A tutorial on optimisation for multi-agent systems

Jesus Cerquides[1], Alessandro Farinelli[2], Pedro Meseguer[1] and Sarvapali D. Ramchurn[3]

[1] *IIIA, Artificial Intelligence Research Institute*
CSIC, Spanish Council for Scientific Research
Email: {cerquide,pedro}@iiia.csic.es
[2] *University of Verona, Italy*
Email: alessandro.farinelli@univr.it
[3] *School of Electronics and Computer Science, University of Southampton, SO17 1BJ, UK*
Email: {sdr1,mv2y11}@soton.ac.uk

Research on optimisation in multi-agent systems has contributed with a wealth of techniques to solve many of the challenges arising in a wide range of multi-agent application domains. Multi-agent optimisation focuses on casting multi-agent system problems into optimisation problems whose solving could possibly involve the active participation of the agents in a multi-agent system. Research on multi-agent optimisation has rapidly become a very technical, specialised field. Moreover, the contributions to the field in the literature are largely scattered. These two factors dramatically hinder the access to a basic, general view of the foundations of the field. This tutorial is intended to ease such access by providing a gentle introduction to fundamental concepts and techniques on multi-agent optimisation.

Keywords: Optimisation; Multi-agent systems; coalition formation; resource allocation; auctions; distributed constraint optimisation; DCOP

Received 15 November 2012; revised 15 November 2012

1. INTRODUCTION

Multi-agent system (MAS) applications in a number of areas such as e-commerce, disaster management, and information acquisition through embedded devices (e.g. wireless sensor networks) have generated a number of new challenges for algorithm designers. These challenges mainly take the form of very hard optimisation problems that are substantially different from problems traditionally dealt with in other areas (e.g. industrial processes or scheduling applications). More specifically, novel challenges come from the distributed nature of MAS where the actors reside on different computational units and can communicate only a limited amount of information with their neighbours (e.g., fire fighters on different trucks with limited communication capabilities, or sensor networks spread over a large area). Moreover, the agents may be acting on behalf of different stakeholders (e.g., fire fighters may need to coordinate with ambulances, buyers in online markets interacting with sellers), each with its own aims and objectives (e.g., fire fighters extinguish fires while ambulances aim to provide medical care), have different computation/communication capabilities, and be tied to physical devices prone to failures. Furthermore, given the dynamic nature of the application scenarios (e.g., where fires may spread across a region, or where goods in an online market may arrive at different rates), effective algorithms have to provide anytime solutions and approximate techniques are often required/desirable.

Research on optimisation in MAS has contributed with a wealth of techniques to support many of the challenges outlined above. However, we identify two factors that dramatically hinder the access to a basic, general view of the foundations of this field. First, research on multi-agent optimisation has rapidly become a very technical and specialised field. Second, the contributions to the field in the literature are largely scattered across various papers and literature surveys without any coherent view of how all the optimisation techniques developed by the MAS community compare against or overlap with each other. With the aim to ease such access, this tutorial provides a gentle introduction to fundamental concepts and techniques on multi-agent optimisation. In particular, this tutorial focuses on several fundamental MAS problems that have been cast as multi-agent optimisation problems, namely:

- *How to assemble collectives.* In many applications involving multiple stakeholders, the formation of

the groups or collectives of agents is central to achieving either their individual objectives or some global objective. Assembling such groups of agents to collaborate has been the main focus of *coalition formation* techniques.

- *How to make joint decisions for coordinated actions.* In many cooperative applications, agents form a team that has to coordinate in a decentralised manner to perform the best actions. Research on *distributed constraint optimisation* has developed a clear formulation of the problem along with a significant number of algorithms to handle the problem.
- *How to allocate limited resources.* The assignment of resources within a system of autonomous agents that have preferences over alternative allocations of resources is a fundamental problem in MAS. Research on *multi-agent resource allocation* has contributed with a wealth of techniques to tackle this problem.

This paper is by no means intended to be a thorough survey on multi-agent optimisation (e.g. we do not cover bio-inspired approaches or optimization without communication), though we do provide references for the reader to undertake a deeper investigation of the problems and techniques described herein. This paper is a sequel to the "Optimization in Multi-Agent Systems" tutorial tought by the same authors at IJCAI 2011 in Barcelona[4]. At the end of this tutorial paper, the reader is expected to be able to answer the following questions regarding central issues on multi-agent optimisation:

Coalition formation.

- What is the coalition formation problem?
- What are the key steps of the coalition formation process?
- What are the features of application domains where coalition formation applies?

Distributed constraint optimisation.

- What is a distributed constraint optimisation problem (DCOP)?
- What are the state-of-the-art *exact techniques* for DCOPs?
- What are the state-of-the-art *approximate techniques*[5] for DCOPs?
- What are the benefits and limitations of those techniques?

Multi-agent resource allocation.

- What is a winner determination problem?
- What are the optimisation problems posed by the winner determination problem of state-of-the-art auction mechanisms?
- How can winner determination problems be solved?

The paper is organised as follows. First, Section 2 tackles the problem of assembling coalitions of agents. Then, Section 3 introduce the problems of assessing the optimal joint decisions that enable multi-agent coordination. Next, Section 4 analyzes how to assign resources to the agents in a MAS. Finally, Section 5 summarises and relates the multi-agent optimisation problems visited in this tutorial.

2. HOW TO ASSEMBLE COLLECTIVES: COALITION FORMATION

In many applications involving multiple stakeholders with common or individual objectives, the formation of the groups or collectives of actors is central to achieving these objectives. For example, in emergency management scenarios, it may be more effective to allocate small teams of first responders to multiple disaster sites rather than sending a large team to each site in succession. In the context of mobile sensor network applications, the best way to patrol an area may involve combining capabilities of different types of unmanned vehicles (aerial, ground, underwater) rather than using only one type.

While these examples involve actors or agents that are fully cooperative (i.e., will forego their own benefit for the common good — e.g., a mobile sensor will change its patrolling region in order to maximise the accuracy of the global patrolling effort even if it may have to travel more than the others in its team), in many cases, the actors may actually be self-interested (e.g., in collective energy purchasing schemes where each home is meant to pay a fair price for the energy it uses) and this may render the formation of collectives harder as they would need to agree on the terms that regiment their group actions.

In general, these applications are of particular interest to multi-agent systems research in that they involve actors that may:

1. Have distinct capabilities and local knowledge. For example, fire brigades will extinguish co-located fires while ambulances provide assistance to casualties.
2. Have their individual objectives. For example, each unmanned vehicle will aim to obtain the best information in its own goal area.
3. Need to come together at some point in time and space and disband to form different groupings at other points in time and space.
4. Need to engage in a negotiation process that will allow them to align their individual objectives (e.g.,

[4] Video recordings of that tutorial are freely available online at `http://videolectures.net/ijcai2011_t3_optimization/`.

[5] Approximated techniques are an important topic that pervades all optimisation problems. However, in this paper we consider approximation techniques only for DCOPs because, in contrast to the other themes, recent literature on DCOPs shows a significant interest in this direction.

through peer to peer messaging or through receiving commands from a centre). Such a negotiation process may be particularly important when the actors in the system may be self-interested and therefore require some form of payment or non-monetary reward (e.g., participants in a group-buying programme [1, 2] or collectives of wind generators where each generator belongs to a different stakeholder [3]). In these cases, the process is often termed a 'coalitional game' given the strategic decisions each stakeholder may make to join different collectives or coalitions.

These features define the key computational issues involved in forming collectives and, within the multi-agent systems paradigm, such issues are studied within the overarching theme of *coalition formation* [4]. Coalition formation is one of the fundamental approaches in multi-agent systems for establishing collaborations among agents, each with individual objectives and properties. Building upon the seminal work of Shehory and Kraus [5], Sandholm et al.'s [6] description of the coalition formation process is the most common framework used within the coalition formation literature to characterise the key computational tasks involved, namely:

1. Coalition value calculation and optimisation — this step forms the basis of the coalition formation process as it outputs a well defined value (that may represent a cost or profit) for each subset of the set of all agents. This value numerically captures how good the coalition is, for example, in completing tasks before a given deadline, in making a profit by selling or buying goods in bulk, or in obtaining information with a given degree of accuracy. Abstract approaches to coalition value calculation typically focus on the *process of enumeration* of coalition values while more practical approaches focus on the *optimisation of coalition members' actions* in order to achieve the optimal coalition value (e.g., minimum cost or maximum profit). While the process of enumeration is a well defined combinatorics problem that requires the manipulation of number sequences with simple arithmetic operations (e.g., to find the position of a coalition in a list of coalitions or generating all combinations of numbers from a dynamic list of numbers without repetition), the optimisation of member actions will depend on the type of application studied. For example, in the emergency response domain, the optimisation may involve coming up with a schedule of actions for each responder, while in the mobile sensor domain, the optimisation problem may involve finding the best orientation of sensors to patrol an area.
2. Coalition structure generation (CSG) — this involves partitioning the set of all agents so as to maximise the sum of the values of the chosen coalitions. Hence, if the value of a coalition determines the benefit the coalition brings to its members, maximising the total value generated by all coalitions is equivalent to concept of *social welfare maximisation*. This maximisation problem is termed the *optimal coalition structure generation problem*, which is a particular version of the well-known SET PARTITIONING problem, and is one of the hardest combinatorial optimisation problems to be studied in multi-agent systems.[6] At the core of the problem is the issue that the set of coalitions, in the worst case, is exponential in the number of agents, and this is the input to the coalition structure generation problem. Hence, while some approaches consider the complete input and attempt to navigate such a space using dynamic programming or heuristics, other approaches look at restricting this input in such a way that it still preserves the key features of the coalition formation process and facilitate the search for the optimal coalition structure.
3. Payoff Distribution — this step is mainly relevant to applications where the agents engage in a *coalitional game* i.e., each agent is self-interested and only joins a given coalition to improve its own utility. The optimisation problem here involves finding out the transfer or payment to each agent to ensure it is fairly rewarded for its contribution to its coalition and that it will not find it more beneficial to pair up with some agents or group of agents other than the one it has been assigned to. Thus the goal is to induce a *fair* or *stable* outcome to the process. Most of the literature in this space involves finding out how such payments can be calculated for coalitional games where the coalition value function may be complex and uncertain [7, 8] and proving that stability or fairness can be achieved.

Approaches to coalition formation in the multi-agent systems literature broadly involve some or all of the above features to different degrees. Thus, while points 1 and 2 deal with the *algorithmics* of coalition formation, point 3 deals with *computational economics* of coalition formation, where the focus is on establishing the computational costs of finding stable or fair outcomes. Hence, in this paper we will focus on points 1 and 2 and point the reader to [7] for further details on payoff distribution approaches.

In what follows, we describe the formal models involved, discuss the key computational challenges and how they may be solved, and point to the relevant literature in this area. The aim is to provide the reader with an understanding of the challenges rather than a detailed account of all approaches in the literature.

[6] It is a special case of the winner determination problem (see Section 4) where the input to the problem is exponential in the number of agents and where all agents (agents equal items in the winner determination problem) have to be chosen (not all items need to be chosen in the winner determination problem).

2.1. Coalition Value Calculation

Most research in coalition value calculation typically address the problem at an abstract level and assume the value of a coalition is given, though there are some notable exception focus on defining the coalition values according to particular needs of a given domain. In the next subsections, we will first consider the more theoretical approaches and then go on to describe in some detail other more practical approaches that consider complex task allocation problems.

2.1.1. General Algorithms

Assuming the value of coalitions will be given (e.g., by an oracle), the challenge in coalition value calculation is to enumerate all the feasible coalitions and efficiently distribute this computation among the agents in order to make the best use of their computational resources and to avoid bottlenecks in the system. Typically, this procedure is optimised by enumerating coalitions of each feasible size at a time, that is from size 1 to the number of agents in the system. In what follows, we introduce some formal definitions and then discuss the different approaches in the literature that attempt to generate such lists (instead of providing pseudocode of these algorithms, here we only refer the reader to the papers where these are described in detail).

Given a set of agents A, a coalition is a subset $C \subseteq A$ of agents. Note that subsets of size 1 (e.g., $\{a_1\}$ or $\{a_{10}\}$) are called *singleton* coalitions while the subset containing all agents is called the *grand coalition*. Enumerating all coalitions given a set of agents in then equivalent to enumerating all the subsets of the set of agents and the number of such subsets is exactly $2^{|A|} - 1$. Formally speaking, expressing the coalition value calculation process as the enumeration of coalitions of all sizes $s \in \{1, \cdots, |A|\}$:

$$L_s = \{C | s \in \{1, \cdots, |A|\} \wedge |C| = s \wedge C \in 2^A\} \quad (1)$$

where L_s is the list of coalitions of size s. As can be seen, such lists are drawn from the power set of agents and this enumeration process can quickly unravel as the number of agents grows. Thus, for only 20 agents, the number of coalitions is more than a million. Thus, to combat this complexity, it is important to come up with algorithms that are both efficient in computation and in memory requirements. Distributing the enumeration process across several nodes can also help reduce the computation time but can potentially increase communication requirements.

One of the first approaches to attempt to distribute the coalition value calculation process involved having every agent enumerate the subsets of agents they are part of and communicate to exchange the computed values and to avoid redundant computations of the same coalitions [4]. However, as [9] showed, such a process can be extremely expensive both in time and memory. Instead, it was shown by [9], that the enumeration process is very much similar to that required for the generation of Gray codes [10] whereby every combination of numbers can be generated from a previous combination with only one digit changed. This first requires the trivial assignment of integer ids to agents (where the highest agent id is the number of agents). By so doing, it is possible to enumerate all feasible coalitions and to do so without keeping any of the previously computed coalitions in memory. Thus to enumerate (and store) all the coalitions in A, one can use these simple steps as described by [9]. Such an algorithm can be distributed by assigning to each agent an index i to start computations from for each list of coalitions a given size s (the agent knows when to stop by simply calculating how many coalitions it should count according to an agreed split of computation). Furthermore, they show how this can be fairly distributed based on the known size of such lists and the computation power of each agent. The key benefit of this technique is that, in order to distribute the computation, each agent only needs to know at what index it should start enumerating coalitions from, for every coalition size s. For numbers of agents less than 30, the above enumeration process can be done within minutes.

2.1.2. Restricting Coalition Feasibility

The key problem with the above approach is that it assumes that *all subsets* of agents can exist as coalitions. It uses this fact to impose a strict procedure on the generation of the next coalition from the current coalition. Thus, if given a 100 agents and only about 100 coalitions are feasible, the above algorithm would have to go through all of the $2^{100} - 1$ coalitions in order to enumerate and check for the validity of each. This would be infeasible in practice. Recently, however, a new approach to coalition enumeration has been suggested whereby the 'feasibility' of coalitions can be described using a graphical approach, mimicking the kind of relationships that exist in communication or social networks [11]. This representation of feasible coalitions (as opposed to all real-valued coalitions) in this way gives rise to the *connected sub-graph enumeration* problem, whereby each coalition is a connected subgraph of a graph and the enumeration of all coalitions requires the enumeration of all subgraphs of the graph of agents. Voice et al., (2012) thus propose an algorithm to solve this problem efficiently and non-redundantly and suggest techniques to distribute the computation among agents as fairly as possible. Their techniques have been shown to speed up the coalition enumeration process by orders of magnitude, particularly when the agents are connected within trees of low branching factors.

2.1.3. Practical Coalition Value Calculation
Turning to coalition formation applied to more realistic settings we note that most related work in this space focus on task allocation problems. Thus, we note the work [4, 12] who proposed coalition value calculation algorithms where each agent in a coalition has to perform some tasks (at a certain cost) and the allocation of such tasks within each coalition needs to be optimised. Dang et al., [13], consider a similar problem in the domain of sensor networks where each coalition of agents requires them to orientate their sensors in a particular direction to achieve the maximum coverage as an aggregate, trading off their individual information gains in some respects. In more complex emergency response scenarios, [14] have also proposed techniques to compute coalition values based on the ability of a group of agents to save civilians. In their context, the value of a coalition is dictated by the schedule that each agent adopts to join the coalition at a certain time at a given location in order to dig out civilians.[7] In such a case, coming up with the optimum schedule for each coalition member is a problem that is strongly dependent on the process of deciding on which coalitions to form at different points in time, that is, what is the coalition structure that exists at each point in time. Hence, in the next section, we turn to approaches to solve the optimal coalition structure generation problem.

