0	$v(\{a_4\})$
Non-local	1	1	0	0	0	0
	1	0	1	0	0	0
	1	0	0	1	0	0
	1	0	0	0	1	0

Figure 6: Constraint function F_4.

We now formally prove that COP-GCCF is correct, i.e., that the optimal solution of COP-GCCF is the optimal solution of the corresponding GCCF problem (Proposition 2). First, we prove two necessary lemmas for such proposition.

Lemma 2. *Property 1 holds within COP-GCCF.*

Proof. Definition 8 guarantees that exactly one local variable is activated, and that overlapping variables local to different agents are not enabled at the same time, as a consequence of Proposition 1. As such, Property 1 holds. □

Lemma 3. *Property 2 holds within COP-GCCF.*

Proof. Each F_i only contains variable assignments in which exactly one local variable is enabled. As such, assignments in which a_i is not part of any coalition, i.e., the ones violating Property 2, are unfeasible and cannot be a solution. □

Proposition 2. *The optimal solution of COP-GCCF is the optimal solution of the corresponding GCCF problem.*

Proof. COP-GCCF ensures that the variable assignment produced as solution satisfies Properties 1 and 2. Furthermore, such an assignment maximises the sum of the values of the constraints, which, in COP-GCCF, is the sum of the values of the corresponding coalitions. As such, the solution of COP-GCCF represents the solution of the GCCF problem. □

In what follows, we improve our model by discussing how to reduce the scope of constraint functions, so to improve the memory requirements while maintaining the correctness.

3.3 Reducing the Size of Constraint Functions

Our method of constructing each F_i involves the addition to its scope of every non-local variable that requires a local one. Now, line 6 of Algorithm 2 implies that the set of required variables of a particular variable x_S local to the agent a_i contains variables that are local to a_i's descendants, i.e., agents lower than a_i in the hierarchy induced by $PT(G)$. As a consequence, the *requires* relation never involve variables that are local to the same agent. In contrast, if we could express the same dependencies by using variables that are also local to a_i, we could reduce the amount of variables added to the scope of F_i, and hence, its size.

In our improved model, we achieve this by introducing a slight modification in how the *requires* relation is expressed. As an example, notice that x_{1234}, local to a_1, requires x_{23} and x_4 in our original model, which belong to different branches in $PT(G)$ (Figure 7(a)).

When this happens, we can augment each required variable adding a_1, and obtaining x_{123} and x_{14} as new required variables (Figure 7(b)).

This method is applicable only when we have more than one required variable. In fact, if we applied it to the only required variable of x_{123}, i.e., x_{23}, we would re-obtain x_{123}. This modification allows us to greatly reduce the scope of the constraint functions in COP-GCCF. In fact, notice that the new required variables of x_{1234}, i.e., x_{123} and x_{14}, are both local to a_1, in contrast with the original ones that were local to a_2 and a_4. Thus, we can avoid adding x_{1234} to the scope of F_2 and F_4, since it no longer requires x_{23} and x_4. Furthermore, this improvement does not affect the correctness of our model, since the new *requires* relation is equivalent to the original one, as proven by the following proposition.

Proposition 3. *The* requires *relation obtained with our improved technique is equivalent to the original one.*

Proof. Let $x_S \in X_i$ be a variable local to a_i, and $x_{S'} \in X_j$ a variable local to a_j child of a_i, such that x_S requires $x_{S'}$ in our original model. Assume that, in our improved model, $(x_S, x_{S''}) \in req(PT(G))$, where $S'' = \{a_i\} \cup S'$ is equal to S' augmented with a_i, as a consequence of our improved technique of constructing required variables. Notice that, since S'' contains $\{a_i\}$, it is local to such agent. Furthermore, S'' cannot contain agents (apart from a_i) that are not part of PT_j, i.e., agents outside the subtree of $PT(G)$ rooted in a_j. Hence, a_j is the only child of a_i that does not result in $\emptyset$ at line 4 of Algorithm 2 when we construct the required variables for $x_{S''}$. Instead, the invocation of RecReq$(S'', a_j, PT(G))$ yields $S^* = S'$ as the result of the operation at line 6, since S' is precisely the coalition, strictly composed of agents in $S'' \cap PT_j$, with the maximum intersection with S''. Thus, $(x_{S''}, x_{S'}) \in req(PT(G))$. Now, since x_S requires $x_{S''}$, and $x_{S''}$ requires $x_{S'}$, then x_S indirectly requires $x_{S'}$ (see Definition 7) in our improved model. Therefore, the *requires* relation obtained with our improved technique is equivalent to the original one. □

As an example, Figure 8 shows that we indirectly achieve the original dependency between x_{1234} and x_{23} by means of a dependency with x_{123}, since the relation between x_{123} and x_{23} is unchanged. Figures 9–12 show the constraint functions obtained with our improved model considering the above example. It is clear that our technique allows to greatly reduce the number of non-local variables (cf. Figures 3–6), reducing the total number of columns from 23 to 17 (i.e., −26%).

