Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

A COP Model for Graph-Constrained Coalition Formation (Extended Abstract)

Filippo Bistaffa[1] and **Alessandro Farinelli**[2]
[1] IIIA-CSIC, Campus UAB, 08913, Cerdanyola, Catalonia, Spain
[2] Computer Science Department, University of Verona, Strada le Grazie 15, 37134, Verona, Italy
filippo.bistaffa@iiia.csic.es, alessandro.farinelli@univr.it

Abstract

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

1 Introduction

Coalition Formation (CF) [Sandholm *et al.*, 1999] is a fundamental collaboration approach in multi-agent systems, where multiple agents join forces (or form groups) to achieve either their individual or collective goals. In this paper we focus on the associated optimisation problem, i.e., Coalition Structure Generation (CSG), which aims at 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*.

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 *et al.* [2012a], Voice *et al.* [2012b], and Bistaffa *et al.* [2017a]. GCCF is characterised by a specific type of constraints that encode 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 GCCF, edges enable connected agents to form a coalition and a coalition is *feasible* only if its members are the vertices of a connected subgraph. Several real-world scenarios feature this type of constraints, 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 *et al.*, 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 that solve GCCF either require an exponential amount of memory [Voice *et al.*, 2012b] or make specific assumptions on the characteristic function [Voice *et al.*, 2012a; Bistaffa *et al.*, 2017a].

Now, GCCF can be seen as 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, we propose COP-GCCF,[1] the first approach that models GCCF as a Constrained Optimisation Problem (COP). COP-GCCF does not make any assumption on the structure of the characteristic function. As a consequence, in such general case, GCCF is NP-Hard, hence finding the optimal solution is computationally demanding. Nonetheless, the structure of the characteristic function could be not known and/or hard to formalise, hence exploiting it could be often very difficult or impossible. Moreover, even if the structure of the characteristic function is known, it could be difficult to exploit from the algorithmic point of view. For instance, in the work by Bistaffa *et al.* [2017b] the structure is known a priori, but the authors need to devise a complex, ad-hoc technique (i.e., not easily applicable to scenarios different from ridesharing) to compute an upper-bound on the characteristic function, so as to exploit such a structure within their branch-and-bound CSG algorithm. Therefore, it is important to have a solution technique that does not require any property on the characteristic function.

To achieve this objective, we exploit the *structure of the graph* within COP-GCCF, 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). We represent constraints 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

[1] This paper is an extended abstract of an article in the Journal of Artificial Intelligence Research [Bistaffa and Farinelli, 2018], which contains parts of this paper. The journal version contains a full theoretical discussion of our approach, a comprehensive literature review, and an extended experimental evaluation.

exponential to linear *wrt* the number of variables in the scope. We then solve the COP-GCCF model using a GPU implementation of Bucket Elimination (BE) provided by Bistaffa *et al.* [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 3.3). In fact, SoA COP solution algorithms [Marinescu and Dechter, 2007; Fioretto *et al.*, 2015] without this capability could only solve problems with up to 5 agents in our tests. Our results show that BE can be a practical solution approach if the problem is appropriately modelled as a COP, as also confirmed by Larrosa *et al.* [2005]. Finally, we exploit highly-parallel architectures (i.e., GPUs). This choice is motivated by the successful use of cloud-based [Malapert *et al.*, 2016] and GPU-based [Greengard, 2016] parallel approaches to speed-up the solution of problems that exhibit a high level of parallelism, especially 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. 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 test COP-GCCF both on realistic and synthetic network topologies. Results show that COP-GCCF does not provide any benefit with dense graphs, but it outperforms SoA algorithms (i.e., DyCE and IDP^G) on sparse topologies, both in terms of runtime and memory. COP-GCCF is at least one *order of magnitude* (OoM) faster than counterpart approaches using Twitter [Kwak *et al.*, 2010] as a realistic graph topology. Our tests confirm that COP-GCCF improves upon SoA GCCF solution algorithms, by correctly exploiting the structure of the graph, even without exploiting the structure of the characteristic function.

2 Background

2.1 COPs

Constraint Optimisation Problems (COPs) [Dechter, 2003] are a general class of problems, which can be used to model several optimisation scenarios [Dechter, 1999]. A COP takes as input a cost network, i.e., a graph among the variables and the cost functions of the problem, and requires to compute the assignment of the variables that minimise such functions. Cost functions can be encoded as *tables*, in which each row represents a variable assignment and its resulting value.