2.2. Optimal Coalition Structure Generation

The optimal coalition structure generation (OCSG) problem is a standard optimisation problem that is a special case of the set partitioning problem [15]. The key difference between the OCSG problem and the traditional formulation of the set partitioning problem is that the former typically looks at the fact that *all* subsets of agents are feasible while the latter typically assumes there are large numbers of items (or agents) that can be partitioned into small numbers of subsets. In recent years, a number of approaches have also looked at the middle ground between these two extremes, where not all coalitions are feasible or where the coalition values are structured in such a way that it is easier to identify the optimal coalition structure.

Formally speaking, the optimal coalition structure generation problem finds the solution to:

$$\arg \max_{CS \in \mathcal{F}(G)} \sum_{C \in CS} v(C) \tag{2}$$

where v is the *characteristic function* that returns the value of a given coalition (where this value is independent of the membership of other coalitions), G is the set of all partitions of the set of agents A, $\mathcal{F}$ returns the subset of these coalition structures that contain feasible coalitions (we elaborate on this point later), $CS \in G$ is a coalition structure, i.e., $CS \subseteq 2^A$ where for any $C_i, C_j \in CS$, $C_i \cap C_j = \emptyset$, i.e., no agent is assigned to more than one coalition.

There are two important points to note from the above formulation. First, it assumes that the value of a coalition is independent of any other coalition chosen in the coalition structure, i.e., the $v(C) \in \Re^+$ is a *characteristic function*, which may not always be the case [16]. Formally, for non-characteristic functions, this means assigning the value of to coalition conditioned on the coalition structure it is contained within, i.e., $v(C|CS)$, where $C \in CS$. This renders the problem significantly harder to tackle but is typically regarded as a special case. Second, the above formulation assumes that each agent can only form part of only one coalition at any time. Again, in settings where agents can participate in multiple teams at the same time (e.g., mobile sensors providing information to different teams at the same time, or rescuers helping to recover casualties from under rubble while providing resources to support other teams' communications). These settings are typically regarded as *overlapping* coalition formation settings [17, 4] and typically relate to the well known a set covering problem [15].

In the next subsections, we focus on the most common coalition structure generation cases that involve a characteristic coalition value function and problems where the set of coalitions or coalition value functions may be restricted by some arbitrary domain-specific feature.

2.2.1. General Solutions
The general OCSG problem can be represented in a number of ways, leading the way to different solution approaches. The most straightforward approach to try and solve the problem would be to use a mixed-integer programming (MIP) solver such as IBM ILOG's CPLEX using equation 2 as the objective function and specifying constraints on the membership of agents to exactly one coalition at a time (see details in [18]). However, doing so would not exploit the fact that larger coalitions can naturally be dissected into their smaller parts, which would only need to be evaluated once. In more detail, one of the first representations for the problem was provided by Yun Yeh [19] for the (complete) set partitioning problem whereby, coalitions are categorised according to their size and the traversal of the space of coalitions can simply be done by looking into subdivisions of each coalition, using a dynamic programming (DP) approach (which was recently improved by Rahwan and Jennings [20] to avoid going through redundant subdivisions). The latter grows in $O(3^{|A|})$ in both memory and time and does not allow for anytime computation of a solution.

Now, Sandholm et al. [6] and Dang and Jennings [12], also proposed other ways to represent the problem using

[7] The setting they use is that provided by the RoboCupRescue simulation environment, a realistic disaster simulation platform.

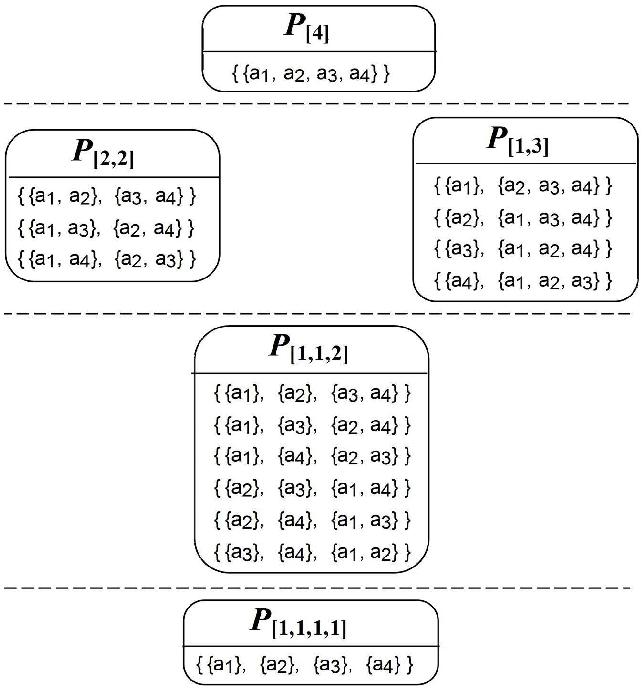

FIGURE 1: Space representation proposed by Rahwan et al. [18], where P_i represents the search space containing coalition structures of a particular configuration fitting an integer partition of the number 4. The search space is split into levels to separate coalition structures in terms of the number of coalitions they contain as used in [21] for example.

the concept of a coalition structure graph, where levels of the graph are defined according to the maximum size of coalitions in each level of the graph. However, even though their approach allows for anytime solutions with well defined worst-case bounds, they are not efficient in finding the optimal solution (grows in $O(|A|^{|A|})$ and do not scale beyond 15 agents as Rahwan et al. showed [9, 18].

Instead, in recent years, a novel approach to representing the problem has emerged using the notion of integer partitions (IP). Thus, it was noted that coalitions in a coalition structure adhere to well defined sizes represented by the integer partitions of the number of agents $|A|$, that is, $\{|C|, C \in CS\}$ is equivalent to a given integer partition of A. Hence, for 3 agents, the integer partitions are $\{1,1,1\},\{1,2\},\{3\}$ and, these can be used to define categories where, for example, a coalition structure $\{\{1\},\{2\},\{3\}\}$, would fit the category $\{1,1,1\}$, and $\{\{1\},\{2,3\}\}$ would fit $\{1,2\}$. Hence, it is possible to partition the search space of all coalition structures by assigning coalition structures to their category according the sizes of the coalitions they involve. A more elaborate example with 4 agents is given in Figure 1. As Rahwan et al. showed, such a representation facilitates the computation of bounds (linear in the input) for individual parts of the search space and therefore help prune the space in an effective way using branch-and-bound techniques. Using such techniques, it was then possible to scale up solutions to the coalition structure generation problem from 15 agents to 27 agents and to generate high-quality solutions anytime. Furthermore, in later endeavours, the approach was further combined with the DP approach above to further improve the search strategy [22] and extended to allow for the distribution of the computation of the optimal coalition structure [23]. However, a number of issues remain. In the worst case, IP algorithms will have to go through all coalition structures to return the optimal solution. In addition to this, due to the lexicographic ordering techniques it employs, the IP approach and extensions of it are all limited by the fact that they require that all coalitions be feasible in order to navigate through the search space efficiently. Thus, such issues restrict these techniques to solving problems involving less than 27 agents. Hence, in the next subsection, we turn to those approaches that try to solve the more general problem where not all coalition structures are feasible and where there may be restrictions on the coalition value function.

2.2.2. Restricted Structures

At the foundations of the OCSG problem lies the issue that the input to the problem is exponential (i.e., the list of all feasible coalitions). Hence, building upon the coalition value calculation work presented earlier, recent approaches have looked at finding the optimal coalition structure when the coalition value function takes a specific shape that precludes the selection of specific coalitions in the optimal structure. This includes imposing restrictions on the individual contribution of an agent to any coalition [24, 25] or on the feasibility of coalitions (i.e., whether some coalitions can be formed or not) without restricting the value an agent brings to the coalition. We focus on the latter approach as it can effectively consider any coalition value function (i.e., those coalitions that are infeasible simply have an infinitely negative value). The state of the art in this respect is the synergy graph representation from Voice et al. [11] where they apply a social network structure as shown in Figure 2. These links between the agents imply constraints that may be due to communication constraints (e.g., non-overlapping communication loci or energy limitations for sending messages across a network), social or trust relationships (e.g., energy consumers who prefer to group with their friends and relatives in forming energy cooperatives), or physical constraints (e.g., emergency responders that have enough fuel to join only specific teams or have life-saving capabilities that match only a limited number of other responders). Formally, agents can form coalitions $C \subseteq A$, however, the set of feasible coalitions, $\mathcal{C}$, is constrained by a graph $G = (A, E)$, where E is a set of edges between agents. We consider the situation where a coalition of agents C is feasible if and only

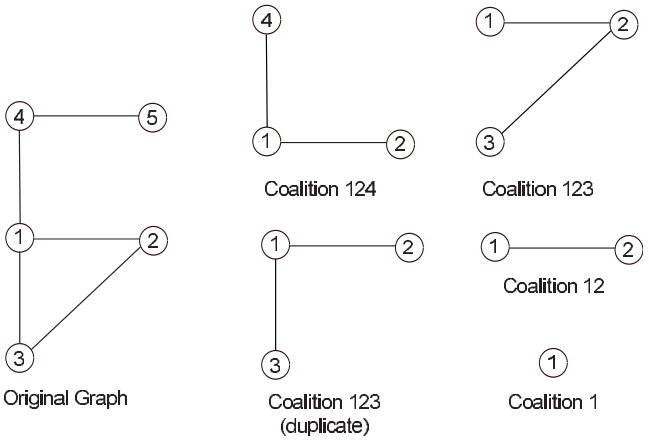

FIGURE 2: Synergy graph example for 5 agents from Voice et al. [11].

if there exists a connected subgraph $G' = (C, E')$ with edges $E' \subseteq E$ and vertex set exactly equal to C. The objective function for such a problem is given as in (2), redefining $\mathcal{F}$ as the function that extracts all feasible coalition structures from the the graph G.

As can be seen in Figure 2, the network restricts the formation of coalitions that do not have an underlying connected subgraph connecting *all* agents in the coalition. For example, coalitions $\{2, 5\}$ cannot be formed because no edge exists between 2 and 5 while coalition $\{1, 2, 4, 5\}$ can exist because there is an edge linking 1 and 4, one linking 1 and 2, and another edge linking 4 and 5. In the worst case, the graph is complete (i.e., there is an edge between every pair of agents), which then maps back to the original problem detailed in the previous section. However, when the graph is sparse, in such structures as in trees, small worlds, or lines, the number of feasible coalitions is significantly reduced and also permit the faster computation of the optimal coalition structure. In effect, the algorithms proposed by Voice et al. has been shown to permit the enumeration of up to 50 agents (on trees) in less than 15 minutes, and the computation of the optimal coalition structure for up to 30 agents within days, in the worst case (on a complete graph) and in less than 6 minutes in the best case (on trees). The algorithm they proposed for OCSG on sparse synergy graphs (DyCE) is very similar to the IDP algorithm proposed by Rahwan et al. [20]. However, contrary to IDP, DyCE avoids enumerating a large number of infeasible coalition structures as it uses the structure of the network to iterate only through those that contain feasible coalitions.

It is also to be noted that graph-restricted structures as described above have also been the subject of studies in the economics literature as they pose interesting challenges for payoff distribution mechanisms and the design of generalised solution concepts. In this context, it is highly recommended to complement the design and analysis of graph-restricted OCSG algorithms with some background and requirements from [26, 27, 28, 29].

2.3. Challenges and Further Readings

So far, we have described the key steps of the coalition formation process and how it can be useful in addressing specific collaboration challenges in real-world applications. Our survey of some of the recent literature also shows that there is a dearth of algorithms to solve coalition formation problems for domains presenting issues of scale, where hundreds or thousands of agents may need to be grouped into coalitions in the most effective manner. Moreover, it is not clear how the summarisation of the effectiveness of a coalition into a value function is at all useful in real-world applications that may involve humans each with their own understandings of the world around them and of their preferences to interact with team members they may be familiarly with. Finally, in this paper we have only focused on one-shot coalition formation algorithms and ignored the fact that many coalition formation problems may require coalitions to disband and reform over time, leading to challenges in defining the sequentially optimal way to form such coalitions [30]. Hence, we summarise some of the key challenges for coalition formation research as follows:

1. There is a need to design efficient anytime algorithms for domains with large numbers of agents beyond hundreds, in order to provide realistic solutions to coalition formation problems involving artificial or human agents.
2. Better representations are need to capture the value of forming coalitions involving human agents operating alongside software agents and robots.
3. Algorithms for sequential formation of coalitions over time, particularly when the value of future coalitions may be liable to significant degrees of uncertainty.

Now, the parts covered in this tutorial are limited to the basics of the state of the art algorithms and do not delve into the details of the algorithms intentionally in order to give a flavour of the key representations and algorithms. In order to get an in-depth understanding of particular algorithms and mechanisms, the reader is referred to chapters 8 and 17 of [31] as well as the following:

1. Coalition value calculation: the key algorithms for the enumeration of sequences of numbers are covered by Knuth in his well known book series [10]. The PhD thesis of Rahwan [32] also contains a wealth of analysis of more elaborate coalition enumeration problems (including dynamic ones) and discussions on the topic.
2. Coalition structure generation: most of the latest algorithms have been mentioned already and, as

yet, one of the most comprehensive benchmarks and comparisons of the most important approaches are given in Rahwan's thesis [32] which is a must-read for the topic.

3. Payoff distribution: we did not explicitly deal with this topic in this tutorial as this is extensively covered in the recent publication by Chalkiadakis et al. [7] which covers the analysis of the computational aspects of cooperative game theoretic solutions to the problem of coalition formation.

3. HOW TO MAKE JOINT DECISIONS: DISTRIBUTED CONSTRAINT OPTIMISATION

When agents operate in the context of a team they must take joint rather than single decisions to maximize a system-wide objective. Teams of agents arise whenever agents' preferences are aligned with the system objective, this is often the case in emergency management, surveillance and more in general for applications involving a single stakeholder that is responsible for designing the whole system (e.g., most applications involving roots or embedded devices). In several situations, the computation of these coordinated actions may be a necessary step to address more general problems. For example, as discussed in section 2, one critical aspect of coalition formation is how to assess the value of a coalition which may go from a very simple computation to a hard optimization problem involving the coordination of agents actions [33]. Similarly, in the auction mechanisms presented in section 4 bidders may need to coordinate internally to assess the value for which they bid for a service (e.g. a company may need to coordinate their carriers to assess the logistic cost of delivering the services for which they are bidding).

Choosing the individual actions that jointly maximize a system wide objective is a key issue for coordination and there are several frameworks that can be used to tackle this problem (e.g., Sequential Decision Making [34, 35], Multi-Agent Planning [36, 37], to name a few). In this tutorial paper we focus on constraint processing [38] and more specifically on Distributed Constraint Optimization Problems (DCOPs) [39, 40] for several reasons: i) DCOPs have a strong focus on decentralised approaches where agents negotiate a joint solution through local message exchange; ii) the proposed solution techniques exploit the structure of the domain (by encoding this into constraints) to tackle hard computational problems; iii) there is a wide choice of solutions for DCOPs ranging from complete algorithms [41, 42] to suboptimal algorithms [43, 44, 45, 46, 47]. Such approaches have been widely studied and applied in many reference domains (disaster management, traffic control, intelligent light control or energy-efficient sensor networks). To provide the reader with a gentle introduction to DCOP formalism, in what follows we introduce a case study inspired by disaster management and use this to exemplify the key elements of a standard DCOP formalization.

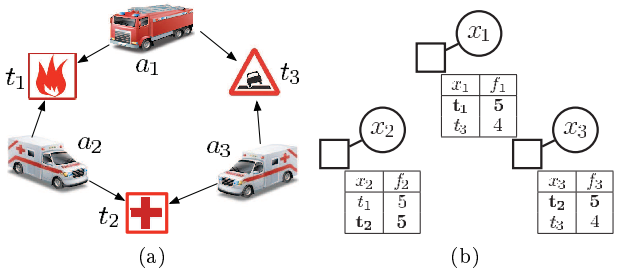

FIGURE 3: a) A diagram showing an exemplar coordination scenario and b) a factor graph corresponding to a possible DCOP formulation considering no synergies among agent's actions.

