A Distributed Optimization Method for the Geographically Distributed Data Centres Problem[⋆⋆⋆]

Mohamed Wahbi, Diarmuid Grimes, Deepak Mehta, Kenneth N. Brown, Barry O'Sullivan

Insight Centre for Data Analytics, School of Computer Science and IT, University College Cork, Ireland
firstname.lastname@insight-centre.org

Abstract. The *geographically distributed data centres* problem (GDDC) is a naturally distributed resource allocation problem. The problem involves allocating a set of virtual machines (VM) amongst the data centres (DC) in each time period of an operating horizon. The goal is to optimize the allocation of workload across a set of DCs such that the energy cost is minimized, while respecting limitations on data centre capacities, migrations of VMs, etc.. In this paper, we propose a distributed optimization method for GDDC using the distributed constraint optimization (DCOP) framework. First, we develop a new model of the GDDC as a DCOP where each DC operator is represented by an agent. Secondly, since traditional DCOP approaches are unsuited to these types of large-scale problem with multiple variables per agent and global constraints, we introduce a novel semi-asynchronous distributed algorithm for solving such DCOPs. Preliminary results illustrate the benefits of the new method.

1 Introduction

Distributed constraint reasoning (DCR) gained an increasing interest in recent years due to its ability to handle cooperative multi-agent problems that are naturally distributed. DCR has been applied to solve a wide range of applications in multi-agent coordination such as distributed scheduling [20], distributed planning [7], distributed resource allocation [24], target tracking in sensor networks [23], distributed vehicle routing [17], optimal dispatch in smart grid [22], etc. These applications can be solved by a centralized approach once the knowledge about the problem is delivered to a centralized authority. However, in such applications, it may be undesirable or even impossible to gather the whole problem knowledge into a single authority. In general, this restriction is mainly due to privacy and/or security requirements: constraints may represent strategic information that should not be revealed to other agents that can be seen as competitors, or even to a central authority. The cost or the inability of translating all information to a single format may be another reason: in many cases, constraints arise from complex decision processes that are internal to an agent and cannot be articulated to a central authority. More reasons why distributed methods may be desirable for such applications and often make a centralized process inadequate have been listed in [11].

[⋆] This work is funded by the European Commission under FP7 Grant 608826 (GENiC - Globally Optimised Energy Efficient Data Centres).

[⋆⋆] This work is funded by Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289.

In DCR, a problem is expressed as a distributed constraint network. A distributed constraint network is composed of a group of autonomous agents where each agent has control of some elements of information about the problem, that is, variables and constraints. Each agent owns its local constraint network, and variables in different agents are connected by constraints. Traditionally, there are two large classes of distributed constraint networks. The first class considers problems where all constraints are described by boolean relations (*hard* constraint) on possible assignments of variables they involve. They are called *distributed constraint satisfaction problems* (DisCSP). The second class of problems are *distributed constraint optimization problems* (DCOP) where constraints are described by a set of *cost* functions for combinations of values assigned to the variables they connect. In DisCSP, the goal is to find assignments of values to variables such that all (hard) constraints are satisfied while in DCOP the goal is to find assignments that minimize the objective function defined by the sum of all constraint costs.

Researchers in the DCR field have developed a range of different constraint satisfaction and optimisation algorithms. The main algorithms and protocols include synchronous [37,15,10], asynchronous [38,6,23,36] and semi-synchronous [21,34,13] search, dynamic programming methods [25], and algorithms which respect privacy and autonomy [35,8] versus those which perform local search [16,40]. In order to simplify the algorithm specification, most of these algorithms assume that all constraints are binary, and that each agent controls exactly one variable. Such assumptions simplify the algorithm specification but represent a limitation in the adoption of distributed constraint reasoning techniques in real-world applications.

The first assumption was justified by techniques which translated non-binary problems into equivalent binary ones; however, recent research has demonstrated the benefits of handling non-binary constraints directly in distributed algorithms [5,32]. The second assumption is justified by the fact that any DCR problem with several variables per agent can be solved by those algorithms once transformed using one of the following reformulations [37,9]: (i) compilation, where for each agent we define a new variable whose domain is the set of solutions to the original local problem; (ii) decomposition, where for each agent we create a virtual agent for each of its variables. However, neither of these methods scales up well as the size of the local problems increase, either (i) because of the space and time requirements of the reformulation, or (ii) because of the extra communication overhead and the loss of a complete view on the local problem structure. Only a few algorithms for handling multiple local variables in DisCSP have been proposed [3,39,19]. These algorithms are specific to DisCSP, since they reason about hard constraints, and cannot be applied directly to DCOP, which are concerned with costs. Another limitation is that most DCOP algorithms do not actively exploit hard constraints as they require all constraints to be expressed as cost functions.