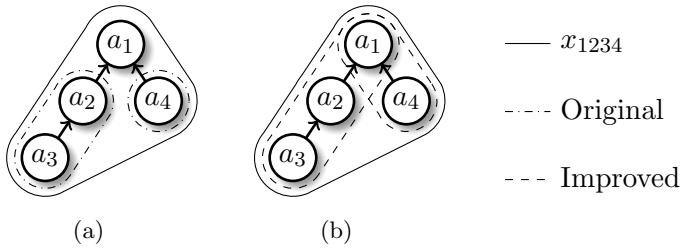

Figure 7: Required variables for x_{1234} (original vs. improved).

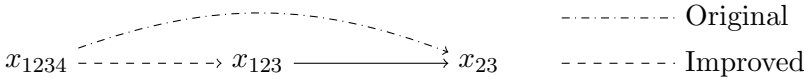

Figure 8: Relationships among x_{1234}, x_{123}, and x_{23}.

	x_1	x_{12}	x_{13}	x_{14}	x_{123}	x_{124}	x_{134}	x_{1234}	Value
Local	1	0	0	0	0	0	0	0	$v(\{a_1\})$
	0	1	0	0	0	0	0	0	$v(\{a_1, a_2\})$
	0	0	1	0	0	0	0	0	$v(\{a_1, a_3\})$
	0	0	0	1	0	0	0	0	$v(\{a_1, a_4\})$
	0	0	0	0	1	0	0	0	$v(\{a_1, a_2, a_3\})$
	0	1	0	1	0	1	0	0	$v(\{a_1, a_2, a_4\})$
	0	0	1	1	0	0	1	0	$v(\{a_1, a_3, a_4\})$
	0	0	0	1	1	0	0	1	$v(\{a_1, a_2, a_3, a_4\})$

Figure 9: Improved constraint function F_1.

	x_2	x_{23}	x_{12}	x_{123}	Value
Local	1	0	0	0	$v(\{a_2\})$
	0	1	0	0	$v(\{a_2, a_3\})$
Non-local	1	0	1	0	0
	0	1	0	1	0

Figure 10: Improved constraint function F_2.

	x_3	x_{13}	x_{23}	Value
Local	1	0	0	$v(\{a_3\})$
Non-local	1	1	0	0
	1	0	1	0

Figure 11: Improved constraint function F_3.

	x_4	x_{14}	Value
Local	1	0	$v(\{a_4\})$
Non-local	1	1	0

Figure 12: Improved constraint function F_4.

In what follows, we discuss how we compute the optimal variable assignment of the above defined COP.

3.4 Solving the COP

As previously discussed, COPs can be solved both with BE and with search-based approaches. However, a fundamental feature of COP-GCCF is the high constraint tightness (i.e., a lot of variable assignments in each F_i table are unfeasible, see Section 3.2). Hence,

Remark 1. *Using a solver that is able to* internally represent *constraints as* incomplete *tables, which allow to significantly reduce memory requirements, is a crucial aspect in the choice of the COP solution algorithm. This feature should not be confused with the ability of the solver of* reading *problem instances in a format that allows to specify constraints as incomplete tables (e.g., the WCSP format). In fact, several solvers (see below) can read problem instances with incomplete tables, which are then converted and internally represented as* complete *ones, thus still incurring high memory requirements.*[5]

The SoA search-based solution approach (Marinescu & Dechter, 2007) adopts BE-based heuristics to guide the traversal of a particular AND/OR search tree. Such an algorithm can read problem instances with incomplete tables, which are then internally converted to complete ones, i.e., with each unfeasible assignment associated to $-\infty$ (see Remark 1). This results in a huge consumption of memory (see Section 3.2), since each F_i has to contain $2^{|Q_i|}$ rows, $2^{|Q_i|} - |Q_i|$ of which are associated to $-\infty$. We tested Marinescu and Dechter's approach on problem instances expressed using incomplete tables (i.e., in WCSP format). Our results show that this approach could not solve COPs representing GCCF problems with > 5 agents, due to high memory requirements. We obtained similar results with ToulBar2 (Allouche, de Givry, & Schiex, 2010).[6] For this reason, we do not report these results in our experimental evaluation.