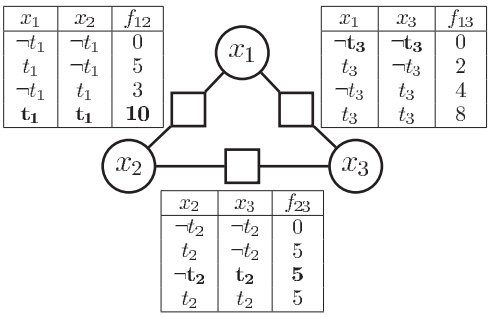

FIGURE 4: A factor graph corresponding to a possible DCOP formulation for the scenario in Figure 3a considering pairwise dependencies among agent's actions. Optimal assignments are boldfaced.

3.1. Case study: Coordination in disaster management

Consider a set of rescue units, one fire brigade and two ambulances, that need to handle the disaster management scenario depicted in Figure 3a. At some point in time three rescue tasks are reported in the disaster area: one building on fire (t_1), injured people (t_2) and a road blocked with rubble (t_3). The fire brigade is required for tasks t_1 and t_3. Ambulances are required in all tasks, but the first ambulance can only reach t_1 and t_2 whereas the second can only access t_1 and t_3 (i.e., only the ones within a given travel distance from their current position). We can model this specific problem as a DCOP by defining the following five components:

- *Agents* ($\mathcal{A}$). Each rescue unit is modeled as an agent a_i.
- *Variables* ($\mathcal{X}$). The individual decision of each rescue unit is modeled with a discrete variable x_i, which represents the next task to be attended by the agent that controls such variable.

- *A set of discrete and finite domains ($\mathcal{D}$).* In the example, the domain of a variable x_i, D_i, contains one value for each possible task that each agent can tackle.
- *A set of constraints ($\mathcal{F}$)* where a constraint is a function $f_V(\mathbf{x}_V) : D_{j_1} \times \cdots \times D_{j_{|V|}} \rightarrow \Re \cup \{-\infty\}$ that assigns a real value for each possible joint assignment of the variables it depends on ($x_V \subseteq \mathcal{X}$) quantifying the relative utility for the system[8]. Hence, in the following we will refer to these as utility functions[9]. In the example utility functions quantify the gain in terms of avoiding casualties and reducing damages to infrastructures that correspond to the joint agent action.
- *A function $M : \mathcal{X} \rightarrow \mathcal{A}$ that maps each variable to the controlling agent.* For the sake of simplicity in all the examples given in this tutorial each agent a_i controls a discrete variable x_i but in general each agent can control several variables (although each variable is controlled by a single agent).

Given this, agents aim at finding the joint variable assignment that maximizes the sum of utility functions, i.e. $\mathbf{x}^* = \arg\max_{\mathbf{x}} \sum_{f_V \in \mathcal{F}} f_V(\mathbf{x}_V)$.

A key concept for DCOPs is the factor graph, a bipartite graph that graphically represent agents' interactions. A factor graph is a bipartite graph with two types of nodes: variable and function nodes. Each function node is linked to a variable node if the variable belongs to the function scope. When functions are all defined over two variables, factor nodes are simply omitted linking the two variables directly in the graph, and the graphical representation is then called constraint graph. Given an agent a_i, the agents directly connected with a_i in the constraint/factor graph are the *neighbours* of a_i.

Figures 3b depicts a factor graph for a scenario in which the utility of each rescue unit to attend a task is independent from the assignment of other rescue units to that task (there is no synergy among their actions). Tables associated with rescue unit decision variables report the utility for each of the possible task that the unit can execute. For instance, the fire brigade unit (x_1) receives a utility of 5 when assigned to t_1 and a utility of 4 when assigned to t_3. With independent utilities agents do no need to coordinate: each agent just takes the task that gives the highest utility. Thus, x_1 chooses task t_1 (with a utility of 5), x_2 chooses task t_2 (with a utility of 5) and x_3 chooses task t_2 (with a utility of 5). The computational complexity of each agent when solving the DCOP is linear with respect to its number of actions $\mathcal{O}(|D_i|)$ (each agent needs to go through all actions and pick the one with maximum utility).

As the reader may have noticed, the assumption of individual utilities taken in scenario 1 does not hold in many real-world situations. For example, assume that in the building on fire (t_1) there are injured people that can only be saved by the joint action of the fire brigade and an ambulance unit. Thus, the fire brigade and the ambulance unit will receive an utility of 10 only if both of them are assigned to t_1, however, if only the ambulance is assigned to t_1 then this utility is reduced to 3. Thus, the actions of the two agents in this case are complementary. In other tasks, the actions of agents may not be complementary but rather substitutable. For example, consider that to help injured people (t_2) only one ambulance is required. Then the two ambulances do not get 5 each when assigned to t_2 but rather 5 together. Figure 4 reports the factor graph corresponding to the DCOP formulation for such scenario. There is one utility function representing each task, and each function depends only on the subset of variables that correspond to agents that can execute the task. Observe that now the joint agent action that would be optimal in scenario 1 is no longer optimal here. Thus, to solve this maximization task, agents need to collaboratively solve the underlying DCOP. In the next Section we present the most prominent complete approaches that can be used when aiming for the optimal solution.

In the remainder of this section we will discuss a subset of the available solution techniques for DCOPs. This is not an exhaustive survey of DCOP solution approaches but rather an overview of different solution patterns that are of particular interested for people working in the field. In more detail, we start from complete solution techniques (e.g., techniques that are guaranteed to compute the optimal solution) and then proceed on suboptimal approaches.

3.2. Distributed constraint optimization: complete algorithms

In the following we present three distributed search algorithms [48, 41, 49] and one distributed inference algorithm [42] for DCOP solving (more algorithms have been proposed, we restrict ourselves to these four to give a representative sample of solving strategies). Regarding distributed search algorithms, we focus our attention on asynchronous ones, where agents communicate among them at any time, and no agent has to wait for any other agent. We start with ABT [48] because it was the first distributed algorithm for DCOP solving (assuming a special case). We provide a kind of informal –but accurate– description of these algorithms. For a deeper or more formal treatment, the reader is referred to the original sources.

To be aligned with relevant literature on complete DCOP approaches, in this Subsection we assume a

[8] Here the index of a function is the set of indices of its scope variables. Hence, V is a subset of variable indices and we use it to indicate functions (e.g., $f_{12}(x_1, x_2)$) and to specify subsets of variables (e.g., $\mathbf{x}_{12} = \{x_1, x_2\}$).

[9] If constraint functions represent costs, as it is often the case in DCOP literature, then the objective function is a minimization task and we refer to these functions as cost functions.

slightly different –but equivalent– DCOP definition, replacing utility functions by cost functions. An optimal solution is a complete assignment (involving all variables) that minimizes the global cost, computed as the addition of all cost functions. For simplicity, we also assume that cost functions are binary, although the following approaches can be easily extended to deal with cost functions of any arity.

3.2.1. Distributed constraint satisfaction: ABT

There is a special case of distributed optimization when the possible costs associated with any value tuple are only two: either 0 or ∞. This case is known as *distributed constraint satisfaction* (DisCSP) because cost functions become *constraints* made of *permitted/forbidden* tuples (any value pair is either permitted –it costs 0– or forbidden –it costs ∞–), and the obvious optimum is when all selected variable assignments are taken from permitted pairs so the global cost is 0.

The pioneer algorithm to solve this kind of problems is asynchronous backtracking (ABT) [48]. ABT is an asynchronous algorithm executed in each agent of the constraint network. Because it is asynchronous, during ABT execution agents may change their values at any time, without generating any malfunction. It has been proved formally that ABT is a correct and complete algorithm for DisCSPs solving. In general, the advantage of search algorithms is that they require polynomial memory. Their downside is that they may produce a very large number of small messages, resulting in large communication overhead.

ABT requires (*i*) a total agent ordering and (*ii*) constraints to be directed. A constraint between two agents, a_i and a_j, a_i higher than a_j in the ordering, is depicted as a directed link starting from a_i –called the value-sending agent– and arriving to a_j, –called the constraint-evaluating agent–. In this way, it is assured that the network is cycle-free.

The operation of ABT inside each agent is as follows. Each agent keeps its own agent view and nogood store. The agent view of a_i is the set of values that a_i believes that are assigned to agents connected to a_i by incoming links. The nogood store is a memory of received nogoods.[10] These nogoods are justifications of why some values are inconsistent.

ABT is based on message passing. The types and structure of ABT messages appears in Figure 5. Agents exchange assignments and nogoods, using OK? and NGD messages. An OK? message contains a value assignment, while a NGD message contains a nogood. Their usage is as follows. When a_i makes an assignment, it informs those agents connected to it by outgoing links using OK? messages. When a_i receives OK? messages, it always accepts their information, updating its agent view accordingly. When a_i receives a NGD message, the contained nogood is accepted if it is consistent with the agent view of the receiver –including its own value–, otherwise it is discarded as obsolete. An accepted nogood is added to the nogood store of the receiver, to justify the deletion of the value it targets. When a_i cannot take any value consistent with its agent view, because of the original constraints or because of the received nogoods, new nogoods are generated as inconsistent subsets of the agent view, and are sent to the closest agent involved, causing backtracking. The simplest way to generate an inconsistent subset is *resolving* all nogoods [50]. If a_i receives a nogood mentioning another agent not connected with it, a_i requires to add a link from that agent to a_i, using the ADDL message. From this point on, a link from the other agent to a_i will exist (although there is no constraint between them). The process terminates when achieving quiescence, meaning that a solution has been found, or when the empty nogood is generated, meaning that the problem is unsolvable. While the empty nogood is detected by ABT (and the agent noticing it sends STOP messages to all its neighbors, which resend it before stopping) quiescence is detected by external algorithms.

OK?(*sender,receiver,value*)
NGD(*sender, receiver, nogood*)
ADDL(*sender, receiver*)
STOP(*sender, receiver*)

FIGURE 5: Messages used by the ABT algorithm

VALUE(*sender, receiver, value*)
COST(*sender, receiver, lb, ub, context*)
THRESHOLD(*sender, receiver, threshold*)
TERMINATE(*sender, receiver*)

FIGURE 6: Messages used by the ADOPT algorithm

3.2.2. Distributed constraint optimization: ADOPT

When cost functions may output any non-negative value, we are in the general case of distributed constraint optimization. In this case, the ADOPT algorithm [41] is able to solve this problem; it has some historical relevance because it was the first proposed algorithm for optimal DCOP solving in 2003.

ADOPT (acronym for Asynchronous Distributed OPTimization) assumes that agents are arranged in a DFS tree (acronym for Depth First Search tree), which is a special case of pseudotree with all its links belonging to the constraint graph. As pseudotree, the DFS tree

[10] A nogood is simply a conjunction of assignments that has been found inconsistent; often nogoods are written in directed form, as a logical implication with left and right-hand sides. The right-hand side of the nogood mentions the assignment of the last agent in the ordering, while the left-hand side is the conjunction of all the other assignments. This conjunction is the justification for the last agent in the ordering not to take the mentioned value.

has the property that constrained agents have to be in the same branch. In this way, independent subproblems are naturally located in different branches. The DFS tree can be found –by distributed algorithms– from a depth-first traversal of the constraint graph, where a subset of links are the arcs of a tree and the rest of links are pseudo-arcs. Although more than one path may exist between two nodes, there is only one path if we limit ourselves to these arcs. Finding the DFS tree of minimum height is an NP-hard problem [51], although several heuristics can be used to compute DFS trees of reasonable quality.

In a preprocessing phase, the DFS tree is built. At this point, each agent knows its parent and pseudoparents (ancestors connected by arc/pseudoarcs), and also its children and pseudo-children (descendants connected by arcs/pseudo-arcs). Then, ADOPT execution starts. An agent takes the most promising value, as the one with minimum lower bound. Agents exchange information using VALUE and COST messages. The whole process terminates when agents find a value for which its bounds are equal.

In the following we describe the ADOPT algorithm in more detail. We detail its internal data structures, communication through message passing, and its coherence detection among messages.

DATA STRUCTURES. Each agent a_i maintains for its variable x_i: its current value v_i; its current context X_i (= recorded current values of its ancestors); the lower and upper bounds $LB_i(v)$ and $UB_i(v)$ for all values $v \in D_i$, given that a_i takes the value v and its ancestors take their respective values in X_i; the lower and upper bounds $lb_i^c(v)$ and $ub_i^c(v)$ for all values $v \in D_i$ and children a_c; the thresholds TH_i and $th_i^c(v)$ for all values $v \in D_i$ and children a_c, which are used to speed up the solution reconstruction process. Formally, bounds expressions are,

$$LB_i(v) = \delta_i(v) + \sum_{x_c \in CH_i} lb_i^c(v) \qquad LB_i = \min_{v \in D_i} \{LB_i(v)\}$$

$$UB_i(v) = \delta_i(v) + \sum_{x_c \in CH_i} ub_i^c(v) \qquad UB_i = \min_{v \in D_i} \{UB_i(v)\}$$

$$\delta_i(v) = \sum_{(x_j, v_j) \in X_i} F_{ij}(v, v_j) \qquad v_i = \arg\min_{v \in D_i} \{LB_i(v)\}$$

for all values $v \in D_i$, where CH_i is the set of children of agent a_i and $\delta_i(v)$ is the sum of the costs of all cost functions between a_i and its ancestors given that a_i takes the value v and the ancestors take their respective values in X_i.

At the start, each agent x_i initializes its current context X_i to $\emptyset$, lower and upper bounds $lb_i^c(d)$ and $ub_i^c(d)$ to user-provided heuristic values $h_i^c(d)$ (or 0) and ∞, respectively. The current context and lower and upper bound tables are updated as messages arrive.

Each agent has a threshold, informed by its parent (except the root, for which it is always ∞). Thresholds are used to decide when to change value. When an agent takes a new value, it informs its children of thresholds associated to its new value.

COMMUNICATION. ADOPT is based on message passing. The types and structure of ADOPT messages appears in Figure 6. When an agent takes a value, it informs of it to its children and pseudochildren though VALUE messages. Upon reception, the receiver agent sends a COST message to its parent, with its lower and upper bounds and its current context. When an agent receives a COST message (always from one of its children), it updates its internal tables, changes value if needed, and sends again VALUE messages to its children and pseudochildren, and a COST message to its parent. This process is repeated at every agent until the root finds that its lower and upper bounds are equal. This means that the optimum cost has been found, and the algorithm may terminate execution. The root sends TERMINATE messages to its children, which send again TERMINATE messages to their children and terminate execution after reaching that their lower bounds are equal to their upper bounds. It has been proved that ADOPT always terminate, producing the optimum cost [41]. In addition, each agent takes the value that minimizes the global cost.

OPERATION. Agents exchange messages, and this causes agents to update their internal data structures. When the current context becomes incompatible (differing in the value of at least one variable) with the context of a child, the lower and upper bounds, and the context of this child are reinitialized. When the context of a COST message is compatible with the current context, its bounds are copied in the internal tables of the receiver agent. Otherwise, these tables are not updated, and the message is discarded. After processing either VALUE or COST message, it recalculates the remaining lower and upper bounds and takes its best value using the above equations and sends VALUE and COST messages.

Due to memory limitations, each agent a_i can only store lower and upper bounds for the current context. Thus, it reinitializes its bounds each time the context changes. If its context changes back to a previous one, it has to update its bounds from scratch. ADOPT optimizes this process by having the parent of a_i send a_i the lower bound computed earlier as threshold TH_i in a THRESHOLD message. This optimization changes the condition for which an agent changes its value. Each agent a_i now changes its value v_i only when $LB_i(v_i) \geq TH_i$.