Definition 1 (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, Bucket Elimination (BE) [Dechter, 1999; Dechter, 2003] is the most important DP algorithm that solves COPs, which has been recently implemented on GPUs [Bistaffa *et al.*, 2016]. 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 3.3).

2.2 The GCCF Problem

The Coalition Structure Generation (CSG) problem [Shehory and 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 $v : 2^A \to \mathbb{R}$, 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 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 Graph-Constrained Coalition Formation (GCCF) problem [Voice *et al.*, 2012b] 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 GCCF is to identify CS^*, which is the most valuable feasible coalition structure, i.e., $CS^* = \arg\max_{CS \in \mathcal{CS}(G)} V(CS)$. GCCF is NP-Hard [Voice *et al.*, 2012a].

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.[2]

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 and 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.,

[2] 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}.

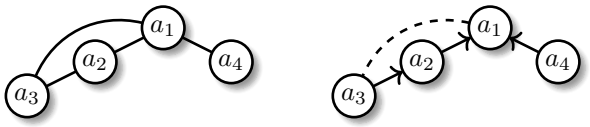

Figure 1: Example graph G and the corresponding $PT(G)$.

Property 1. *There are no overlapping coalitions in $\mathcal{S}$.*

Property 2. *Each agent in A is part of a coalition in $\mathcal{S}$.*

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. $PT(G)$ 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 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.

Then, we partition the set of variables X in n sets X_i, each corresponding to $a_i \in A$, such that $X_i = \{x_S \mid S$ contains only a_i or 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.

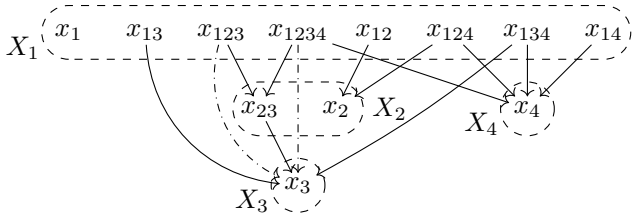

Figure 2: The *requires* relation (indirect requirements drawn as dash-dotted lines).

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. By enforcing this, we ensure that no overlapping variables local to different agents are activated at the same time.

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 2 (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 3.3.

	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 3: Constraint function F_4.

Figure 3 shows an example of constraint function (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 $Q_4 = X_4 \cup \{x_{1234}, x_{124}, x_{134}, x_{14}\}$. In the journal version of this paper [Bistaffa and Farinelli, 2018] we also discuss an optimised version of our model that allows one to significantly reduce the number of propagated variables.

COP-GCCF is correct (as we prove in the journal version [Bistaffa and Farinelli, 2018]), i.e., the optimal solution of COP-GCCF is the optimal solution of the corresponding GCCF problem.

3.3 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., several variable assignments in each F_i table are unfeasible, see Section 3.2). Hence, 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.

The SoA search-based solution approach [Marinescu and 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 $-\infty$. 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 *et al.*, 2010].[3] 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.

[3]Topology-aware solvers are not effective for our model, since the constraint graph has usually a very high treewidth (i.e., 20–30).

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 and with IDP^G.

Following Voice *et al.* [2012b] and Bistaffa *et al.* [2017b], we generate random GCCF instances considering three different network topologies for the graph G, i.e., scale-free networks obtained with the Barabási-Albert model with $m = 1$ and 2, and subgraphs of a large crawl of the Twitter social graph [Kwak *et al.*, 2010]. G is obtained by means of a BFS 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]. Each feasible coalition is associated to an uniformly distributed random value within $[-10, 10]$, while, for IDP^G, unfeasible coalitions have a value of $-\infty$. We vary n within $[20, 30]$, generating 20 random instances for each of the above network topologies and solving each of them with the three considered algorithms. All our experiments are run on a machine with a 3.40GHz CPU, 16GB of memory and a GeForce GTX 680 GPU.[4] For DyCE and IDP^G, we used the authors' implementations.

In terms of runtime, our results show that COP-GCCF outperforms DyCE in all the considered network topologies, and it is one OoM faster than IDP^G on a realistic dataset, i.e., Twitter. On the other hand, the trend in our tests also shows that IDP^G performs better than COP-GCCF on dense or almost completely connected graphs. This is expected, as IDP^G is specifically devised for completely connected graphs, and, hence, is a preferable solution technique in such scenarios.