In this paper, we present a general distributed model for solving real-life DCOPs, including hard constraints, in order to model the geographically distributed data centres problem [27,4,29,28]. After finding that traditional DCOP approaches are unsuited to these types of large-scale problem with multiple variables per agent and global (hard) constraints, we introduce, AGAC-ng, a novel optimization distributed search algorithm for solving such DCOPs. In the AGAC-ng algorithm agents assign their variables and generate a partial solution sequentially and synchronously following an ordering on agents. However, generated partial solutions are propagated asynchronously by agents with unassigned variables. The concurrent propagation of partial solutions enables early detection of a need to backtrack and saves a lot of search effort. AGAC-ng can perform bounded-error approximation while maintaining a theoretical guarantee on solution quality. In AGAC-ng, a global upper bound which represents the cost of the best known solution to the problem is maintained by all agents. When propagating partial solutions agents ensure that the current lower bound on the global objective augmented by the bounded-error distance does not exceed the global upper bound.

Data centres consume considerable amounts of energy: an estimated 416.2 terawatt-hours was consumed in 2014 [2] with the US accounting for approximately 91 terawatts [1]. While many industries and economies are dependent on such infrastructure, the increase in high-computing requirements for cloud-based services around internet-of-things, big data, etc. have lead to some experts predicting that this consumption could treble in the next decade [2] unless significant breakthroughs are made in reducing consumption.

Geographically distributed data centres present many possible benefits in terms of reducing energy costs through global, rather than local, optimisation. In particular each location may have different unit energy costs, external weather conditions, local renewable sources, etc. Therefore reasoning at a global level can exploit these differences through optimally reallocating workload in each time period through migration of *virtual machines* (VMs), subject to constraints on number of migrations, virtual machine sovereignty, data centre capacities, etc.

Gathering the whole knowledge about the problem into a centralized location may not be feasible in the geographically distributed data centres because of the large amount of information (problem specifications) each data centre would need to communicate to the centralized solver to solve the problem. In addition, data centres may wish to keep some information about their local constraints, costs, and topology confidential and not share it with other data centres.

The maximum energy that can be consumed by a data centre and its running capacity might also be confidential information that data centres want to keep private from operators of other data centres. Thus, the geographically distributed data centres is a naturally distributed resource allocation problem where data centres may not be willing to reveal their private information to other data centres. In addition, sending the whole knowledge about the problem to a centralized location will create a bottleneck on the communication towards that location. Thus, a distributed solving process is preferred for the geographically distributed data centres problem.

This paper is structured as follows. Section 2 gives the necessary background for distributed constraint reasoning and the general definition of the geographically distributed data centres. We present our model of the GDDC problem as DCOP in Section 3. Section 4 introduces our new algorithm, AGAC-ng, for solving DCOP with global hard constraints and multiple variables per agent. We report experimental results in Section 5. Section 6 gives a brief overview of related works on distributed constraint reasoning. Finally, we conclude the paper in Section 7.

2 Preliminaries/Background

2.1 Distributed Constraint Optimization Problem

A *constraint satisfaction problem* (CSP) has been defined for a centralized architecture by a triple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, where $\mathcal{X} = \{x_1, \dots, x_p\}$ is a set of p variables, $\mathcal{D} = \{d_1, \dots, d_p\}$ is the set of their respective finite domains, and $\mathcal{C}$ is a set of (hard) constraints. A constraint $c(X) \in \mathcal{C}$ is a relation over the ordered subset of variables $X = (x_{j_1}, \dots, x_{j_k})$, which defines those value combinations which may be assigned simultaneously to the variables in X. $|X|$ is the *arity* of $c(X)$, and X is its *scope*. A tuple $\tau = (v_{j_1}, \dots, v_{j_k}) \in c(X)$ is a *support* for $c(X)$, and $\tau[x_i]$ is the value of x_i in τ. A solution to a CSP is an assignment to each variable of a value from its domain, such that all constraints are satisfied.

A *global constraint* captures a relation over an arbitrary number of variables. For example, the AllDiff constraint states that the values assigned to the variables in its scope must all be different [30]. Filtering algorithms which exploit the specific structure of global constraints are one of the main strengths of constraint programming. During search, any value $v \in d_i$ that is not *generalized arc-consistent* (GAC) can be removed from d_i. A value $v_i \in d_i$, $x_i \in X$ is ***generalized arc-consistent*** with respect to a constraint $c(X)$ iff there exists a support τ for $c(X)$ such that $v_i = \tau[x_i]$, and for every $x_j \in X$, $x_i \neq x_j$, $\tau[x_j] \in d_j$. Variable x_i is GAC if all its values are GAC with respect to every constraint in $\mathcal{C}$. A CSP is GAC if all its variables are GAC.

A *distributed constraint satisfaction problem* (DisCSP) is a CSP where the variables, domains and constraints of the underlying network are distributed over a set of autonomous agents. Formally, a distributed constraint network (DisCSP) is defined by a 5-tuple $(\mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C}, \varphi)$, where $\mathcal{X}, \mathcal{D}$ and $\mathcal{C}$ are as above. $\mathcal{A}$ is a set of n agents $\{a_1, \dots, a_n\}$, and $\varphi : \mathcal{X} \rightarrow \mathcal{A}$ is a function specifying an agent to control each variable. Each variable belongs to one agent and only the agent who owns a variable has control of its value and knowledge of its domain. Let X_i denotes the set of variables belonging to agent a_i, i.e. $X_i = \{\forall x_j \in \mathcal{X} : \varphi(x_j) = a_i\}$. Agent a_i knows all constraints, $\mathcal{C}_i$, involving its variables X_i. X_i can be partitioned in two disjoint subsets $P_i = \{x_j \mid \forall c(X) \in \mathcal{C}_i, x_j \in X \rightarrow X \subseteq X_i\}$ and $E_i = X_i \setminus P_i$. P_i is a set of private variables, which only share constraints with variables inside a_i. Conversely, E_i is a set of variables linked to the outside world and sometimes referred to as external (or negotiation) variables.