This process repeats until the root agent x_r reaches the termination condition $LB_r = UB_r$, which means that it has found the optimal cost. It then sends a TERMINATE message to each of its children and terminate. Upon receipt of a TERMINATE message, each agent operates until reaching the termination condition, and it sends TERMINATE messages to its children (which again perform a similar process ...).

3.2.3. Distributed constraint optimization: BnB-ADOPT

The BnB-ADOPT algorithm [49] shares most of the data structures and messages of ADOPT. The main difference is their search strategies. ADOPT employs a *best-first search* strategy while BnB-ADOPT employs a *depth-first branch-and-bound search* strategy. This difference in search strategies is reflected by how agents change their values. While each agent a_i in ADOPT eagerly takes the value that minimizes its lower bound $LB_i(v)$, each agent a_i in BnB-ADOPT changes its value only when it is able to determine that the optimal solution for that value is provably no better than the best solution found so far for its current context. In other words, when $LB(v_i) \geq UB_i$ for its current value v_i.

The role of thresholds in the two algorithms is also different. As described earlier, each agent in ADOPT uses thresholds to store the lower bound previously computed for its current context such that it can reconstruct the partial solution more efficiently. On the other hand, each agent in BnB-ADOPT uses thresholds to store the cost of the best solution found so far for all contexts and uses them to change its values more efficiently. Therefore, each agent x_i now changes its value v_i only when $LB_i(v_i) \geq \min\{TH_i, UB_i\}$.

3.2.4. Distributed constraint optimization: DPOP

The DPOP (acronym for Distributed Pseudotree Optimization Procedure) algorithm [42] was the first DCOP inference algorithm based on dynamic programming. More concretely, DPOP performs variable elimination on a DFS in a distributed fashion. Thus, as the previous ADOPT and BnB-ADOPT, DPOP works on a DFS tree of the constraint graph. The advantage of inference algorithms with respect to search approaches is that they require a linear number of messages. Their downside is that the messages agents exchange may be exponentially large.

Before describing the algorithm, some concepts are needed. An *assignment or tuple* t_S with scope S is an ordered sequence of values, each corresponding to a variable of $S \subseteq X$. The *projection* of t_S on a subset of variables $T \subset S$, written $t_S[T]$, is formed from t_S by removing the values of variables that do not appear in T. This idea can be extended to cost functions: the projection of f_V on $T \subset V$, is a new cost function $f_V[T]$ formed by the tuples of f_V removing the values of variables that do not appear in T, removing duplicates and keeping the minimum cost of the original tuples in f_V. The *concatenation* of two tuples t_S and t'_T, written $t_S \cdot t'_T$, is a new tuple with scope $S \cup T$, formed by the values appearing in t_S and t'_T. This concatenation is only defined when common variables have the same values in t_S and t'_T. The *cost* of a tuple t_X (involving all variables) is $\sum_{f \in \mathcal{F}} f(t_X)$, that is, the addition of the individual cost functions evaluated on t_X (implicitly, it is assumed that, $f(t_X) = f(t_X[var(f)])$, where $var(f)$ is the set of variables that form the scope of f). Two operations on functions are defined,

1. *Projecting out* a variable $x \in V$ from f_V, denoted $f_V[-x]$, is a new function with scope $V - \{x\}$ defined as projecting f_V on $V - \{x\}$, $f_V[-x] = f_V[V - \{x\}]$.
2. *Adding* two functions f_V and g_W is a new function $f + g$ with scope $V \cup W$ and $\forall t \in \prod_{x_i \in V} D_i$, $\forall t' \in \prod_{x_j \in W} D_j$ s.t. $t \cdot t'$ is defined, $(f+g)_{V \cup W}(t \cdot t') = f_V(t) + g_W(t')$.

DPOP performs three phases in sequence: (1) DFS phase. An agent is selected as root (for instance, by a leader election process). From this agent, a distributed DFS traversal of the constraint graph is started. At the end, each agent labels its neighbors as parents, pseudoparents, children or pseudochildren. (2) UTIL phase. Each agent (starting from leaves) sends a UTIL message to its parent, that contain an aggregated cost function computed adding received UTIL messages from its children with its own cost functions with parent and pseudoparents. The sent cost function does not contain the agent's variable, which is projected out. (3) VALUE phase. Each agent determines its optimal value using the cost function computed in phase 2 and the VALUE message received from its parent. Then, it informs its children using VALUE messages. The agent at the root starts this phase.

3.3. Distributed constraint optimisation: sub-optimal algorithms

In many real world applications we cannot afford to pay the price of an exponential coordination overhead to achieve optimal solutions. This is particularly true in applications that must deal with real-time constraints and dynamism, such as coordination in disaster management or applications involving low power devices that exhibit severe restrictions in terms of computation and communication capabilities.

In these settings, sub-optimal approaches are usually preferred [43, 44, 45, 46, 47, 52, 53, 54]. These algorithms are based on local information exchange and can provide good solutions while being efficient in terms of computation and communication. Here we will briefly describe two of these approaches: local search and sub-optimal inference-based approaches. Finally, we will describe region optimality, a framework used to define quality guarantees on the sub-optimal solutions returned by these algorithms.

3.3.1. Local search algorithms

In local search [55] agents initialize their variables with a random assignment and then they perform a series of *local* moves to optimize the objective function. For example, in Figure 4 assume that agents depart from the initial assignment $\mathbf{x}^0 = \{x_1 = t_3, x_2 = t_2, x_3 = t_3\}$. A

local move involves changing the joint assignment of the variables corresponding to a subset of the agents (aka *neighborhood*) to optimize the local *gain* (the difference between the sum of local utility functions evaluated with the new assignment with respect to the previous one). Now, with respect to Figure 4, consider changing the assignment of variable x_1 from t_3 to t_1. This local move has a local gain of 1: there is an increment utility of 5 when assigning a_1 to t_1 (given by function f_{12}) and an utility reduction of 4 when not assigning it to t_3 (since a_3 is still assigned to t_3, function f_{13} still returns some utility). This is the only local move involving a single variable that has a positive gain in this case: moving a_2 from t_2 to t_1 gives a negative gain of -2 (we get an utility increment of 3 by assigning it to t_1 but we lost all the utility given by t_2 since a_3 is not assigned) and moving a_3 from t_3 a t_2 gets a negative gain of -6. Thus, the selected local move leads to new assignment $\mathbf{x}^1 = \{x_1 = t_1, x_2 = t_2, x_3 = t_3\}$. Local search algorithms iteratively repeat this step until the gain can not be further improved by performing any local move (i.e., when the search reaches a *local* maxima). Following our example, the algorithm will not perform any further moves that involve changing the assignment of a single agent since the obtained solution can not be improved. Observe that the current assignment is not optimal, and the algorithm results in a *local* maxima.

When local search techniques operate in a decentralized context, agents evaluate and perform local moves in parallel, informing their neighbours of the new assignment after each move. Notice that such parallel execution without coordination can results in poor system performance since agents can act based on out-of-date knowledge about the choices of other agents. Two solutions have been proposed for this problem each of them leading to a different algorithm in the DCOP community: DSA [43] and MGM [44]. DSA introduces stochasticity by only allowing each agent to optimize its local gain if a random number exceeds a certain activation probability in that iteration. In contrast, MGM performs an explicit coordination phase to decide who is in the best position to perform the local move in an agent neighbourhood (i.e., the agent with the maximum local gain). Empirical evidence indicates that MGM has comparable performance to DSA, however MGM does not require parameter tuning and it is guaranteed to have anytime performance [44].

Notice that, the work by Maheswaran et al.[44] proposes a general framework that advocates game theoretic approaches for DCOPs. In such framework, DCOPs are represented as graphical games [56] and locally optimal solutions for a DCOP are shown to be Nash equilibria for the corresponding graphical game. This intuition allows to provide an analytical unifying framework for the local search techniques described above (DSA and MGM) as well as many refinements of those. Such unifying analytical framework was proposed by Chapman et al.[57].

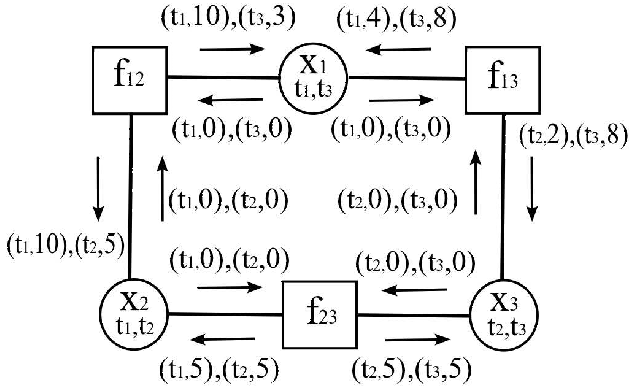

FIGURE 7: The messages corresponding to the first iteration of the max-sum algorithm for the factor graph corresponding to Figure 4.

Whereas DSA and MGM algorithms only consider moves involving the variable of one single agent (we refer to them as 1-size algorithms), solutions of higher quality can be obtained by considering local moves that involve larger groups of agents. For example, in Figure 4, consider now local moves involving pairs of agents. In this case, the algorithm will not be trapped in the local maximum $\mathbf{x}^1$ since agents a_2 and a_3 can make a local move that involves their variables at once and move to $x_1 = t_1, x_2 = t_1, x_3 = t_2$ with a local gain of 1. Along this direction, Vinyals and colleagues in [46] propose a generalization of the above described algorithms, the DALO algorithm, which allows agents to asynchronously execute local moves involving any agent's neighbourhood. When considering groups of two or more agents, these algorithms require coordination techniques to find a joint solution that is optimal for the whole group. For example, to find the joint local move with highest gain, agents a_2 and a_3 need to solve the following optimization problem: $\max_{\mathbf{x_2},\mathbf{x_3}} (f_{12}(x_1 = t_1, \mathbf{x_2}) + f_{23}(\mathbf{x_2}, \mathbf{x_3}) + f_{13}(x_1 = t_1, \mathbf{x_3}))$. This problem involves coordination as function f_{23} depends on both decision variables.

3.3.2. Sub-optimal Inference-based approaches

Sub-optimal inference-based approaches offer a very different perspective for solving DCOPs. In such approaches, each agent tries to compute an estimation of the impact that each of its action has on the global optimization function (usually called the $z_i(x_i)$ function). Such an estimate is build up by iteratively exchanging messages with its neighbours, and once the $z_i(x_i)$ function is built each agent chooses the assignment that maximizes this function.

The most prominent approach for sub-optimal inference-based methods in DCOPs is the max-sum approach. The operations of the max-sum algorithm are best understood on a factor graph representation

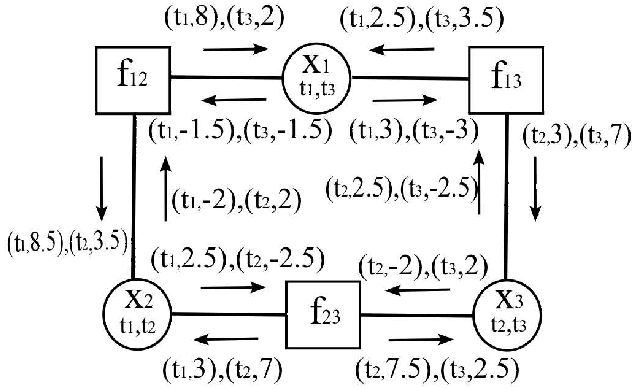

FIGURE 8: Reports the messages corresponding to the fifth iteration of max-sum for the factor graph corresponding to Figure 4. For this specific problem after five iterations max-sum converges to a fixed point for the messages and to the optimal allocation, this is not always the case in general cyclic graphs, however for graphs with a single loop there are strong guarantees on convergence and solution quality (see [58] for more details).

of the agents' interactions. In more detail, to compute the mentioned $z_i(x_i)$ function, the max-sum algorithm exchanges two kinds of messages in the factor graph: variable to function messages $q_{x_i \to f}(x_i)$ and function to variable messages $r_{f \to x_j}(x_j)$.

The messages flowing into and out of the variable nodes within the factor graph are functions that represent the total utility of the network for each of the possible value assignments of the variable that is sending/receiving the message. For example, consider the factor graph shown in Figure 7, here message $r_{f_{13} \to x_1}(x_1) = [(\neg t_3, 4), (t_3, 8)]$ represents the utility that the sub-graph connected to x_3 through f_{13} can achieve if x_1 takes value t_3 (i.e., 8) and any other value different from t_3 (i.e., 4). At any time during the propagation of these messages, agent a_i is able to determine which value it should adopt such that the sum over all the agents' utilities is maximised. This is done by locally calculating the function, $z_i(x_i)$, from the messages flowing into agent i's variable node:

$$z_i(x_i) = \sum_{g \in \mathcal{M}_i} r_{g \to x_i}(x_i) \tag{3}$$

and then finding $\arg\max_{x_i} z_i(x_i)$. Here $\mathcal{M}_i$ is the set of functions connected to x_i. Consider again the factor graph shown in Figure 7, we have that $z_3(x_3) = r_{f_{13} \to x_3}(x_3) + r_{f_{23} \to x_3}(x_3) = [(t_2, 2), (t_3, 8)] + [(t_2, 5), (t_3, 5)] = [(t_2, 7), (t_3, 13)]$, hence agent a_3 will decide to execute task t_3. Notice that with respect to our example, this is an incorrect decision (we know from Figure 4 that the optimal allocation requires a_3 to execute task t_2) and this is due to the fact that this is the first iteration of max-sum and hence variable must acquire more information to compute a correct evidence. Furthermore, notice that in general in a cyclic network the sub-graphs connected to the variable do depend on each other (i.e., here f_{13} and f_{23} are connected through another path that does not include x_3) hence the computation of $z_i(x_i)$ is in such cases only an approximation of the maximum utility that the system can achieve in the optimal configuration.

Algorithm 1 max-sum

```
1: Q ← ∅ {Initialize the set of received variable to function message}
2: R ← ∅ {Initialize the set of received function to variable message}
3: while termination condition is not met do
4:    for x_j ∈ N_i do
5:       r_{f→x_j}(x_j) = computeMessageToVariable(x_j, f_V, Q)
6:       SendMsg(r_{f→x_j j}(x_j),a_j) {a_j agent that controls variable x_j}
7:    end for
8:    for g ∈ M_i do
9:       q_{x_i→g}(x_i) = computeMessageToFunction(x_i, g, R)
10:      SendMsg(q_{x_i→g}(x_i),a_g) {a_g agent that controls function g}
11:   end for
12:   Q ← getMessagesFromFunctions()
13:   R ← getMessagesFromVariables()
14:   x_i^* = updateCurrentValue(x_i, R)
15: end while
```

The operations performed by each agent in the max-sum approach are reported in Algorithm 1. At each execution step, each agent computes and sends the variable to function ($\mathbf{Q}$) and function to variable ((R)) messages. Next, each agent updates the incoming $\mathbf{Q}$ and $\mathbf{R}$ messages and updates its current value by computing the variable assignment that maximises function $z_i(x_i)$.

When computing a message to a function, $q_{x_i \to f}(x_i)$, agent a_i (i.e., the agent responsible for variable x_i), sums up all the last messages received by linked functions except the one received from function f. This is simply a component wise sum of the values in the messages (similar to what we have seen for the computation of $z_i(x_i)$). In formulas, $q_{x_i \to f}(x_i) = \sum_{g \in \mathcal{M}_i, g \neq f} r_{g \to x_i}(x_i)$. In cyclic graphs, messages are normalized to prevent them increasing endlessly[11].

When computing a message to a variable, $r_{f_V \to x_j}(x_j)$, agent a_i projects out variable x_j from the function that results by summing up the function f_V and the last messages received by linked variables (except the one received from variable x_j)[12]. For example, consider the factor graph reported in Figure 8 which reports the value of messages computed by the algorithm

[11] This normalization can be performed by subtracting from each component of a message to function $q_{x_i \to f}(x_i)$ a value $\alpha = \frac{\sum_{d_i \in D_i} q_{x_i \to f}(d_i)}{|D_i|}$ so that $\sum_{d_i \in D_i} q_{x_i \to f}(d_i) = 0$.