To the best of our knowledge, the only COP solution approach that internally represents constraints as incomplete tables is CUBE, i.e., the GPU version of BE by Bistaffa et al. (2016). Fioretto et al. (2015) also proposed a GPU version of BE, which, on the other hand, does not internally employ incomplete tables, and hence, would incur in the same drawbacks discussed above. As a consequence, we adopt Bistaffa et al.'s approach to solve COP-GCCF.

4. Experimental Evaluation

The main goals of our empirical analysis are i) to evaluate the performance of COP-GCCF in terms of runtime and memory requirements and ii) to compare it with DyCE, i.e., the SoA algorithm to solve general GCCF, and with IDP^G, i.e., the only GPU implementation of an algorithm that solves CSG.

5. Adapting existing approaches to internally support incomplete tables would require deep changes to the implementation. Making such changes is out of the scope of the paper.
6. Topology-aware solvers are not effective for our model, since the constraint graph has usually a very high treewidth (i.e., 20–30).

4.1 Experimental Methodology

We generate random GCCF instances considering three different network topologies, i.e., scale-free networks obtained with the Albert and Barabási (2002) model with the m parameter equal to 1 and 2, and subgraphs of a large crawl of the Twitter social graph (Kwak et al., 2010). G is obtained by means of a breadth-first traversal starting from a random node of the whole graph, adding each node and the corresponding arcs to G, until the desired number of nodes is reached (Russell, 2013). The number of edges is $m \cdot n - \frac{m \cdot (m+1)}{2}$ and $\sim 1.6 \cdot n$ for scale-free networks and Twitter subgraphs respectively. Each feasible coalition has an uniformly distributed random value within $[-10, 10]$, while, for IDP^G, unfeasible coalitions have a value of $-\infty$. We vary n within $[20, 30]$,[7] generating 20 random instances for each of the above network topologies and solving each of them with the three considered algorithms. For each n, we report the average and the standard error of the mean of such 20 repetitions. All our experiments are run on a machine with a 3.40GHz CPU, 16GB of memory and a GeForce GTX 680 GPU. The building phase of COP-GCCF is sequential and has been implemented in C++, while CUBE has been implemented in CUDA. See (Bistaffa et al., 2016) for a detailed analysis of the parallel speed-up of CUBE. For DyCE and IDP^G, we used the implementations provided by the respective authors.

4.2 Runtime

In order to fully understand our results, it is important to notice that the complexity of any GCCF problem is significantly influenced by the density of the graph G, as a larger number of connections results in a larger number of feasible coalitions, and, therefore, a larger solution space. For scale-free networks, density is directly determined by the parameter m, which represents the number of edges incident to each newly added node using the Barabási-Albert generation model. For Twitter subgraphs, we verified that such topology results in a density equivalent to a scale-free network with $1 \leq m \leq 2$. Figures 13–15 show the runtime needed to solve[8] the GCCF problems considering the above discussed network topologies. Our approach outperforms DyCE and IDP^G on scale-free networks with $m = 1$ and Twitter subgraphs, computing solutions up to 4 orders of magnitude faster than counterparts in the former scenario, and 1 order of magnitude faster than IDP^G in the latter one. For scale-free networks with $m = 2$, our approach is one order of magnitude slower than IDP^G, but it computes solutions 2 times faster than DyCE for 30 agents. In general, our results show that COP-GCCF outperforms DyCE in all the considered network topologies, and it is one order of magnitude faster than IDP^G when considering a realistic dataset, i.e., Twitter.

4.3 Memory

Figures 16–18 show the memory requirements of the considered approaches. For COP-GCCF, we measure the size of the largest table generated during the entire execution of the algorithm, while the memory requirements of DyCE and IDP^G are both $\Theta(2^n)$, regardless of the network topology. Specifically, DyCE and IDP^G require $4 \cdot 2^n$ bytes, since coalitional values are stored as `float` values.

7. DyCE and IDP^G cannot solve instances larger than 30 agents due to their memory requirements.
8. Runtime values include the building phase of COP-GCCF, which is negligible *wrt* the solution time.