This distribution of variables divides $\mathcal{C}$ in two disjoint subsets, $\mathcal{C}^{intra}$ which are between variables of same agent (i.e., $c(X) \in \mathcal{C}^{intra}$, $X \subseteq X_i$), and $\mathcal{C}^{inter}$ which are between variables of different agents called intra-agent (private) and inter-agent constraint sets, respectively. An intra-agent constraint $c(X)$ is known by the agent owner of X, and it is unknown by other agents. Usually, it is considered that an inter-agent constraint $c(X)$ is known by every agent owning a variable in X. Two agents are *neighbours* if they control variables that share a constraint; we denote by $\mathcal{N}_i$ the set of neighbours of agent a_i.

As in the centralized case, a solution to a DisCSP is an assignment to each variable of a value from its domain, satisfying all constraints. A *distributed constraint optimisation problem* (DCOP) is defined by a DisCSP $(\mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C}, \varphi)$, together with an objective function. A solution to a DCOP is a solution for the DisCSP which is optimal with respect to the objective function.[1]

For the rest of the paper we consider a generic agent $a_i \in \mathcal{A}$. Agent a_i stores a unique ***order*** on agents, i.e. an ordered tuple of all the agent IDs, denoted by $\prec_o$. $\prec_o$ is called the current order of a_i. Agents appearing before agent a_i in $\prec_o$ are the higher priority agents (predecessors) and conversely the lower priority agents (successors) are agents appearing after a_i in $\prec_o$. For sake of clarity, we assume that the order is the lexicographic ordering $[1, 2, \dots, n]$. Each agent maintains a counter, and increments it whenever it changes its assignment. The current value of the counter *tags* each generated assignment.

Definition 1. *An* ***assignment*** *for an agent $a_i \in \mathcal{A}$ is an assignment for each external variable of a value from its domain. That is, a tuple $(\langle E_i, V_i \rangle, t_i)$ where $V_i \in \underset{x_j \in E_i}{\times} d_j$,*

[1] Cost functions can be implemented using element constraints [31].

$V_i[x_j] \in d_j$ and t_i is the tag value. When comparing two assignments, the most up to date is the one with the greatest tag t_i.

Definition 2. *A **current partial assignment** CPA is an ordered set of assignments of external variables $[(\langle E_1, V_1\rangle, t_1), \ldots, (\langle E_k, V_k\rangle, t_k)]$. Two CPAs are **compatible** if the values of each* common *variable amongst the two CPAs are equal. $ag(\text{CPA})$ is the set of all agents with assigned variables in the CPA.*

Definition 3. *A **timestamp** associated with a CPA is an ordered list of counters $[t_1, \ldots, t_k]$ where $\forall j \in 1..k$, t_j is the tag of the agent a_j. When comparing two CPAs, the **strongest** one is that associated with the lexicographically greater timestamp. That is, the CPA with greatest value on the first counter on which they differ, if any, otherwise the longest one.*

During search agents can infer inconsistent sets of assignments called no-goods.

Definition 4. *A **no-good** or conflict set is an assignment set of variables that is not contained in any solution. A no-good is a clause of the form $\neg[(x_i = v_i) \wedge (x_j = v_j) \wedge \ldots \wedge (x_k = v_k)]$, meaning that these assignments cannot be extended to a solution. We say that a no-good is **compatible** with a CPA if every common variable is assigned the same value in both.*

Agents use these no-goods to prune the search space. To stay polynomial we only keep no-goods that are compatible with the current state of the search. These no-goods can be a direct consequence of propagating constraints (e.g., any assignment set that violates a constraint is a no-good) or can be derived from a set of no-goods. Literally, when all values of the variable x_k are ruled out by some no-good, they are resolved computing a new no-good as follows. The new generated no-good is the conjunction of all these no-goods for values of x_k removing variable x_k. If the generated no-good contains assignments of local variables, agent a_i uses this no-good locally to prune the search space. Otherwise, agent a_i reports the no-good to the agent having the lowest priority among those having variables in the new generated no-good.

2.2 Geographically Distributed Data Centres Problem

The geographically distributed data centres (GDDC) problem considered here can be defined as follows. We are given a set of data centre locations L, and a set of virtual machines that must be assigned to physical servers in each time period of an operating horizon T. Each location has its own unit energy price ep^l_t for each time period (sample real time prices are shown in Figure 1). The cost of executing a virtual machine (VM) in a location can differ for each time period for the same location, and for each location in the same time period. This cost constitutes not only the electricity price from the local utility, but is also affected by external factors such as the outdoor temperature, equipment quality, etc. The latter can be estimated using the *Power Usage Effectiveness* (PUE) which is the ratio of the total power consumption of a DC to the power consumption of the IT equipment alone.