[12] see Section 3.2 for the definition of projection and sum operators

in the fifth iteration[13]. Consider message $r_{f_{13}\to x_3}(x_3)$, the value of $r_{f_{13}\to x_3}(x_3 = t2)$ is given by $max([f_{13}(x_1 = t3, x_3 = t2) + q_{x_1\to f_{13}}(x_1 = t3)], [f_{13}(x_1 = t1, x_3 = t_2) + q_{x_1\to f_{13}}(x_1 = t1)]) = max(-1, 3)$. In formulas, agent a_i considers all the joint variable assignments $(d_j, d_{k_1}, \cdots, d_{k_s})$ of variables $V = (x_j, x_{k_1}, \cdots, x_{k_s})$, then for each value $d \in D_j$ it computes $r_{f_V\to x_j}(d) = \max_{d_{k_1},\cdots,d_{k_s}}[f_V(d, d_{k_1}, \cdots, d_{k_s}) + \sum_{k\in(k_1,\cdots,k_s)} q_{x_k\to f_V}(d_k)]$

The max-sum iterative message passing process terminates when the messages reach a fixed point (i.e., new messages are identical to previous messages). However, the max-sum algorithm is guaranteed to converge to a fixed point only if the factor graph is acyclic and in this case it computes the optimal assignment. In more general settings, there are only limited guarantees on convergence and solution quality [58], however extensive empirical results [45, 52] show that when executed on a loopy factor graph max-sum often achieves very good solutions. Since in these cases convergence is not guaranteed the iteration process is usually performed for an arbitrary (relatively small) number of coordination cycles.

Finally, notice that in general there can be multiple joint variable assignments that achieve the optimal solution. In such cases an extra coordination phase (usually called value propagation) is required to make sure agents chooses the same joint assignment [59]. Another approach that works well in practice is to break the symmetry of the problem by inserting small random preference for each variable over its domain values [45].

3.3.3. Approximate quality assessment

All the algorithms presented in this section are low-cost algorithms that can scale to large problems or return fast solution in dynamic environments at the cost of returning sub-optimal solutions. In particular, suboptimal coordination returns good solutions on average but can also converge to very poor solutions. Whereas in many applications optimality is not achievable, it does not mean that guarantees on solution quality are not important (e.g. some applications may need to guarantee that the algorithm does not converge to very poor solutions). Hence, in this section, we analyse frameworks that can provide such guarantees.

The most prominent approach to provide guarantees to suboptimal DCOP algorithms is the region optimality framework [46]. More concretely, a quality guarantee δ ensures that the value of a solution $\mathbf{x}$ is within a give distance δ from the optimal $\mathbf{x}^*$. Region optimality defines worst-case guarantees on the solution quality of any region optimal solution in any arbitrary region $\mathcal{C}$. We can regard a collection of neighborhoods $\mathcal{C}$ as an exploration region in a constraint graph. A solution is $\mathcal{C}$-optimal when its value cannot be improved by changing the decision of any group of agents, which we shall refer to as neighbourhoods, inside the region $\mathcal{C}$.

From this definition, the reader can foresee that the solutions to which algorithms introduced in sections above converge are region optimal, each on its own particular region. More specifically, MGM and DSA algorithms are optimal in a region that contains one neighbourhood per agent, composed of its single variable. We refer to these algorithms as 1-size region optimal (e.g. the region is characterised by including all neighborhoods of size 1). DALO is a generic region optimal algorithm that finds a region optimal for any arbitrary region $\mathcal{C}$. Finally, the solutions returned by Max-Sum on convergence are region optimal in the SLT-region – namely, the region that contains one neighborhood for any subset of variables whose induced subgraph in the constraint graph contains at most one cycle [60]. Figure 9 depicts the set of possible neighborhoods for the DCOP in Figure 4 where boldfaced nodes in the DCOP constraint graph stand for variables included in the neighborhood.

Region optimality provides two mechanisms to compute these quality guarantees that differ on their computational cost. The first mechanism directly searches the space of problems to find a problem where the quality of the region optimum with respect to the global optimum is minimized. The main drawback of this mechanism is that it requires to generate and solve a linear program (LP) with a number of constraints exponential to the number of variables in the largest neighborhood in the region. Here, we explain the second mechanism that allows to assess the quality guarantees in linear time at the cost of loosing bound tightness. Before describing this method, prior we need to define some relationships between the scope of the functions in the DCOP formulation and the region. In more detail, given a region $\mathcal{C}$, for each $f_V \in \mathcal{F}$ we define:

- $cc(f_V, \mathcal{C}) = |S \in \mathcal{C} \text{ s.t. } V \subseteq S|$, that is the number of neighborhoods in $\mathcal{C}$ that cover the domain of f_V completely. For example in Figure 9 the constraint between variable x_1 and x_2 (f_{12}) is totally covered by neighborhoods (d) and (g).
- $nc(f_V, \mathcal{C}) = |S \in \mathcal{C} \text{ s.t. } V \cap S = \emptyset|$, that is the number of neighborhoods in $\mathcal{C}$ that do not cover variables in the domain of f_V at all. For example in Figure 9 the constraint between variable x_1 and x_2 (f_{12}) is non-covered at all by neighborhood (c).

Given these definitions, the region optimality bound is defined as:

$$\delta = \frac{cc_*}{|\mathcal{C}| - nc_*} \tag{4}$$

where $cc_* = \min_{f\in\mathcal{F}} cc(f, \mathcal{F})$, $nc_* = \min_{f\in\mathcal{F}} cc(f, \mathcal{C})$ and, for any $\mathcal{C}$-optimal assignment, $\mathbf{x}^{\mathbf{C}}$ in any DCOP with non-negative utilities[14], the following equation

[13] To perform this computation we assume a synchronous execution model for the agents, i.e. each agent computes at the same time and then all messages are sent. However, max-sum can operate also in asynchronous settings [45].

[14] Every DCOP that does not have infinite negative costs can

holds:

$$F(\mathbf{x^C}) \geq \delta \cdot F(\mathbf{x}^*) \quad (5)$$

In what follows we give some intuition behind Equation 5 (we refer the interested reader to [46] for a detailed derivation of this bound). Take $|\mathcal{C}|$ assignments, X, inside region $\mathcal{C}$, one per neighbourhood in $\mathcal{C}$. Without loss of generality, we assume that the set of variables that vary in each $\mathbf{x} \in X$ with respect to $\mathbf{x^C}$ (corresponding to one neighborhood in the region) take the same value than in the optimal assignment $\mathbf{x}^*$. By its definition, $\mathbf{x^C}$ is guaranteed to have greater value than any assignment $\mathbf{x} \in X$ and consequently than their average value: $F(\mathbf{x^C}) \geq \frac{\sum_{\mathbf{x} \in X} F(\mathbf{x})}{|\mathcal{C}|}$. Then Equation 4 exploits that: cc_* is the number of times that all constraints take the value of the optimal in these assignments and nc_* is the number of times that all constraints take value of $\mathbf{x^C}$ to define the bound.

Equation 4 directly provides a simple algorithm to compute a bound δ. Given a region $\mathcal{C}$ and a constraint graph, for each constraint $f_V \in \mathcal{F}$ we can directly assess cc_* and nc_* by computing $cc(f_V, \mathcal{C})$ and $nc(f_V, \mathcal{C})$ and taking the minimum. Next we illustrate the computation of this bound for the DCOP in Figure 4 when using regions whose neighborhoods are characterized by its size (the so-called k-size regions):

- For the 1-size region ($\mathcal{C} = \{\{x_1\}, \{x_2\}, \{x_3\}\}$, corresponding to neighborhoods (a)(b)(c) in Figure 9) none of the constraints is totally covered by any neighbourhood, then $cc_* = 0$ and $\delta = 0$. This bound is the same for any 1-size region optimal algorithm (as are DSA and MGM reviewed in Section 4) meaning that region optimality can not provide any guarantees for regions containing neighbourhoods composed of single variables.
- For the 2-size region ($\mathcal{C} = \{\{x_1, x_2\}, \{x_1, x_3\}, \{x_2, x_3\}\}$, corresponding to neighborhoods (d)(e)(f) in Figure 9), each function is totally covered exactly once ($cc_* = 1$) and the is no neighbourhood that do not cover at all any function ($nc_* = 0$). Hence, $cc_* = 1$, $nc_* = 0$, $|\mathcal{C}| = 3$ and the bound in this case is $\delta = \frac{1}{3}$;
- The 3-size region contains a group composed of all variables ($\mathcal{C} = \{\{x_1, x_2, x_3\}$, corresponding to neighborhood (g) in Figure 9) and consistently the bound $\delta = 1$ guarantees optimality in this case.

As the reader may have noticed from Equation 4, the set of neighborhoods that compose a region is of central importance because they determine the degree of dominance of the region optima, hence directly affecting the definition of the corresponding quality guarantees. Along this line, researchers have explored different criteria, other than size, to characterize the neighborhoods to be included in a region. For example, the t-distance criterion, proposed in [61], includes one neighborhood per agent composed of all agents at distance at most t from this one in the constraint graph. For example, the 0-distance neighborhood centered of a_1 in the constraint graph in Figure 9 contains its variable (corresponding to neighborhood (a)) whereas its 1-distance neighborhood contains its own variable and the variable of its direct neighborhoods in the graph, that is x_2 and x_3 (corresponding to neighborhood (g)). Lastly, an alternative trade-off to size and distance, the so-called the s-size-bounded distance criterion, was proposed in [46] which includes for each agent the largest t-distance region centered on it whose number of variables does not exceed the limit s. For example, the 2-size-bounded distance neighborhood of agent a_1 in Figure 9 corresponds to the 0-distance neighborhood since as we have seen the 1-distance neighborhood contains three variables and hence, exceeds the maximum size threshold. The experimental results provided in [46] show how this more fine grained control of the complexity of the neighbourhoods provided by the size-bounded distance criterion leads to better solution qualities than when employing k-size or t-distance criteria.

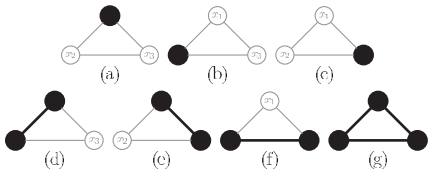

FIGURE 9: All possible neighborhoods for the DCOP in Figure 4. Boldfaced nodes stand for variables included in a neighborhood and boldfaced edges for the scope of functions totally covered by the neighborhood.

Finally, notice that the quality guarantees provided by the region optimality framework are very general, in the sense that are valid for any constraint graph with the considered agents' interactions (or with any subset of them), independently of the particular utility values returned by the constraints (what we shall refer to as reward structure). Thus, the bound of $\frac{1}{3}$ computed by the 2-size region of the DCOP in Figure 4 indeed holds for any DCOP with three decision variables since it is independent of the reward structure and considered the complete structure of interactions (any pair of agents is connected in the corresponding constraint graph). This, together with the property that these bounds can be assessed at design time (prior to the execution of the algorithm) makes these generic-instance bounds particularly interesting for dynamic environments or for problems for which little knowledge is available at design time. However, a much better bound can be obtained if we restrict our attention to

be normalized to one with all non-negative rewards. However the analysis is not directly applicable to DCOPs that include hard constraints.

specific constraint graph topologies or assuming some *a priori* knowledge on the reward structure [62, 63]. For example, Bowring et al. show that the k-size approximation bounds can be improved by knowing the ratio between the minimum reward to the maximum reward [62].

3.4. Challenges and Further Readings

In Sections 3.2 and 3.3 we provided a broad overview of DCOPs for Multi-Agent System coordination. Specifically, we described what a DCOP is and how a joint decision making problem can be modelled as a DCOP. We presented several solution approaches for DCOPs discussing three main categories: exact approaches, heuristics and approximate algorithms. Our overview clearly shows that there are many solution techniques for solving DCOPs and that they tackle several aspects of the coordination problem: (i) complete algorithms can be used when computational resources are available and it is crucial to achieve an optimal solution; (ii) suboptimal algorithms provide fast and light approaches for computing good solutions in practical applications with scarce resources (i.e., robotics or low power devices); and finally, (iii) approximate quality assessment over the solutions returned by these algorithms is particularly useful for large scale critical operations, when optimality is not achievable but pathological behaviours of the algorithms must be avoided (e.g., emergency management).

As mentioned in Section 3.3.1, there is an interesting and strong relationship between DCOPs and graphical games. In this perspective, it is worth noting that the important problem of computing Nash and correlated equilibria in graphical games has been recently addressed by using message passing techniques that show deep relationships with the inference-based approaches discussed here. In particular the work by Ortiz and Kearns [64] proposes a message passing approach for computing Nash equilibria and the work by Chapman et al. [65] proposes a message passing approach to compute pure strategy Nash equilibria that optimise standard criteria (such as the utilitarian social welfare function). In the interest of space we do not expand this theme here but we refer the interested reader to the above cited references for further readings.

While previous sections present a rich wealth of solution approaches for DCOPs, there are still many aspects which are crucial for several practical applications that are not properly addressed by current state of the art. In particular, a crucial challenge for future DCOP techniques is to deal with the uncertainty that comes from lack of knowledge or limited perception of the agents when operating in real world environments. In more detail, the DCOP formalization we detailed in Section 3 assumes full knowledge of the rewards for every combination of variable assignments. In practical applications this is often a strong assumption and agents typically have an uncertain estimate of the rewards that they can achieve given their actions or must learn these values when acting. Hence the uncertainty must be somehow captured into the DCOP model and dealt with in the solution technique. There are a number of approaches that tackle these issues, in particular, we refer the interest reader to [66] for approaches that model uncertainty of the reward and attempts to find optimal solution considering such uncertainty, and to [67, 68] for approaches that aim at learning unknown rewards of agents' joint actions.

Regarding quality assessment, we can be broadly divide most approaches in two main categories: *instance-generic* and *instance-specific* approaches. The former can provide guarantees on solution quality without running any algorithm on the specific problem instances, as in the described region optimality framework. In contrast, the latter can only provide quality guarantees for a solution after processing a specific problem instance. Instance-specific approaches are complementary to instance-generic ones, as they usually give accurate bounds but only for specific problem instances [54, 47]. In this respect, the Bounded Max-Sum (BMS) approach is a good representative for this kind of technique [54]. The main idea behind BMS is to remove cycles from the original constraint network by simply ignoring some of the dependencies between agents. It is then possible to optimally solve the resulting tree structured constraint network, whilst simultaneously computing the approximation ratio for the original problem instance.

Another crucial challenge for future DCOP techniques is to model and handle the dynamism of the environment. In fact, current DCOP models consider one shot problems where agents build a DCOP representation of the environment, negotiate over a joint solution and then execute. In most applications the environment changes whenever agents execute their actions. For example in applications involving physical agents (e.g., robotics or emergency management) when agents execute their action their positions will change and consequently the constraint graph that describes the agents' interactions will be different. While many DCOP techniques have been used in domains with these characteristics by simply repeating a new optimization problem at subsequent time steps (see for example [69]), there are relatively few approaches that explicitly take dynamism into account. We refer the interest reader to [70] and [71] for approaches that deal with unexpected changes in the constraint graph.

Finally, we have not discussed self-emergence and bio-inspired approaches since they typically involve indirect coordination, with no direct communication among agents. We refer the interested reader to [72, 73] for overviews of these specific areas.

4. HOW TO ASSIGN TASKS AND RESOURCES: MULTI-AGENT RESOURCE ALLOCATION

As established in [74]:

> Multiagent Resource Allocation (MARA) is the process of distributing a number of items amongst a number of agents.

MARA encompasses a wide variety of research topics: (i) preference representation languages; (ii) social welfare measures; (iii) computational complexity of determining feasible/optimal allocations; (iv) negotiation protocols; (v) efficient algorithm design; (vi) mechanism design; and (vii) implementation, simulation and experimentation of MARA systems. All these topics are thoroughly described and reviewed in the survey by Chevaleyre et al. [74]. Henceforth, we mainly focus on the usage of auctions to solve MARA problems, since auctions [75] have been widely and successfully employed in MAS to allocate tasks and resources. Nonetheless the reader should keep in mind that auctions are not the only means of solving MARA problems.