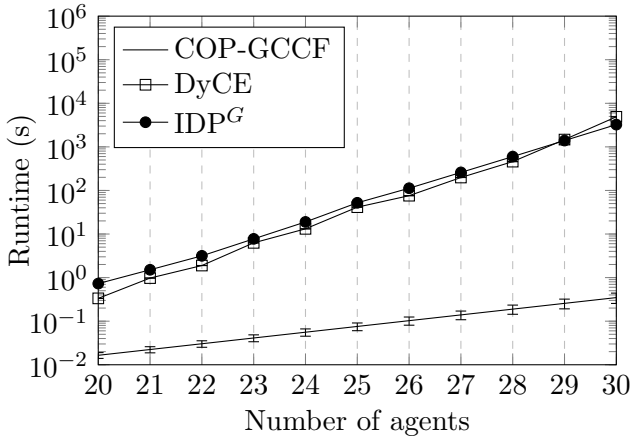

Figure 13: Runtime for scale-free networks with $m = 1$.

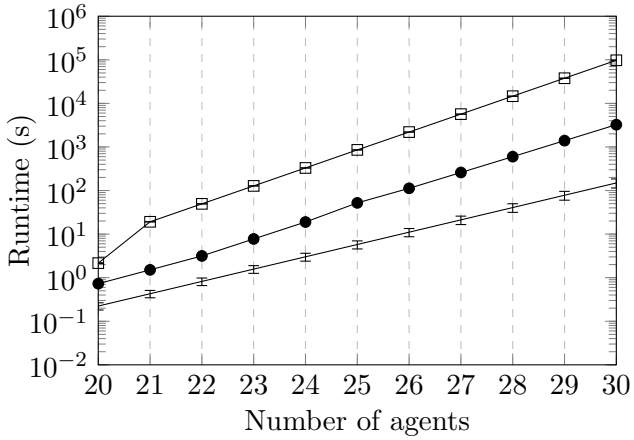

Figure 14: Runtime for Twitter subgraphs.

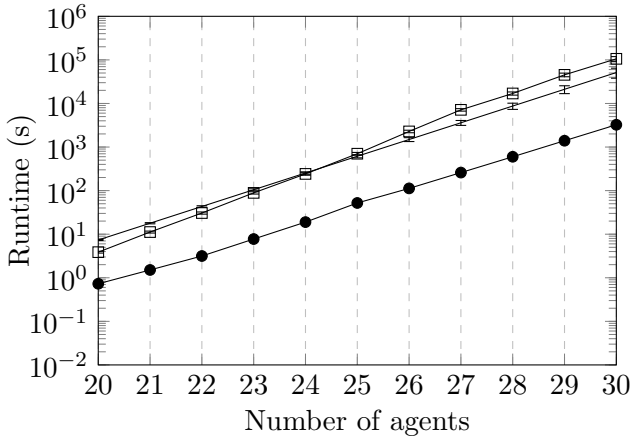

Figure 15: Runtime for scale-free networks with $m = 2$.

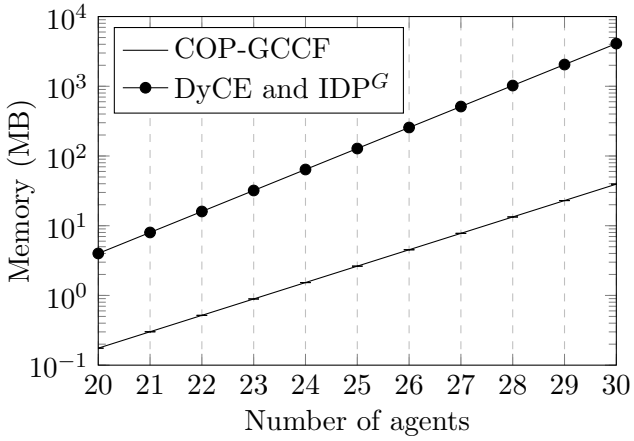

Figure 16: Memory for scale-free networks with $m = 1$.