Furthermore each location has an associated region k that a subset of *sovereign* VMs can only be performed in (e.g., for security reasons). Therefore for some VMs the set of locations where they can be performed is restricted. A data centre has a maximum capacity on the IT power consumption $pmax^l$ (aggregated across all servers), and a maximum number of migrations in/out, $mmax^l$, that can occur in each time period across all VM types.

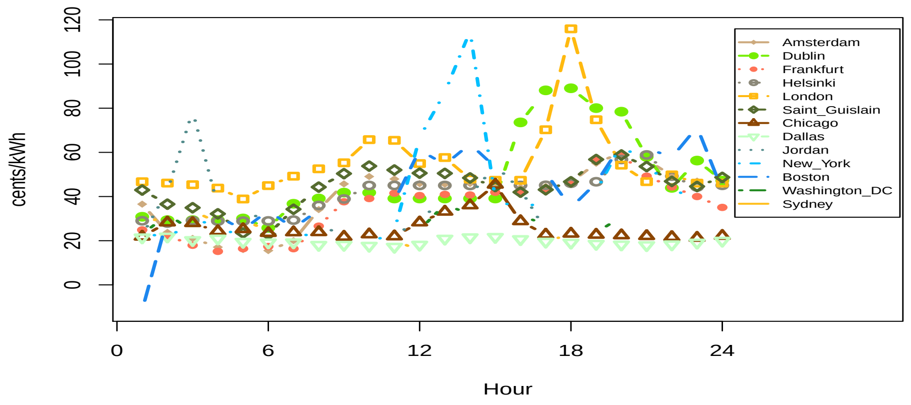

Fig. 1: Sample real time prices for 13 different locations over the same 24 hour period.

There is a set of VM types V, where each type v has an associated power consumption p_v, and quantity n_v that are running in each time period. The type of a VM type can be further subdivided into those that can be run everywhere and those that can only be run in a specific region, i.e. that can only be run at its current location or in one of a limited set of other locations (e.g., a subset of VMs can only be run in European data centres). Let R^l be the set of VM types that can be run in location l.

We must then decide how many VMs of each type to allocate to each data centre in each time period such that the total energy cost of performing the VMs over the horizon, plus the costs of all migrations (where emi_v/emo_v is the energy cost of migrating a vm in/out), is minimized. This allocation is subject to capacity restrictions in the data centres, limitations on the number of migrations per data centre per time period, and limitations on migration of sovereign VMs.

3 GDDC as DCOP

The geographically distributed DCs can naturally be modeled as a DCOP as follows. (For simplicity we consider the time periods to be of duration one hour and thus energy and power values can be used interchangeably.)

Agents.

– Each DC is represented by an agent $\mathcal{A} = L$.

(External) Variables.

– In each DC/agent there is an integer variable x^l_{vt} that represents the number of VMs of type v allocated to DC/agent a_l in period t.
– In each DC/agent there is an integer variable c^l that indicates the total energy cost for running and migrating VMs in R^l over all time-periods.

Note here that agents only have variables x^l_{vt} if $v \in R^l$. This is sufficient for enforcing the sovereignty constraint.

(Private) Variables.

– In each DC/agent there is an integer variable mi^l_{vt} that indicates how many VMs of type v are migrated *in* to the DC a_l in time-period t.
– In each DC/agent there is an integer variable mo^l_{vt} that indicates how many VMs of type v are migrated *out* of the DC a_l in time-period t.
– In each DC/agent there is a real variable u^l_t that indicates the energy consumption of DC l for time-period t.
– In each DC/agent there is an integer variable r^l that indicates the total energy cost of running VMs in R^l over the entire horizon.
– In each DC/agent there is an integer variable m^l that indicates the total energy cost for migrating VMs in R^l over all time-periods.

(Intra-agent) Constraints.

We present here the intra-agent constraints held by agent $a_l \in \mathcal{A}$. Only agent/DC a_l is aware of the existence of its intra-agent constraints.
Allocation. The number of VMs of a given type running in time period are equal to the number running in the previous time period plus the incomings, minus the outgoings.

$$\forall_{v\in R^l}, \forall_{t\in T}: \quad x^l_{v(t+1)} = x^l_{vt} + mi^l_{vt} - mo^l_{vt} \tag{3.1}$$

Capacity. The total energy consumed by a DC a_l for running VMs is bounded:

$$\forall_{t\in T}: \quad u^l_t = \sum_{v\in R^l} x^l_{vt} \cdot p_v \leq pmax^l \tag{3.2}$$

Migration. The amount of incoming/outgoing VMs per time period for each DC is limited by a given threshold:

$$\forall_{t\in T}: \quad \sum_{v\in R^l} mi^l_{vt} + mo^l_{vt} \leq mmax^l \tag{3.3}$$