In terms of memory consumption, our results follow the behaviour discussed above, i.e., the memory requirements of COP-GCCF are lower *wrt* DyCE and IDP^G for scale-free networks with $m = 1$ and Twitter subgraphs. The memory consumption of our approach is 2 OoM lower in the former case, and 1 OoM lower in the latter one. For scale-free networks with $m = 2$, COP-GCCF requires twice as much the memory *wrt* DyCE and IDP^G, due to the higher density of G that results in a larger number of variables.

5 Conclusions

We proposed 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.

[4]Our implementation is available at https://github.com/filippobistaffa/COP-GCCF.

References

[Allouche *et al.*, 2010] David Allouche, Simon de Givry, and Thomas Schiex. Toulbar2, an open source exact cost function network solver. Technical report, INRIA, 2010.

[Bistaffa and Farinelli, 2018] Filippo Bistaffa and Alessandro Farinelli. A COP Model for Graph-Constrained Coalition Formation. *Journal of Artificial Intelligence Research*, 62:133–153, 2018.

[Bistaffa *et al.*, 2016] Filippo Bistaffa, Nicola Bombieri, and Alessandro Farinelli. An efficient approach for accelerating bucket elimination on GPUs. *IEEE Transactions on Cybernetics*, 47(11):3967–3979, 2016.

[Bistaffa *et al.*, 2017a] Filippo Bistaffa, Alessandro Farinelli, Jesús Cerquides, Juan Rodríguez-Aguilar, and Sarvapali D. Ramchurn. Algorithms for graph-constrained coalition formation in the real world. *ACM Transactions on Intelligent Systems and Technology*, 8(4), 2017.

[Bistaffa *et al.*, 2017b] Filippo Bistaffa, Alessandro Farinelli, Georgios Chalkiadakis, and Sarvapali D. Ramchurn. A cooperative game-theoretic approach to the social ridesharing problem. *Artificial Intelligence*, 246:86–117, 2017.

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

[Dechter, 2003] Rina Dechter. *Constraint processing*. Morgan Kaufmann, 2003.

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

[Fioretto *et al.*, 2015] Ferdinando Fioretto, Tiep Le, Enrico Pontelli, William Yeoh, and TranCao Son. Exploiting GPUs in solving (distributed) constraint optimization problems with dynamic programming. In *Principles and Practice of Constraint Programming*, pages 121–139. 2015.

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

[Kwak *et al.*, 2010] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is Twitter, a social network or a news media? In *International World Wide Web Conference*, pages 591–600, 2010.

[Larrosa *et al.*, 2005] Javier Larrosa, Enric Morancho, and David Niso. 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, 2005.

[Malapert *et al.*, 2016] Arnaud Malapert, Jean-Charles Régin, and Mohamed Rezgui. Embarrassingly parallel search in constraint programming. *Journal of Artificial Intelligence Research*, 57:421–464, 2016.

[Marinescu and Dechter, 2007] Radu Marinescu and Rina Dechter. Best-first AND/OR search for graphical models. In *AAAI Conference on Artificial Intelligence*, pages 1171–1176, 2007.

[Moerkotte and Neumann, 2006] Guido Moerkotte and Thomas Neumann. 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*, pages 930–941, 2006.

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

[Petcu, 2007] Adrian Petcu. *A Class of Algorithms for Distributed Constraint Optimization*. Phd. thesis no. 3942, Swiss Federal Institute of Technology (EPFL), 2007.

[Russell, 2013] Matthew A. Russell. *Mining the Social Web*. O'Reilly Media, 2013.

[Sandholm *et al.*, 1999] Tuomas Sandholm, Kate Larson, Martin Andersson, Onn Shehory, and Fernando Tohmé. Coalition structure generation with worst case guarantees. *Artificial Intelligence*, 111(1):209–238, 1999.

[Shehory and Kraus, 1998] Onn Shehory and Sarit Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165–200, 1998.

[Voice *et al.*, 2012a] Thomas Voice, Maria Polukarov, and Nicholas Jennings. Coalition structure generation over graphs. *Journal of Artificial Intelligence Research*, 45:165–196, 2012.

[Voice *et al.*, 2012b] Thomas Voice, Sarvapali D. Ramchurn, and Nicholas R. Jennings. On coalition formation with sparse synergies. In *International Conference on Autonomous Agents and Multi-Agent Systems*, pages 223–230, 2012.