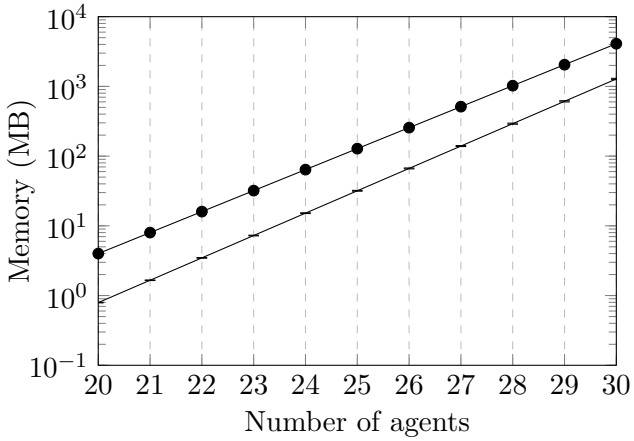

Figure 17: Memory for Twitter subgraphs.

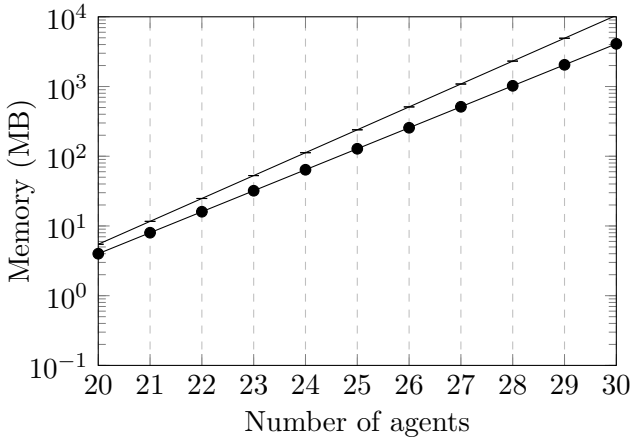

Figure 18: Memory for scale-free networks with $m = 2$.

Our results follow the behaviour discussed in the previous section, i.e., the memory requirements of COP-GCCF are lower with respect to DyCE and IDP^G for scale-free networks with $m = 1$ and Twitter subgraphs. Specifically, the memory consumption of our approach is 2 orders of magnitude lower in the former case, and 1 order of magnitude lower in the latter one. For scale-free networks with $m = 2$, COP-GCCF requires twice as much memory with respect to DyCE and IDP^G, due to the higher density of G that results in a larger number of variables.

5. Conclusions

We propose COP-GCCF, a novel approach that models the GCCF problem as a COP, and we solve such COP with a GPU algorithm based on BE. Our results show that our approach outperforms SoA algorithms on a realistic dataset, both in terms of runtime and memory. Moreover, we establish a clear link between GCCF and COPs, which, to the best of our knowledge, has never been proposed before in the literature. Future work will aim at applying our model to realistic scenarios, such as Collective Energy Purchasing (Bistaffa et al., 2017a) and Social Ridesharing (Bistaffa et al., 2017b).

Acknowledgments

Bistaffa was supported by the H2020-MSCA-IF-2016 HPA4CF project.

References

Albert, R. & Barabási, A. L. (2002). Statistical mechanics of complex networks. *Reviews of Modern Physics*, *74*(1), 47–97.

Allouche, D., de Givry, S., & Schiex, T. (2010). *Toulbar2, an open source exact cost function network solver*. INRIA.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., & Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, *46*(3), 316–329.

Berend, D. & Tassa, T. (2010). Improved bounds on bell numbers and on moments of sums of random variables. *Probability and Mathematical Statistics*, *30*(2), 185–205.

Bistaffa, F., Bombieri, N., & Farinelli, A. (2016). An efficient approach for accelerating bucket elimination on GPUs. *IEEE Transactions on Cybernetics*, *47*(11), 3967–3979.

Bistaffa, F., Farinelli, A., Cerquides, J., Rodríguez-Aguilar, J., & Ramchurn, S. D. (2017a). Algorithms for graph-constrained coalition formation in the real world. *ACM Transactions on Intelligent Systems and Technology*, *8*(4).

Bistaffa, F., Farinelli, A., Chalkiadakis, G., & Ramchurn, S. D. (2017b). A cooperative game-theoretic approach to the social ridesharing problem. *Artificial Intelligence*, *246*, 86–117.

Bredström, D., Jörnsten, K., Rönnqvist, M., & Bouchard, M. (2014). Searching for optimal integer solutions to set partitioning problems using column generation. *Operational Research*, *21*(2), 177–197.

Dechter, R. (1999). Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, *113*(1–2), 41–85.

Dechter, R. (2003). *Constraint processing*. Morgan Kaufmann.

Demange, G. (2004). On group stability in hierarchies and networks. *Political Economy*, *112*(4), 754–778.