We further add the following redundant constraint on migration in/out variable pairs, enforcing that at least one must be 0 in every location in every time period for every type:

$$\forall_{v\in R^l}, \forall_{t\in T}: \quad mi^l_{vt} = 0 \vee mo^l_{vt} = 0 \tag{3.4}$$

Running cost. The total energy cost of running VMs in a_l over the entire horizon is:

$$r^l = \sum_{t\in T} \quad ep^l_t \cdot u^l_t \tag{3.5}$$

Migration cost. The total energy cost for migrating VMs in R^l over all time-periods is:

$$m^l = \sum_{v\in R^l} \sum_{t\in T} (mi^l_{vt} \cdot emi_v + mo^l_{vt} \cdot emo_v) \cdot ep^l_t \tag{3.6}$$

Internal cost. The total energy cost for running and migrating VMs in R^l over all time-periods is:

$$c^l = r^l + m^l \tag{3.7}$$

(Inter-agent) Constraints.

An inter-agent constraint is totally known by all agents owning variables it involves. **Assignment.** VMs of type v must all be assigned to DCs in each time-period where v is in R^l:

$$\forall_{v \in V}, \forall_{t \in T} : \qquad \sum_{l \in L | v \in R^l} x^l_{vt} = n_v \tag{3.8}$$

Objective. The global objective is to minimize the sum of the total energy cost of running VMs in all DCs together with total energy cost for migrating VMs over the entire horizon:

$$obj = \sum_{l \in L} c^l$$

4 Nogood-based Asynchronous Generalized Arc-Consistency AGAC-ng

To solve a challenging distributed constraint optimization problem such as GDDC, we propose a new DCOP algorithm, called AGAC-ng (*nogood-based asynchronous generalized arc-consistency*). To the best of our knowledge, AGAC-ng is the first algorithm for solving DCOP with multiple variables per agent and non-binary and hard constraints that can find the optimal solution, or a solution within a user-specified distance from the optimal using polynomial space at each agent.

When solving distributed constraint networks, the solution process is restricted: each agent a_i is only responsible for making decisions (assignments) of the variables it controls (X_i). Thus, agents must communicate with each other exchanging messages about their variable assignments and conflicts of constraints in order to find a global (optimal) solution. Several distributed algorithms for solving the DCOP have been designed by the distributed constraint reasoning community. Regarding the manner on which assignments are processed and search performed on these algorithms, they can be categorized into synchronous, asynchronous, or semi-synchronous.

The first category consists of those algorithms in which agents assign values to their variables in a synchronous and sequential way. Although synchronous algorithms do not exploit the parallelism inherent from the distributed system, their agents receive consistent information from each other. The second category consists of algorithms in which the process of proposing values to the variables and exchanging these proposals is performed concurrently and asynchronously between agents. Agents take advantage from the distributed formalism to enhance the degree of concurrency. However, in asynchronous algorithms, the global assignment state at any particular agent is in general inconsistent. The third category is that of algorithms combining both sequential value assignments by agents together with concurrent computation. Agents take advantage from both the above-mentioned categories: they perform concurrent computation while exchanging consistent information between agents.

In this section we propose a novel semi-synchronous search algorithm for optimally solve DCOPs called AGAC-ng (*nogood-based asynchronous generalized arc-consistency*). In AGAC-ng algorithm, agents assign their variables and generate a partial solution sequentially and synchronously following an ordering on agents. However, generated partial solutions are propagated asynchronously by neighbours with unassigned variables. The concurrent propagation of partial solutions enables an early detection of a need to backtrack and saves search effort.

AGAC-ng incorporates an asynchronously generalized arc-consistency phase in a synchronous search procedure. AGAC-ng follows an ordering on agents to perform the sequential assignments by agents. Agents assign their variables only when they hold the CPA. The CPA is a unique message (token) that is passed from one agent to the next in the ordering. The CPA message carries the current partial assignment that agents attempt to extend into a complete solution by assigning their variables in it.[2] When an agent succeeds in assigning its variables on the CPA, it sends this CPA (token) to the next agent on the ordering. Furthermore, copies of the CPA are sent to all neighbors whose assignments are not yet on the CPA. These agents maintain the generalized arc-consistency asynchronously in order to detect as early as possible inconsistent partial assignments. The generalized arc-consistency process is performed as follows. When an agent receives a CPA, it updates the domain of its variables and copies of neighbors variables, removing all values that are not GAC using the no-goods as justification of value deletions.

When an agent generates an empty domain as a result of maintaining GAC, it resolves the no-goods ruling out values from that domain, producing a new no-good. Then, the agent backtracks to the agent with the lowest priority in the conflict by sending the resolved no-good. Hence, multiple backtracks may be performed at the same time coming from different agents having an empty domain. These backtracks are sent concurrently by these different agents to different destinations. The reassignments of the destination agents then happen simultaneously and generate several CPAs. However, the strongest CPA coming from the highest level in the agent ordering will eventually dominate all others. Agents use timestamps attached to CPAs to detect the strongest one (see Definition 3). Interestingly, the search process of higher levels with stronger CPA can use no-goods reported by the (killed) lower level processes, so that it benefits from their computational effort.