An auction is a protocol that allows agents to indicate their interests in one or more resources and that uses these indications of interest to determine both an allocation of resources and a set of payments by the agents [76]. The goal of an auction is to allocate the goods to those who value them most. Therefore auctions offer *allocation procedures* for MAS resource allocation.

There are several reasons why auctions are widely employed in MAS. First, they can use the market to dynamically assess the value of resources whose value is unknown. Second, auctions are flexible and can be used to efficiently allocate complex resources. Third, auctions can be easily automated: the use of simple rules reduces the complexity of negotiations, hence being ideal for computer implementation.

Computer science, and in particular artificial intelligence, has paid special attention to the notion of *markets-as-computation* [77], namely the use of a market-based method, such as an auction, to compute the outcome to a distributed problem. Along this line, auctions have been largely, and successfully, employed to solve a wealth of distributed task allocation (e.g. collaborative planning [78], formation of virtual organisations [79] , coordination for robot navigation [80], coordination in disaster management [81]), and distributed resource allocation problems, such as resource allocation in sensor networks [82]. All these market-based distributed systems share the commonality of being flexible and robust to rapidly adapt to changes and respond to failures (e.g. refer to [83]). Furthermore, there is the assumption that agents participating in auctions bid truthfully.

We take the same stance in this tutorial. We do not discuss the concern that bidders in an auction might not report their true valuations[15] because of strategic considerations. We focus on the optimisation problem faced by an auctioneer that must identify the winner(s) of an auction: given a set of bids, find an allocation of resources to bidders that maximizes the auctioneer's revenue. Neither do we discuss the issue that in some auctions the prices charged from winning bidders differ from the bids made, implying that the value of the optimal allocation is not equal to the revenue for the auctioneer. However, as argued in [84], the optimisation problems discussed below are key to designing auction mechanisms that *motivate* bidders to report their true valuations. In those mechanisms payments, namely prices charged to winning bidders, may differ from the bids made. We show a simple example supporting this observation in Section 4.1.

In what follows, we introduce a case study inspired by transportation procurement, a domain where auctions, and in particular combinatorial auctions, have been widely applied (e.g. [85]). We introduce several problems appearing in our case study and show how to solve them by means of auctions. This part of the tutorial ends with some recommendations on readings that may help the reader to delve further into auctions.

4.1. Case study: News distribution

Consider a company, *News Ltd.*, which is in charge of distributing newspapers to the newsagents of a zone. The company itself hauls newspapers to newsagents, where they are sold to customers. *News Ltd.* owns two depots, W_1 and W_2, as pictured in Figure 10. As newsagents request to be serviced, *News Ltd.* must decide whether to ship the newspapers from either depot W_1 or depot W_2. This decision will depend on the transportation costs from each depot and on the revenue generated by each requesting newsagent. Our purpose is to design an auction protocol that helps *News Ltd.* make such a decision. With this aim, in what follows we will gradually show several auction alternatives that will serve to illustrate different optimisation problems.

All the auctions introduced below will follow a common pattern divided into four sequential steps [16]:

1. *Bid call.* The auctioneer broadcasts a call for bids to serve some customer(s) and declares when the auction will close.
2. *Bid collection.* Depots transmit their bids to the auctioneer, and the auctioneer collects the bids. Depots bid trutfhully and each bid is computed by each depot as the cost of delivering newspapers to a set of customers.

[15] A bidder's valuation is a function that defines the bidder's preferences for the objects at auction.

[16] As shown in [77], in general we can think of all auctions as being instantiations of a general auction process, which is broken down into sub-processes. Thus, an auctioneer equipped with this model would be able to run any kind of auction by instantiating each process appropriately. The interested reader should refer to [77] for details about this abstract auction process model.

FIGURE 10: Single customer scenario

	Depots			
	W1		**W2**	
Cheapest route servicing A	12	✗	8	✓

TABLE 1: Single customer bids

3. *Winner determination.* Upon collection of all bids, the auctioneer determines, depending on the auction mechanism, who gets which customers (namely, an allocation).
4. *Clearing.* The auctioneer informs the depots about the resulting allocation.

Notice that since bidders only bid once, all auction protocols that we consider next are called *one-shot*.

4.1.1. Scenario 1: Handling a single customer

Consider the scenario depicted in Figure 10. A single customer, newsagent A, requests to be delivered newspapers. *News Ltd.* must decide whether to serve that request from either depot W_1 or depot W_2. The auction starts by issuing a bid call to serve customer A. After the bid call, the auctioneer collects two bids: W_1's bid, $BID(\langle W_1, A\rangle, 12)$, indicates that its cost to serve A is 12 ; W_2's bid, $BID(\{A\}, 8)$, indicates that its cost to serve A is 8. Each bid is a pair whose first component is a journey and its second component is the cost of the journey. For instance, $BID(\langle W_1, A\rangle, 12)$ is an offer to go from W_1 to A, and back from A to W_1. Thus, the cost considers that there is a cost to deliver to A (6) plus a cost to return to W_1 (6). The winner determination problem faced by the auctioneer is rather straightforward: to pick up the bid that minimises the cost of the service. Then, the auctioneer would clear the auction by informing W_1 and W_2 about the resulting allocation: W_2 will deliver newspapers to A. Thus, the cost of the allocation is 8.

In general, given a set of n bids, the winner determination problem (WDP) will consist in picking up the bid with minimum cost. Therefore, the computational complexity of the WDP is $O(n)$.

Consider now that customers are selfish and hence may not necessarily bid truthfully. In that case we could select the winner of the auction solving the same winner determination problem described above, but using a different clearing. Thus, we could employ a second-price auction, which allocates the resource to the winner, but has him pay the second highest price. Although the price that the winner would pay in this auction is different from the auction described above, both solve exactly the same winner determination problem. In general, this applies to all the Groves mechanisms, and in particular to the Vickrey-Clarke-Groves mechanism [76].

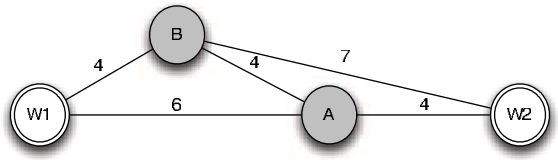

FIGURE 11: Multiple customer scenario

	Depot			
Cheapest route servicing	**W1**		**W2**	
A	12	✗	8	✓
B	8	✓	14	✗

TABLE 2: Parallel auctions bids

4.1.2. Scenario 2: Handling multiple customers with parallel auctions

Consider now a scenario where *News Ltd.* must serve multiple customers such as depicted in Figure 11. There, two customers, newsagents A and B, request to be delivered newspapers. In this case, *News Ltd.* decides to run two separate auctions in parallel, one per customer. After the two separate bid calls, the auctioneer of each of the auctions collects two bids as shown in Table 2. The auctioneer collecting bids to serve customer A receives the same bids as in the single-customer scenario. The auctioneer collecting bids to serve customer B receives two bids: W_1's bid, $BID(\langle W_1, B\rangle, 8)$, indicates that its cost to serve B is 8; W_2's bid, $BID(\langle W_2, B\rangle, 14)$, indicates that its cost to serve B is 14. Likewise in the above-described scenario, the winner determination problem faced by each auctioneer is rather straightforward: to pick up the bid that minimises the cost of the service. Therefore, the auction to serve customer A would allocate the service to W_2, whereas the auction to serve customer B would allocate the service to W_1. Thus, the total cost of the allocation is 16.

In this case, each auctioneer faces the very same optimisation problem than the auctioneer of the single-customer scenario. In general, if there are m resources and n agents, the computational complexity of parallel auctions as an allocation procedure is $O(m \cdot n)$.

	Routes							
	None		Only A		Only B		A & B	
W1	0	✗	12	✗	8	✗	14	✓
W2	0	✓	8	✗	14	✗	15	✗

TABLE 3: Combinatorial auction bids. Service to both A and B should be guaranteed.

4.1.3. Scenario 3: Handling multiple customers with combinatorial auction

At this point, the reader might have noticed that the cost of the allocation obtained in our latest scenario can be decreased had bidders been allowed to bid to service more than a single customer. Consider again Figure 11. Regarding W_1, its cost to service both A and B is 14 that we split into the following costs: from W_1 to A (6), from A to B (4), and from B back to W_1 (4). Regarding W_2, its cost to service both A and B is 15, after adding the cost from W_2 to A (4), from A to B (4), and from B back to W_2 (7). Therefore, allocating both delivery services to either W_1 or W_2 would be more advantageous to *News Ltd.* than the allocation obtained by the parallel auctions considered in the latest scenario.

In combinatorial auctions (CA) [86] different goods can be traded simultaneously and bidders can submit bids for subsets of the goods at trade.

Then *News Ltd.* sets up a CA where bidders are allowed to bid for bundles (subsets) of the services in $R = \{A, B\}$. Thus, *News Ltd.* would run a single auction this time. In the bid call, both W_1 and W_2 are requested to communicate the cost of servicing each possible subset of R: (1) to do no delivery at all; (2) to deliver only to A; (3) to deliver only to B; and (4) to deliver to both A and B. In this case, the winner determination problem faced by the auctioneer is a combinatorial optimization problem: choosing which bidder to award to each service so as to minimize the overall total cost.

In general, let R be a set of resources and A a set of agents. Let 2^R be the powerset (set of subsets) of R. For each agent $a \in A$ and each subset of resouces $R' \in 2^R$ we define: (1) $p_{a,R'}$, the cost for agent a to provide the set of resources R' ; and (2) $x_{a,R'}$, a binary decision variable that takes on value 1 if agent a is allocated the set of resources R' and 0 otherwise. Solving the WDP for a CA amounts to solving the following integer linear program:

$$\begin{aligned}
\text{minimize} \quad & \sum_{a \in A} \sum_{R' \in 2^R} p_{a,R'} \cdot x_{a,R'} & \\
\text{subject to} \quad & \sum_{R' \in 2^R} x_{a,R'} = 1 & \forall a \in A \\
& \sum_{a \in A} \sum_{R' \supseteq \{r\}} x_{a,R'} = 1 & \forall r \in R \\
& x_{a,R'} \in \{0,1\} & \forall a \in A, \forall R' \in 2^R
\end{aligned}$$

The first constraint guarantees that each agent is assigned a single subset of resources. The second constraint guarantees that each resource is assigned to one and only one agent.

The approach of submitting each possible cost to the auctioneer can be adopted when the number of resources at trade is small. However, as the number of resources at trade increases, it turns out infeasible that the agents communicate 2^R values to the auctioneer to provide their costs in full detail. Thus, it is necessary to define a language that allows for a succinct communication of the costs of each agent to the auctioneer. Such languages are known as *bidding languages*. The reader interested in learning more about bidding languages for combinatorial auctions is suggested to read [87]. To illustrate the concept of bidding language we provide an example, known as the OR language and the corresponding WDP.

Given a set of resources $R = \{r_1, \ldots, r_m\}$ and a set of agents A, the OR language allows each agent to submit as many bids as she wants for different subsets of goods. The auctioneer is free to select as many as she wants to be in the winning set. Thus, the bids of all agents are included into a set of bids $B = \{\langle R_1, p_1\rangle, \ldots, \langle R_n, p_n\rangle\}$, where $R_1, \ldots, R_n \subseteq 2^R$ and $p_1, \ldots, p_n$ are the bid prices.

We now show how to map the WDP for a CA using the OR language into integer programming (IP) [15]. Let x_i be a decision variable that takes on value 1 if bid $\langle R_i, p_i\rangle$ is chosen as a winning bid, and 0 otherwise. Now solving the WDP for a CA using the OR language amounts to solving the following integer program:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1} p_i \cdot x_i & \\
\text{subject to} \quad & \sum_{R_i \supseteq \{r\}} x_i = 1 & \forall r \in R \\
& x_i \in \{0,1\} & \forall i \in B
\end{aligned}$$

The (decision problem underlying the) WDP for standard CAs is known to be $\mathcal{NP}$-complete, with respect to the number of goods [86]. NP-hardness can, for instance, be shown by reduction from the well-known SET PACKING problem.

Back to the problem in our scenario, after running the integer program above, the auctioneer would choose the bid submitted by W_1 to serve both customers, $BID(\langle W_2, A, B\rangle, 14)$, as the winning bid and would discard all bids submitted by W_2. Thus, the total cost of the allocation is 14, which is less than the allocation cost obtained through parallel auctions. Here the auctioneer benefits from the fact that bidders have non-additive valuation functions. Indeed notice that the bid to service A plus the bid to service B separately is greater than the bid to service A and B altogether for both W_1 and W_2 (e.g. considering W_1, its offers to serve A and B separately –$BID(\langle W_1, A\rangle, 12)$, $BID(\langle W_1, B\rangle, 8)$ –add up to 20, while its offer for the bundle is 14 – $BID(\langle W_1, A, B\rangle, 14)$). In our example, we say that the services at trade are *complementary*. In general, CAs are interesting when the goods at trade are either complementary (e.g. a left shoe and a right shoe) or *substitutable* (e.g. two tickets to different movies that you like, playing at the same time).

CAs lead to more efficient allocations by avoiding the exposure problem inherent to sequential and parallel

auctions. This problem occurs when bidders fail to obtain complementary goods (e.g. imagine that a bidder manages to buy a left shoe but not the right shoe). Nonetheless, there are drawbacks that hinder their application. On the one hand, such as the computational complexity of the WDP and the size of the valuation function to communicate to the auctioneer (for each bidder, there is a bid per element in 2^R).

To cope with the first problem, there are two main alternatives: (1) require bids to come from a restricted set, guaranteeing that the WDP can be solved in polynomial time[17]; and (2) use heuristic methods to solve the problem (this works pretty well in practice, making it possible to solve WDPs with many hundreds of goods and thousands of bids). As argued before, bidding languages [87] can cope with the second problem, providing a more efficient way of communicating a value function.

4.1.4. Scenario 4: Including valuations and costs via a combinatorial exchange

The scenarios analysed so far have not taken into account that customers may be differently valued by *News Ltd.*. For instance, consider again the scenario depicted in Figure 10 along with the bids issued by W_1 and W_2, namely $BID(\langle W_1, A\rangle, 12)$ and $BID(\langle W_2, A\rangle, 8)$. Moreover, say that the auctioneer knows also the value of customer A (e.g. in terms of the sale price to be charged to the customer), and hence she intends to take into account this value to determine the winner of the auction. First, say that the valuation of customer A is 3. In that case, the auctioneer would allocate the delivery to A neither to W_1 nor to W_2 because that would bring a loss (at least -5). However, if the valuation of customer A were 15, then the auctioneer would allocate the service to W_2 to generate a revenue of 7 $(15 - 8)$.

Going one step further, we consider again the scenario involving multiple customers depicted in Figure 12, but now we count on the valuations of customers as follows: $v(\emptyset) = 0$, $v(\{A\}) = 3$, $v(\{B\}) = 10$ and $v(\{A, B\}) = 13$. Again, the winner determination problem faced by the auctioneer is a combinatorial optimization problem: choosing which services to award to each bidder so as to maximise the benefit (as the difference between customer valuation and service costs). We can regard customers' valuations as offers to buy delivery services and depots' bids as offers to sell delivery services. Hence, in fact the auctioneer is dealing with a *combinatorial exchange* (CE) [89] because it involves combinatorial offers involving multiple buyers (the customers) and multiple sellers (the delivery service providers). A combinatorial exchange is a generalisation of a CA where participanting agents are allowed to both buy and sell (bundles of) resources, or just buy or just sell.

[17] The problem is that these restricted sets must be very restricted (see [88]).

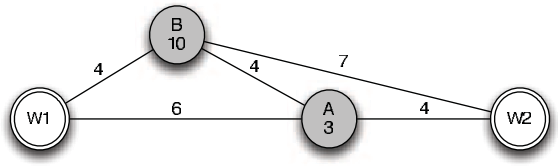

FIGURE 12: Combinatorial exchange scenario

In general, let R be a set of m resources and A a set of agents. As in section 4.1.3, the bids of all agents are included into a set of bids $B = \{\langle R_1, p_1\rangle, \ldots, \langle R_n, p_n\rangle\}$, where $R_1, \ldots, R_n \subseteq 2^R$ are sets of resources and $p_1, \ldots, p_n$ are the prices requested to provide each of those sets.