Fioretto, F., Le, T., Pontelli, E., Yeoh, W., & Son, T. (2015). Exploiting GPUs in solving (distributed) constraint optimization problems with dynamic programming. In *Principles and Practice of Constraint Programming*, pp. 121–139.

Greengard. (2016). GPUs reshape computing. *Communications of the ACM*, *59*(9).

Kwak, H., Lee, C., Park, H., & Moon, S. (2010). What is Twitter, a social network or a news media? In *International World Wide Web Conference*, pp. 591–600.

Larrosa, J., Morancho, E., & Niso, D. (2005). On the practical use of variable elimination in constraint optimization problems: "still-life" as a case study. *Journal of Artificial Intelligence Research*, *23*, 421–440.

Malapert, A., Régin, J.-C., & Rezgui, M. (2016). Embarrassingly parallel search in constraint programming. *Journal of Artificial Intelligence Research*, *57*, 421–464.

Marinescu, R. & Dechter, R. (2007). Best-first AND/OR search for graphical models. In *AAAI Conference on Artificial Intelligence*, pp. 1171–1176.

Michalak, T., Rahwan, T., Elkind, E., Wooldridge, M., & Jennings, N. R. (2016). A hybrid exact algorithm for complete set partitioning. *Artificial Intelligence*, *230*, 14–50.

Moerkotte, G. & Neumann, T. (2006). Analysis of two existing and one new dynamic programming algorithm for the generation of optimal bushy join trees without cross products. In *International Conference on Very Large Data Bases*, pp. 930–941.

Myerson, R. B. (1977). Graphs and cooperation in games. *Mathematics of Operations Research*, *2*(3), 225–229.

Ohta, N., Conitzer, V., Ichimura, R., Sakurai, Y., Iwasaki, A., & Yokoo, M. (2009). Coalition structure generation utilizing compact characteristic function representations. In *Principles and Practice of Constraint Programming*, pp. 623–638.

Pawlowski, K., Kurach, K., Svensson, K., Ramchurn, S. D., Michalak, T. P., & Rahwan, T. (2014). Coalition structure generation with the graphics processing unit. In *International Conference on Autonomous Agents and Multi-Agent Systems*, pp. 293–300.

Petcu, A. (2007). *A class of algorithms for distributed constraint optimization* (PhD. Thesis No. 3942, Swiss Federal Institute of Technology (EPFL)).

Rahwan, T., Michalak, T. P., Elkind, E., Faliszewski, P., Sroka, J., Wooldridge, M., & Jennings, N. R. (2011). Constrained coalition formation. In *AAAI Conference on Artificial Intelligence*, pp. 719–725.

Rahwan, T., Michalak, T. P., Wooldridge, M., & Jennings, N. R. (2015). Coalition structure generation: A survey. *Artificial Intelligence*, *229*, 139–174.

Russell, M. A. (2013). *Mining the social web*. O'Reilly Media.

Sandholm, T., Larson, K., Andersson, M., Shehory, O., & Tohmé, F. (1999). Coalition structure generation with worst case guarantees. *Artificial Intelligence*, *111*(1), 209–238.

Shehory, O. & Kraus, S. (1998). Methods for task allocation via agent coalition formation. *Artificial Intelligence*, *101*(1-2), 165–200.

Ueda, S., Iwasaki, A., Yokoo, M., Silaghi, M.-C., Hirayama, K., & Matsui, T. (2010). Coalition structure generation based on distributed constraint optimization. In *AAAI Conference on Artificial Intelligence*, pp. 197–203.

Ueda, S., Kitaki, M., Iwasaki, A., & Yokoo, M. (2011). Concise characteristic function representations in coalitional games based on agent types. In *International Conference on Autonomous Agents and Multi-Agent Systems*, pp. 1271–1272.

Voice, T., Polukarov, M., & Jennings, N. (2012a). Coalition structure generation over graphs. *Journal of Artificial Intelligence Research*, *45*, 165–196.

Voice, T., Ramchurn, S. D., & Jennings, N. (2012b). On coalition formation with sparse synergies. In *International Conference on Autonomous Agents and Multi-Agent Systems*, pp. 223–230.

Yeh, D. Y. (1986). A dynamic programming approach to the complete set partitioning problem. *BIT Numerical Mathematics*, *26*(4), 467–474.

Zaghrouti, A., Soumis, F., & El Hallaoui, I. (2014). Integral simplex using decomposition for the set partitioning problem. *Operations Research*, *62*(2), 435–449.