4.1 The algorithm description

Each agent $a_i \in \mathcal{A}$ executes the pseudo-code shown in Figure 2. The agent a_i has a local solver where it stores and propagates the most up-to-date assignments received from higher priority agents (*solver*.CPA) w.r.t. the agent ordering ($\prec_o$). In AGAC-ng, agents exchange the following types of messages (where a_i is the sender):

ok?: agent a_i passes on the current partial assignment (CPA) to a lower priority agent. According to the ID of the agent that has the token attached to the message by a_i, the receiver will try to extend the CPA (when it is the next agent on the ordering) or maintain generalized arc-consistency phase.

ngd: agent a_i reports the inconsistency to a higher priority agent. The inconsistency is reported by a no-good (i.e., a subset of the CPA).

sol: agent a_i informs all other agents of the new best solution (CPA) and new better bound.

stp: agent a_i informs agents if either an optimal solution is found or the problem is found to be unsolvable.

Agent a_i running the AGAC-ng algorithm starts the search by calling procedure `initialize` in which a_i sets the upper bound to $+\infty$ and initializes its local *solver* and sets the objective variable to minimize (lines 9 to 11) before setting counter tags

[2] A complete solution here is a complete assignments of all agents' external variables.

of other agents to zero (line 12). If agent a_i is the initialising agent (the first agent in the agent ordering $\prec_o$), it initiates the search by calling procedure `assign` (line 15) after setting itself as the agent that has the token (i.e., the privilege to make decisions) (line 14). Then, a loop considers the receiving and processing of the possible message types (lines 2 to 8).

When calling procedure `assign`, agent a_i tries to find a local solution, which is consistent with the assignments of higher agents (solver.CPA). If agent a_i fails to find a consistent assignment, it calls procedure `backtrack` (line 23). If a_i succeeds, it increments its counter t_i and generates a CPA from higher agents assignments (solver.CPA) augmented by the assignments of its external variables (E_i), lines 17 to 18.[3] If the CPA includes assignments of all agents (a_i is the last agent in the order), agent a_i calls procedure `reportSolution()` to report a new solution (line 20). Otherwise, agent a_i sends forward the CPA to every neighboring agent ($\mathcal{N}_i \setminus ag(\text{CPA})$) whose assignments are not yet on the CPA including the next agent that will extend the CPA (i.e., the agent that will have the token) (lines 28 to 30) by calling procedure `sendForward()`, line 22.

When agent a_i (the last agent) calls procedure `reportSolution`, a complete assignment has been reached, with a new global cost of B (line 24). Agent a_i sends the full current partial assignment (CPA), i.e. solution, with the new global cost to all other agents (line 25). Agent a_i calls then procedure `storeSolution()` (line 26) to set the upper bound to new global cost value and the best solution to the newly found one, lines 54 to 55, and to post a new constraint requiring that the global cost should not exceed the cost of the best solution found so far (taking into account the error-bound). Finally, agent a_i calls procedure `assign()` to continue the search for new solutions with better cost (line 27). Whenever agent a_i receives a **sol** message it calls procedure `storeSolution()`.

Whenever a_i receives an **ok?** message, procedure `processOK` is called (line 5). Agent a_i checks if the received CPA is stronger than its current CPA (solver.CPA) by comparing the timestamp of the received CPA to that stored locally, function `compareTimeStamp` call (line 31). If it is not the case, the received CPA is discarded. Otherwise, a_i sets the token to the newly received one and updates the local counters of all agents in the CPA by those freshly received (lines 33 to 35). Next, agent a_i updates its solver (*solver*.`update`) to include all assignments of agents in the received CPA and propagates their effects locally (line 36). If agent a_i generates an empty domain as a result of calling *solver*.`update`, a_i calls procedure `backtrack` (line 39), otherwise, a_i checks if it has to assign its variables (if it has the token) and then calls procedure `assign` if it is the case (line 38).

When agent a_i generates an empty domain after propagating its constraints, the procedure `backtrack` is called to resolve the conflict. Agent a_i requires its local solver to explain the failure by generating a new no-good (ng), that is, the subset of assignments that produced the inconsistency, (line 40). If the new no-good (ng) is empty, agent a_i terminates execution after sending a **stp** message to all other agents in the system meaning that the last solution found is the optimal one (line 42). Otherwise, agent a_i sets the token to be the agent having the lowest priority among those having variables in the newly generated no-good ng. If the token is different than a_i, the no-good ng is reported to *token* through a **ngd** message, line 46. Otherwise, a_i has to seek a new local solution for its variables by calling procedure `assign` after storing the generated no-good in its local solver (lines 48 to 49).