Similarly, valuations are included into a set of valuations $V = \{\langle Q_1, v_1\rangle, \ldots, \langle Q_m, v_m\rangle\}$, such that each valuation $V_l \in V$ takes the same form as a bid. However, unlike bids, given a valuation $\langle Q_l, v_l\rangle$, Q_l is the set of resources requested and v_l indicates how much the buyer is willing to pay for getting Q_l (the valuation price). Now we define two types of decision variables: x_i, to decide whether bid B_i is taken or not; and y_l, to decide whether valuation V_l is taken or not. Then, solving the WDP for a CE in the OR language amounts to solving the following integer linear program:

$$\text{maximize} \sum_{l=1}^{|V|} v_l \cdot y_l - \sum_{i=1}^{|B|} p_i \cdot x_i$$

$$\text{subject to} \sum_{Q_l \supseteq \{r\}} y_l = \sum_{R_i \supseteq \{r\}} x_i \qquad \forall r \in R$$

$$x_i, y_l \in \{0, 1\} \qquad \forall B_i \in B \;\;,\;\; \forall V_l \in V$$

The function to maximize computes the benefit of an allocation. The first constraint ensures that the demand and supply of resources are balanced.

The interested reader may refer to [89] for a more general, compressed formulation of the WDP for CE that allows the trading of multiple units of resources.

Back to our example in Figure 12, an auctioneer solving the WDP of the CE by means of the linear program above would allocate W_1 to service customer B, but would leave customer A without service. The reason is that the valuation for A is too low. Given the bids and valuations in this scenario, allocating a delivery service to A would lead to an allocation with negative value: -1, which results from adding the valuations for A and B and subtracting the combinatorial bid issued by W_1. Instead, allocating a service only to B has value 2 (10, the value for B, minus 8, corresponding to the single bid for B sent by W_1). Notice that this allocation differs from the one formerly obtained by a CA when disregarding valuations.

	Routes							
	None		Only A		Only B		A & B	
W1	0	✗	12	✗	8	✓	14	✗
W2	0	✓	8	✗	14	✗	15	✗
Value	0	✗	3	✗	10	✓	13	✗

TABLE 4: Combinatorial exchange bids. The best option is not to service A.

4.2. Discussion

Notice that from a designer's point of view, choosing (or defining) a market price system (e.g. an auction) and defining the bidding functions for the agents involved in the computational market become central issues. The methodology described in [90] provides valuable guidelines as to how to tackle this endeavour.

As discussed in [91] (see Chapter 3), the motivation for market-based approaches has been to produce efficient solutions to distributed problems. Hence computational aspects were investigated before game-theoretical properties of market mechanisms. Nonetheless, more recently there is an ongoing effort to integrate game-theoretic concerns and computational concerns in the emerging field of computational mechanism design (CMD) (refer to [91, 92, 93] for excellent introductions to the field). The challenge of CMD is to make mechanisms computationally feasible while maintaining useful game-theoretical properties such as efficiency (allocating the goods to those who value them most) and strategy-proofness (enforcing agents to report truthfully). Unlike the auction approaches studied in this section, CMD does not assume that the agents composing a MAS necessarily cooperate to find a good system-wide solution. Instead, CMD assumes that the designer might not be able to enforce bidding strategies on the market participants.

4.3. Further readings and advanced concepts

Sections 4.1 and 4.2 have illustrated concepts and simple techniques that arise to allocate resources among different agents. In this section we provide references that could prove valuable for the interested reader and quickly review some additional techniques and lines of work.

4.3.1. Further readings

Multi-agent resource allocation can be enclosed into the broader field of computational social choice. [94] provides an introductory overview of the main ideas and problems of this field that lies in the intersection of computer science, artificial intelligence and social choice. [74] provides a review of the MARA field wider than the short overview presented in this section.

Parsons et al. [77] survey the auction literature, primarily from the viewpoint of computer scientists who are interested in learning about auction theory. Furthermore, they also provide pointers into the economics literature for those who want a deeper technical understanding. Additionally to that survey, we also recommend the book on combinatorial auctions by Cramton et al. [95] for a wide, exhaustive perspective of the field.

In this section, we have mostly concentrated on auctions as a coordination tool in cooperative multi-agent systems. Latest advances on this area are reported in [96]. Koenig states that the standard testbed of auction based coordination systems is multi-robot routing [97]. A seminal paper on auctions for multi-robot routing is [98].

Mechanism design is a large area of research in economics and computer science. For detailed technical discussions, the interested reader is referred to the excellent introductions in [91, Chapter 2], [99], [100] and the most recent survey on mechanism design for computer scientists in [101]. As to computational mechanism design, a good starting point is [91, Chapter 3], though we definitely recommend the reader to dive into [76, Chapter 10] for further discussions into implementation issues as well as for a description of several computational applications of mechanism design.

4.3.2. Advanced optimisation problems

This tutorial is only intended to help the reader learn the foundations of some of the problems that multi-agent resource allocation poses. The reader might be interested in exploring further optimisation problems defined by other resource allocation mechanisms in the literature. Below we provide a selection of articles that by no means intends to be exhaustive, but only representative of general problems:

- *Supply chain formation.* Supply chain formation is a highly relevant problem in which auctions are the method of choice. Quoting [102] "A *supply chain* is a network of production and exchange relationships that spans multiple levels of production. Whenever we have a producer that buys inputs and sells outputs, we have a supply chain. Supply chain formation is the process of determining the participants in the supply chain, who will exchange what with whom, and the terms of the exchanges." The problem is clearly of utmost importance for today's networked economy. Mixed multi-unit combinatorial auctions (MMUCAs) [103] generalize several types of auctions and are a very convenient tool for automating supply chain formation. However, the winner determination problem is solved by a single centralized auctioneer and requires extensive computational resources. Diminishing the negative effect of these two issues is subject of current research. Thus, approximate distributed solvers, based in parallel auctions [102] or max-sum [104, 105], have been proposed

for certain types of MMUCAs. Despite their expressiveness, one of the main drawbacks of MMUCAs is that they are not able to manage time information, although there is ongoing work to extend the bidding language and WDP in that direction [106]. Another relevant issue in supply chain formation is dealing with eventual failure of agents to fulfill their commitments. Dealing with failures can be done by either taking into account the possibilities of failure during the supply chain formation process [107, 108] or by providing the agents with ways to repair the supply chain whenever failure happens [109].

- *Trust-based resource allocation.* Ramchurn et al. [110] extend combinatorial exchanges to include the trust and reputation of agents in a task allocation setting. Furthermore, the authors design a mechanism to cope with self-interested agents that is aimed at preventing agents' failures when performing tasks.
- *Robust resource allocation.* The work in [108, 111] introduces robust auctions, which can take into account the cost of repairing failures occurring in an allocation. Thus, the WDP is designed to take into account such costs to compute a robust enough allocation at the price of losing some revenue.

5. SUMMARY

In this section we summarise the key messages from the tutorial and provide the reader with an overview of the different approaches she could use to solve multi-agent optimisation problems. In more detail, in this paper we presented multi-agent optimisation techniques that can be used to form collectives of agents, coordinate their actions, and allocate resources among themselves. It is clear that there are overlaps and differences in the types of problems these techniques aim to solve. For example:

1. Assembling a collective of agents and assigning actions to each agent in a decentralised manner are two closely related problems in that in both cases, the optimal grouping of agents to perform some tasks is the main objective. However, the research agendas for these two problems target different issues. Thus, while research on coalition formation focuses on the *combinatorics of the formation of groupings*, the distributed constraints optimisation algorithms essentially focus on the *algorithmics of the communication protocols* needed to reach an optimal assignment of tasks or resources to agents. In some cases, it may even be possible to use the combinatoric techniques developed in coalition formation within DCOP algorithms to speed up the computation [14].
2. Allocating resources to agents (whether combinatorial or not) could either be carried out using auctions or DCOPs. While auctions are deemed appropriate for competitive settings where each agent is vying for some resource and is unlikely to reveal its private information to gain an advantage, DCOPs are more appropriate when the agents aim at maximising a system wide objective and might be willing to exchange messages containing some of their private information to do so. Nonetheless, privacy is considered as a crucial issue in DCOP solution techniques and there are approaches that analyse the privacy-efficiency tradeoff (e.g.,[112]) or propose the use of DCOPs for issues related to mechanism design, such as devising a faithful distributed implementation for efficient social choice (e.g., [113]). These approaches lead toward interesting research directions where DCOPs could be used as efficient tools for resource allocation problems that are typically solved by using auctions.

Now, given a particular problem, a practitioner would also need to decide what are the trade-offs that apply to a given technique based on the computational costs that this may involve. Thus, the need for decentralised decision making should be balanced against the cost of communication that may incur while the need for optimality may need to be balanced against the computation time allowed in the chosen scenario. While the final decision will largely depend on the context, our hope is that this survey can act not only as a recipe of solutions for real-world problems but also help those new to these research areas identify how their work correlates with the whole multi-agent optimisation research space.

6. FUNDING

This work was carried out as part of the following projects: ORCHID (funded by EPSRC EP/I011587/1), TIN2009-13591-C02-02, Generalitat de Catalunya 2009-SGR-1434, COR (TIN2012-38876-C02-01), AT (CONSOLIDER CSD2007-0022), MECER (201250E053).

7. ACKNOWLEDGEMENTS

We would like to thank Meritxell Vinyals for her extensive contribution on section 3.3. All the credit for that section should be given to her, and any remaining mistakes are the sole responsibility of the authors.

REFERENCES

[1] Lu, T. and Boutilier, C. (2012) Matching models for preference-sensitive group purchasing. *Proceedings of the 13th ACM Conference on Electronic Commerce*, pp. 723–740. ACM.

[2] Vinyals, M., Bistaffa, F., Farinelli, A., and Rogers, A. (2012) Coalitional energy purchasing in the smart grid. *Proceedings of the IEEE International Energy Conference & Exhibition (ENERGYCON 2012)*, pp. 848–853.

[3] Chalkiadakis, G., Robu, V., Kota, R., Rogers, A., and Jennings, N. (2011) Cooperatives of distributed

energy resources for efficient virtual power plants. *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems*, pp. 787–794. IFAAMAS.

[4] Shehory, O. and Kraus, S. (1998) Methods for task allocation via agent coalition formation. *Artificial Intelligence*, **101**, 165–200.

[5] Shehory, O. and Kraus, S. (1995) Task allocation via coalition formation among autonomous agents. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pp. 655–661.

[6] Sandholm, T. W., Larson, K., Andersson, M., Shehory, O., and Tohme, F. (1999) Coalition structure generation with worst case guarantees. *Artificial Intelligence*, **111**, 209–238.

[7] Chalkiadakis, G., Elkind, E., and Wooldridge, M. (2011) Computational aspects of cooperative game theory. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, **5**, 1–168.

[8] Chalkiadakis, G., Markakis, E., and Boutilier, C. (2007) Coalition formation under uncertainty: Bargaining equilibria and the Bayesian core stability concept. *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pp. 400–407. IFAAMAS.

[9] Rahwan, T. and Jennings, N. R. (2007) An algorithm for distributing coalitional value calculations among cooperating agents. *Artificial Intelligence*, **171**, 535–567.

[10] Knuth, D. E. (2004) Generating all n-tuples. Addison-Wesley.

[11] Voice, T., Ramchurn, S., and Jennings, N. (2012) On coalition formation with sparse synergies. *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, pp. 223–230. IFAAMAS.

[12] Dang, V. and Jennings, N. (2004) Generating coalition structures with finite bound from the optimal guarantees. *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 564–571. IEEE Computer Society.

[13] Dang, V. D., Dash, R. K., Rogers, A., and Jennings, N. R. (2006) Overlapping coalition formation for efficient data fusion in multi-sensor networks. *Proceedings of 21st National Conference on Artificial Intelligence (AAAI-06)*, pp. 635–640. AAAI Press.

[14] Ramchurn, S. D., Polukarov, M., Farinelli, A., Jennings, N., and Trong, C. (2010) Coalition formation with spatial and temporal constraints. *Proceedings of the 9th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2010)*, pp. 1181–1188. IFAAMAS.

[15] Hillier, F. S. and Lieberman, G. J. (2009) *Introduction to Operations Research*. McGraw-Hill.

[16] Rahwan, T., Michalak, T., Jennings, N., Wooldridge, M., and McBurney, P. (2009) Coalition structure generation in multi-agent systems with positive and negative externalities. *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pp. 257–263. Morgan Kaufmann.

[17] Elkind, E., Chalkiadakis, G., and Jennings, N. (2008) Coalition structures in weighted voting games. *Proceedings of the 18th European Conference on Artificial Intelligence* Frontiers in Artificial Intelligence and Applications, pp. 393–397. IOS Press.

[18] Rahwan, T., Ramchurn, S., Jennings, N. R., and Giovannucci, A. (2009) An anytime algorithm for optimal coalition structure generation. *Journal of Artificial Intelligence Research*, **34**, 521–567.

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

[20] Rahwan, T. and Jennings, N. R. (2008) An improved dynamic programming algorithm for coalition structure generation. *Proceedings of the 7th International Conference on Autonomous Agents and Multi-Agent Systems*, pp. 1417–1420. IFAAMAS.

[21] Sandholm, T., Larson, K., Andersson, M., Shehory, O., and Tohmé, F. (1998) Anytime coalition structure generation with worst case guarantees. *Proceedings of the 15th National Conference on Artificial Intelligence*, pp. 46–53. AAAI Press / The MIT Press.

[22] Rahwan, T. and Jennings, N. R. (2008) Coalition structure generation: Dynamic programming meets anytime optimisation. *Proceedings of the 23th AAAI Conference on Artificial Intelligence*, pp. 156–161. AAAI Press.

[23] Michalak, T. P., Sroka, J., Rahwan, T., Wooldridge, M., McBurney, P., and Jennings, N. R. (2010) A distributed algorithm for anytime coalition structure generation. *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, pp. 1007–1014. IFAAMAS.

[24] Ieong, S. and Shoham, Y. (2005) Marginal contribution nets: a compact representation scheme for coalitional games. *Proceedings of the 6th ACM conference on Electronic commerce*, pp. 193–202. ACM.

[25] Ohta, N., Iwasaki, A., Yokoo, M., Maruono, K., Conitzer, V., and Sandholm, T. (2006) A compact representation scheme for coalitional games in open anonymous environments. *Proceedings of 21st National Conference on Artificial Intelligence*, pp. 697–702. AAAI Press.

[26] Demange, G. and Wooders, M. (2005) *Group Formation in Economics: Networks, Clubs, and Coalitions*. Cambridge University Press.

[27] Demange, G. (2004) On group stability in hierarchies and networks. *Journal of Political Economy*, **112**, 754–778.

[28] Greco, G., Malizia, E., Palopoli, L., and Scarcello, F. (2011) On the complexity of core, kernel, and bargaining set. *Artificial Intelligence*, **175**, 1877–1910.

[29] Chalkiadakis, G., Markakis, E., and Jennings, N. R. (2012) Coalitional stability in structured environments. *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, pp. 779–786. IFAAMAS.

[30] Matthews, T., Ramchurn, S., and Chalkiadakis, G. (2012) Competing with humans at fantasy football: team formation in large partially-observable domains. *Proceedings of the 26th Conference on Artificial Intelligence*, pp. 1394–1400. AAAI press.

[31] Weiss, G. (2013) *Multi-Agent Systems*. MIT Press.

[32] Rahwan, T. (2007) Algorithms for Coalition Formation in Multi-Agent Systems. PhD thesis University of Southampton.

[33] Ueda, S., Iwasaki, A., Yokoo, M., Silaghi, M.-C., Hirayama, K., and Matsui, T. (2010) Coalition structure generation based on distributed constraint optimization. *Proceedings of the 24th AAAI Conference on Artificial Intelligence*. AAAI Press.

[34] Nair, R., Tambe, M., Yokoo, M., Pynadath, D. V., and Marsella, S. (2003) Taming Decentralized POMDPs: Towards Efficient Policy Computation for Multiagent Settings. *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pp. 705–711. Morgan Kaufmann.