When a **ngd** message is received by an agent a_i, it checks the validity of the received no-good (line 50). A no-good is valid if the assignments on it are consistent with those

[3] Only external variables linked to unassigned neighbors are needed.

```
procedure AGAC-ng()
01. initialize();
02. while ( $\neg end$ ) do
03.     $msg \leftarrow$ getMsg();
04.     switch ( $msg.type$ ) do
05.         ok?: processOK($msg.cpa$, $msg.next$);
06.         ngd: processNgd($msg.nogood$);
07.         sol : storeSolution($msg.sol$,$msg.b$);
08.         stp : $end \leftarrow$ true ;

procedure initialize()
09. $end \leftarrow$ false; $solution \leftarrow \emptyset$; $UB \leftarrow +\infty$;
10. solver.initialize();
11. solver.setObjective($obj$);
12. foreach ( $a_j \in \mathcal{A}$ ) do $t_j \leftarrow 0$;
13. if ( isFirstAgent($\prec_o$) ) then
14.     $token \leftarrow a_i$;
15.     assign();

procedure assign()
16. if ( solver.findSolution() ) then
17.     $t_i \leftarrow t_i + 1$;
18.     CPA $\leftarrow \{solver.$CPA $\cup E_i\}$;
19.     if ( $n = |ag($CPA$)|$ ) then
20.         reportSolution(CPA);
21.     else
22.         sendForward(CPA, nextAgent());
23. else  backtrack() ;

procedure reportSolution($cpa$)
24. $B \leftarrow obj$.getValue();
25. sendMsg:sol$\langle cpa, B\rangle$ to $\mathcal{A} \setminus a_i$;
26. storeSolution($cpa$, $B$);
27. assign();

procedure sendForward($cpa$, $a_j$)
28. $token \leftarrow a_j$;
29. foreach ( $a_k \in \{\mathcal{N}_i \setminus ag($CPA$)\}$ ) do
30.     sendMsg:ok?$\langle cpa, a_j\rangle$ to $a_k$;

procedure processOK($cpa$, $next$)
31. $s \leftarrow$ compareTimeStamp($cpa$);
32. if ( $s > 0$ ) then
33.     $token \leftarrow next$;
34.     foreach ( $t_j \in cpa$ ) do
35.         $t_j \leftarrow cpa.t_j$;
36.     if ( solver.update($cpa[s..]$) ) then
37.         $t_i \leftarrow 0$;
38.         if ( $token = a_i$ ) then assign();
39.     else  backtrack();

procedure backtrack()
40. $ng \leftarrow$solver.explainFailure() ;
41. if ( $ng = \emptyset$ ) then
42.     sendMsg:stp$\langle sol, UB\rangle$ to $\{\mathcal{A} \setminus a_i\}$;
43. else
44.     $token \leftarrow ng$.lastAgent();
45.     if ( $token \neq a_i$ ) then
46.         sendMsg:ngd$\langle ng\rangle$ to $token$;
47.     else
48.         solver.post($ng$);
49.         assign();

procedure processNgd($ng$)
50. if ( isCompatible($ng$) ) then
51.     $token \leftarrow a_i$;
52.     solver.post($ng$);
53.     assign();

procedure storeSolution($cpa$, $bound$)
54. $solution \leftarrow cpa$;
55. $UB \leftarrow bound$;
56. solver.post($obj < UB - errorBound$);

function isCompatible($assignments$)
57. foreach ( $x_j \in assignments$ ) do
58.     if ( $x_j \neq solver.x_j$ ) then
59.         return(false);
60. return(true);

function compareTimeStamp($cpa$)
61. from ( $j \leftarrow 1$ to size($cpa$) ) do
62.     Let $a_k \leftarrow cpa[j]$;
63.     if ( $cpa.t_k > t_k$ ) then
64.         return($j$);
65.     if ( $cpa.t_k < t_k$ ) then
66.         return($-j$);
67. return(0);
```

Fig. 2: AGAC-ng algorithm running by agent a_i.

stored locally in *solver*. If the received no-good (ng) is valid, a_i sets itself as the agent having the token (line 51). Next, agent a_i stores ng and propagates it in its local solver. The procedure `assign` is then called to find a new local solution for the variables of a_i (line 53). Finally, when a **stp** message is received, a_i marks the *end* flag as true to stop the main loop (line 8).

5 Evaluation of Distributed Approach

In this section we experimentally evaluate AGAC-ng, the distributed COP algorithm we proposed in Section 4 for solving the DCOP model of the GDDC problem presented in Section 3. All experiments were performed based on the DisChoco 2.0 platform[4] [33], in which agents are simulated by Java threads that communicate only through message passing. We use the Choco-4.0.0 solver as local solver of each agent in the system [26].

5.1 Empirical Setup

Instances were generated for a scenario with 13 data centres across 3 continents, each with a capacity defined to be 40MW. Real time price data was gathered for each location for each hour for a set of five days. Dynamic PUE values of each DC were generated as a function of the temperature across a sample day. There were 5 VM *types* chosen with associated power consumption values of 20W, 40W, 60W, 80W and 100W respectively. For each DC, VM creation petitions are randomly generated until their consumption reaches a load percentage of 40% of the DC capacity. Each VM creation petition is further randomly assigned a "sovereignty", which is either the continent the DC belongs to or the entire DC set.

Finally, instances had a migration limit for each data centre stating what percentage of average VMs per data centre can be migrated per time period. We generated instances with a limit of 5% and with a limit of 10%. There were 3 instances generated with different seeds for each migration limit and for each of the five days of real time price data, producing a set of 30 instances. Therefore instances typically had 10000 variables, with domain sizes ranging from 15 to more than 500. For c^i variables the domains are larger and domain sizes ranges from 10^7 to more than 10^8. The number of constraints was approximately 7500 with maximum arity of 49 variables. The baseline subsequently used for comparison involves the case where there are 0 migrations, i.e. each DC just performs the load that it was initially assigned in time zero across all time periods.