[35] Bernstein, D. S., Zilberstein, S., and Immerman, N. (2000) The complexity of decentralized control of markov decision processes. *Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence*, pp. 32–37. Morgan Kaufmann.

[36] Srivastava, S., Immerman, N., and Zilberstein, S. (2011) A new representation and associated algorithms for generalized planning. *Artificial Intelligence*, **175**, 615–647.

[37] Kumar, A., Zilberstein, S., and Toussaint, M. (2011) Scalable multiagent planning using probabilistic inference. *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pp. 2140–2146. IJCAI/AAAI.

[38] Dechter, R. (2003) *Constraint Processing*. Morgan Kaufmann.

[39] Farinelli, A., Vinyals, M., Rogers, A., and Jennings, N. (2013) Distributed search and constraint handling. In Weiss, G. (ed.), *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press.

[40] Faltings, B. and Yokoo, M. (2005) Introduction: Special issue on distributed constraint satisfaction. *Artificial Intelligence*, **161**, 1–5.

[41] Modi, P. J., Shen, W.-M., Tambe, M., and Yokoo, M. (2005) Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, **161**, 149–180.

[42] Petcu, A. and Faltings, B. (2005) A scalable method for multiagent constraint optimization. *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pp. 266–271. Professional Book Center.

[43] Fitzpatrick, S. and Meertens, L. (2003) Distributed Sensor Networks: A multiagent perspective. Kluwer Academic.

[44] Maheswaran, R. T., Pearce, J. P., and Tambe, M. (2004) Distributed Algorithms for DCOP: A Graphical-Game-Based Approach. *Proceedings of the ISCA 17th International Conference on Parallel and Distributed Computing Systems*, pp. 432–439. ISCA.

[45] Farinelli, A., Rogers, A., Petcu, A., and Jennings, N. R. (2008) Decentralised coordination of low-power embedded devices using the max-sum algorithm. *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems*, pp. 639–646. IFAAMAS.

[46] Vinyals, M., Shieh, E. A., Cerquides, J., Rodriguez-Aguilar, J. A., Yin, Z., Tambe, M., and Bowring, E. (2011) Quality guarantees for region optimal DCOP algorithms. *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems*, pp. 133–140. IFAAMAS.

[47] Vinyals, M., Pujol, M., Rodriguez-Aguilar, J. A., and Cerquides, J. (2010) Divide-and-coordinate: DCOPs by agreement. *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, pp. 149–156. IFAAMAS.

[48] Yokoo, M., Durfee, E., Ishida, T., and Kuwabara, K. (1998) The distributed constraint satisfaction problem: formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, **10**, 673–685.

[49] Yeoh, W., Felner, A., and Koenig, S. (2010) BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. *Journal of Artificial Intelligence Research*, **38**, 85–133.

[50] Baker, A. (1994) The hazards of fancy backtraking. *Proceedings of the 12th National Conference on Artificial Intelligence*, pp. 288–293. AAAI Press / The MIT Press.

[51] Petcu, A. (2007) A Class of Algorithms for Distributed Constraint Optimization. PhD thesis EPFL Lausanne.

[52] Kok, R. J. and Vlassis, N. (2005) Using the max-plus algorithm for multiagent decision making in coordination graphs. *Proceedings of the Seventeenth Belgium-Netherlands Conference on Artificial Intelligence*, pp. 359–360. Koninklijke Vlaamse Academie van Belie voor Wetenschappen en Kunsten.

[53] Pearce, J. P. and Tambe, M. (2007) Quality Guarantees on k-Optimal Solutions for Distributed Constraint Optimization Problems. *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pp. 1446–1451. IFAAMAS.

[54] Rogers, A., Farinelli, A., Stranders, R., and Jennings, N. R. (2011) Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence*, **175**, 730–759.

[55] Hoos, H. H. and Stützle, T. (2004) *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann.

[56] Kearns, M. J., Littman, M. L., and Singh, S. P. (2001) Graphical models for game theory. *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, pp. 253–260. Morgan Kaufmann.

[57] Chapman, A. C., Rogers, A., Jennings, N. R., and Leslie, D. S. (2011) A unifying framework for iterative approximate best-response algorithms for distributed constraint optimization problems. *Knowledge Engineering Review*, **26**, 411–444.

[58] Weiss, Y. and Freeman, W. T. (2001) On the optimality of solutions of the max-product belief propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, **47**, 723–735.

[59] Zivan, R. and Peled, H. (2012) Max/min-sum distributed constraint optimization through value propagation on an alternating DAG. *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, pp. 265–272. IFAAMAS.

[60] Vinyals, M., Cerquides, J., Farinelli, A., and Rodriguez-Aguilar, J. A. (2010) Worst-case bounds on the quality of max-product fixed-points. *In Proceedings of the*

Neural Information Processing Systems (NIPS), pp. 2325–2333. MIT press.

[61] Kiekintveld, C., Yin, Z., Kumar, A., and Tambe, M. (2010) Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. *Proceedings of 9th International Conference on Autonomous Agents and Multiagent Systems*, pp. 133–140. IFAAMAS.

[62] Bowring, E., Pearce, J., Portway, C., Jain, M., and Tambe, M. (2008) On k-optimal distributed constraint optimization algorithms: new bounds and algorithms. *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent systems*. IFAAMAS.

[63] Vinyals, M., Shieh, E., Cerquides, J., Rodriguez-Aguilar, J. A., Yin, Z., Tambe, M., and E., B. (2011) Reward-based region optimal quality guarantees. *In Proceedings of the 4th International Workshop on Optimization in Multi-Agent Systems*.

[64] Ortiz, L. E. and Kearns, M. J. (2002) Nash propagation for loopy graphical games. *Proceedings of the Neural Information Processing Systems (NIPS)*, pp. 793–800. MIT Press.

[65] Chapman, A. C., Farinelli, A., de Cote, E. M., Rogers, A., and Jennings, N. R. (2010) A distributed algorithm for optimising over pure strategy Nash equilibria. *Proceedings of the 24th AAAI Conference on Artificial Intelligence*. AAAI Press.

[66] Léauté, T. and Faltings, B. (2011) Distributed constraint optimization under stochastic uncertainty. *Proceedings of the 25th Conference on Artificial Intelligence*, pp. 68–73. AAAI Press.

[67] Taylor, M. E., Jain, M., Tandon, P., Yokoo, M., and Tambe, M. (2011) Distributed on-line multi-agent optimization under uncertainty: Balancing exploration and exploitation. *Advances in Complex Systems*, **14**, 471–528.

[68] Stranders, R., Tran-Thanh, L., Fave, F. M. D., Rogers, A., and Jennings, N. R. (2012) DCOPs and bandits: exploration and exploitation in decentralised coordination. *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, pp. 289–296. IFAAMAS.

[69] Lesser, V., Ortiz, C. L., and Tambe, M. (eds.) (2003) *Distributed Sensor Networks A multiagent perspective*. Kluwer Academic.

[70] Petcu, A. and Faltings, B. (2005) S-DPOP: Superstabilizing, fault-containing multiagent combinatorial optimization. *Proceedings of the 20th National Conference on Artificial Intelligence*, pp. 449–454. AAAI Press.

[71] Lass, R., Sultanik, E., Greenstadt, R., and Regli, W. (2009) Robust distributed constraint reasoning. *Proceedings of the workshop of on Distributed Constraint Reasoning*, pp. 75–86.

[72] Serugendo, G. D. M., Gleizes, M. P., and Karageorgos, A. (2006) Self-organisation and emergence in mas: An overview. *Informatica (Slovenia)*, **30**, 45–54.

[73] Mano, J.-P., Bourjot, C., Lopardo, G. A., and Glize, P. (2006) Bio-inspired mechanisms for artificial self-organised systems. *Informatica (Slovenia)*, **30**, 55–62.

[74] Chevaleyre, Y., Dunne, P. E., Endriss, U., Lang, J., Lemaître, M., Maudet, N., Padget, J., Phelps, S., Rodríguez-Aguilar, J. A., and Sousa, P. (2006) Issues in Multiagent Resource Allocation. *Informatica*, **30**, 3–31.

[75] Klemperer, P. (2004) *Auctions: Theory and Practice*. Princeton University Press.

[76] Shoham, Y. and Leyton-Brown, K. (2009) *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press.

[77] Parsons, S., Rodriguez-Aguilar, J. A., and Klein, M. (2011) Auctions and bidding: a guide for computer scientists. *ACM Computing Surveys*, **43**, 10.

[78] Hunsberger, L. and Grosz, B. (2000) A combinatorial auction for collaborative planning. In Durfee, E., Kraus, S., Nakashima, H., and Tambe, M. (eds.), *Proceedings of the 4th International Conference on Multiagent Systems*, pp. 151–158. IEEE Computer Society.

[79] Patel, J., Teacy, W. T. L., Jennings, N. R., Luck, M., Chalmers, S., Oren, N., Norman, T. J., Preece, A. D., Gray, P. M. D., Shercliff, G., Stockreisser, P. J., Shao, J., Gray, W. A., Fiddian, N. J., and Thompson, S. (2005) Agent-based virtual organisations for the grid. *Multiagent and Grid Systems*, **1**, 237–249.

[80] Sierra, C., de Mantaras, R. L., and Busquets, D. (2000) Multiagent bidding mechanisms for robot qualitative navigation. *Intelligent Agents VII*, Lecture Notes in Artificial Intelligence, **1986**, pp. 198–212.

[81] Ramchurn, S. D., Rogers, A., MacArthur, K., Farinelli, A., Vytelingum, P., Vetsikas, I., and Jennings, N. R. (2008) Agent-based coordination technologies in disaster management. *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems*, pp. 1651–1652. IFAAMAS.

[82] Ostwald, J., Lesser, V., and Abdallah, S. (2005) Combinatorial auction for resource allocation in a distributed sensor network. *RTSS '05: Proceedings of the 26th IEEE International Real-Time Systems Symposium*, pp. 266–274. IEEE Computer Society.

[83] Jennings, N. R. and Bussmann, S. (2003) Agent-based control systems. *IEEE Control Systems Magazine*, **23**, 61–74.

[84] Lehmann, D., Müllller, R., and Sandholm, T. W. (2006) The Winner Determination Problem. *Combinatorial Auctions*, chapter 12, pp. 297–317. MIT Press.

[85] Ma, Z. (2008) Combinatorial auctions for truckload transportation procurement. PhD thesis University of Toronto.

[86] Rothkopf, M. H., Pekec, A., and Harstad, R. M. (1998) Computationally manageable combinational auctions. *Management Science*, **44**, 1131–1147.

[87] Nisan, N. (2006) Bidding Languages for Combinatorial Auctions. *Combinatorial Auctions*, chapter 9, pp. 215–231. MIT Press.

[88] Müller, R. (2006) Tractable Cases of The Winner Determination Problem. *Combinatorial Auctions*, chapter 13, pp. 319–336. MIT Press.

[89] Sandholm, T. W., Suri, S., Gilpin, A., and Levine, D. (2002) Winner determination in combinatorial auction generalizations. *Proceedings of The first International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 69–76. ACM.

[90] Bussmann, S., Jennings, N., and Wooldridge, M. (2004) *Multiagent systems for manufacturing control: A design methodology* Series on Agent Technology. Springer-Verlag, Berlin, Germany.

[91] Parkes, D. (2001) Iterative Combinatorial Auctions: Achieving Economic and Computational Efficiency. PhD thesis University of Pennsylvania Department of Computer and Information Science.

[92] Dash, R. K., Parkes, D. C., and Jennings, N. R. (2003) Computational mechanism design: A call to arms. *IEEE Intelligent Systems*, **18**, 40–47.

[93] Conitzer, V. (2008). Mechanism design for MAS. Course at the Dubai Agents and Multiagent Systems School.

[94] Chevaleyre, Y., Endriss, U., Lang, J., and Maudet, N. (2007) A Short Introduction to Computational Social Choice. *SOFSEM 2007 Theory and Practice of Computer Science*, Lecture Notes in Computer Science, **4362**, pp. 51–69. Springer.

[95] Cramton, P., Shoham, Y., and Steinberg, R. (eds.) (2006) *Combinatorial Auctions*. The MIT Press, Cambridge, MA.

[96] Koenig, S., Keskinocak, P., and Tovey, C. (2010) Progress on Agent Coordination with Cooperative Auctions. *Proceeding of the 24th AAAI Conference on Artificial Intelligence*. AAAI Press.

[97] Dias, M., Zlot, R., Kalra, N., and Stentz, a. (2006) Market-Based Multirobot Coordination: A Survey and Analysis. *Proceedings of the IEEE*, **94**, 1257–1270.

[98] Gerkey, B. and Mataric, M. (2002) Sold!: auction methods for multirobot coordination. *IEEE Transactions on Robotics and Automation*, **18**, 758–768.

[99] Jackson, M. O. (2003) Mechanism theory. In Devigs, U. (ed.), *Optimization and Operations Research* The Encyclopedia of Life Support Science. EOLSS Publishers, Oxford, UK.

[100] Maskin, E. S. and Sjöström, T. (2002) Implementation theory. In Arrow, K. J., Sen, A. K., and Suzumura, K. (eds.), *Handbook of Social Choice Theory and Welfare*. North-Holland, Amsterdam.

[101] Nisan, N. (2007) Introduction to mechanism design (for computer scientists). In Nisan, N., Roughgarden, T., Tardos, E., and Vazirani, V. V. (eds.), *Algorithmic Game Theory*. Cambridge University Press.

[102] Walsh, W. E. and Wellman, M. (2003) Decentralized Supply Chain Formation: A Market Protocol and Competitive Equilibrium Analysis. *Journal of Artificial Intelligence Research*, **19**, 513–567.

[103] Cerquides, J., Endriss, U., Giovannucci, A., and Rodriguez-Aguilar, J. A. (2007) Bidding languages and winner determination for mixed multi-unit combinatorial auctions. *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pp. 1221–1227. Morgan Kaufmann.

[104] Winsper, M. and Chli, M. (2010) Decentralised Supply Chain Formation: A Belief Propagation-Based Approach. *Workshop on Agent-Mediated Electronic Commerce*, pp. 1–6.

[105] Penya-Alba, T., Cerquides, J., Rodriguez-Aguilar, J., and Vinyals, M. (2012) A Scalable Message-Passing Algorithm for Supply Chain Formation. *Proceedings of the Twenty-Sixth Conference on Artificial Intelligence, AAAI 2012*, pp. 1436–1442. AAAI Press.

[106] Witzel, A. and Endriss, U. (2010) Time Constraints in Mixed Multi-unit Combinatorial Auctions. *Workshop on Agent-Mediated Electronic Commerce*.

[107] Pan, F. and Nagi, R. (2010) Robust supply chain design under uncertain demand in agile manufacturing. *Computers & Operations Research*, **37**, 668–683.

[108] Muñoz i Solà, V. (2010) Robustness on resource allocation problems. PhD thesis University of Girona.

[109] Fox, M., Gerevini, A., Long, D., and Serina, I. (2006) Plan Stability : Replanning versus Plan Repair. *Proceeding of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 212–221. AAAI.

[110] Ramchurn, S. D., Mezzetti, C., Giovannucci, A., Rodriguez-Aguilar, J. A., Dash, R. K., and Jennings, N. R. (2009) Trust-based mechanisms for robust and efficient task allocation in the presence of execution uncertainty. *Journal of Artificial Intelligence Research*, **35**, 119–159.

[111] Bofill, M., Busquets, D., Muñoz, V., and Villaret, M. (2013) Reformulation based maxsat robustness. *Constraints*, **18**, 202–235.

[112] Greenstadt, R., Pearce, J. P., and Tambe, M. (2006) Analysis of privacy loss in distributed constraint optimization. *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06), July 2006*, pp. 647–653. AAAI Press.

[113] Petcu, A., Faltings, B., and Parkes, D. (2008) M-DPOP: Faithful distributed implementation of efficient social choice problems. *Journal of Artificial Intelligence Research (JAIR)*, **32**, 705–755.