5.2 Search Strategy

The AGAC-ng algorithm requires a total ordering on agents. This ordering is used to pass on the token (the privilege of assigning variables) between agents. The agent ordering can then affect the behaviour of the algorithm and our empirical results (Section 5.3) confirms the effect of agent ordering.

In the following we propose two agent orderings called **o1** and **o2**. In both orderings we use the same measure α_i for an agent a_i. For each agent a_i, α_i is the difference between the most expensive and the cheapest price over all time slots for the DC represented by agent a_i. For **o2**, agents are sorted using an increasing order over α_i. For **o1**, the first

[4] http://dischoco.sourceforge.net

Table 1: Distributed: Number of message exchanged by agents/DCs in solving each instance.

price	Base Euro	AGAC-ng **(o1)**		AGAC-ng **(o2)**	
		Mig 5%	**Mig 10%**	**Mig 5%**	**Mig 10%**
pr1	165.31	15,838	13,619	12,515	24,700
pr2	197.53	1,709	1,966	5,390	6,426
pr3	192.09	3,208	37,528	4,947	2,456
pr4	171.53	18,494	23,909	25,015	23,706
pr5	215.97	5,737	10,798	19,293	14,425

Table 2: Distributed: Results in terms of average monetary cost (in Euros) for migration limits at 5% and 10%.

price	Base Euro	AGAC-ng **(o1)**		AGAC-ng **(o2)**	
		Mig 5%	**Mig 10%**	**Mig 5%**	**Mig 10%**
pr1	165.31	164.11	164.03	162.45	159.71
pr2	197.53	187.92	190.21	186.26	187.92
pr3	192.09	186.85	186.20	186.73	185.58
pr4	171.53	167.55	166.24	166.18	164.11
pr5	215.97	215.69	215.76	215.69	215.40

agent is selected to be the one with median measure α_i followed in increasing order by agents, say a_j, having the smallest distance to the measure α_i, i.e. $\mid \alpha_j - \alpha_i \mid$. For example, let $\alpha_1 = 22$, $\alpha_2 = 10$, $\alpha_3 = 44$, $\alpha_4 = 55$, $\alpha_5 = 30$, thus, **o1**$=[a_5, a_1, a_3, a_2, a_4]$ and **o2**$=[a_2, a_1, a_5, a_3, a_4]$.

We investigated search strategies for the model of each agent. We are using the *migration_in* and *migration_out* variables (mi^i_{vt} and mo^i_{vt}) as decision variables for the local solver of each agent/DC a_i. Every agent/DC a_i only communicates decisions about x^i_{vt} and c^i as they are the external variables of agent a_i while migration variables are private variables. Preliminary results suggested the best approach for solving the problems was to choose decision variables (i.e., *migration in* and *migration out* variables) according to the *domwdeg* variant of the weighted degree heuristic. Values of the *migration in* variables of the cheapest time slot and the *migration out* variables of the most expensive time slot are selected in a decreasing order (the upper bound) and other migration variables on increasing order (the lower bound).

5.3 Empirical Results

We evaluate the performance of the algorithm by communication load and solution quality. Communication load is measured by the total number of exchanged messages among agents during algorithm execution (#*msg*) [18]. Results are presented on Table 1. Solution quality is assessed by two measures: (i) in terms of average monetary cost in € (Table 2) and (ii) in terms of average percentage savings over baseline (Table 3). For all instances we use a timeout limit of one hour per instance and present the average cost (respectively #*msg*) per price/migration-limit instance type (across the three instances).

The results are given for the both static agent ordering presented in Section 5.2 for the three metrics Tables 1 to 3.

Regarding the communication loads shown in Table 1, the agents exchange few messages in solving the problems. In the worst case, AGAC-ng agents exchanges $37,528$ mes-

Table 3: Distributed: Results in terms of average percentage savings over baseline for migration limits at 5% and 10%.

		AGAC-ng **(o1)**		AGAC-ng **(o2)**	
price	**Base Euro**	**Mig 5%**	**Mig 10%**	**Mig 5%**	**Mig 10%**
pr1	165.31	0.73%	0.77%	1.73%	3.39%
pr2	197.53	4.87%	3.71%	5.71%	4.87%
pr3	192.09	2.73%	3.07%	2.79%	3.39%
pr4	171.53	2.32%	3.08%	3.12%	4.33%
pr5	215.97	0.13%	0.10%	0.13%	0.26%

sages to solve the problems. This represents a significant result regarding the complexity and the size of the instances solved by a complete DCOP algorithm. This is mainly due to the expensive local filtering and search per each local solver and the topology of the instances solved: all instances solved have a complete constraint network. Thus, AGAC-ng produces more chronological backtracks than backjumps. This is explained by the behaviour of the algorithm where only small improvements over the first solution/UB were found. After the first solution, the AGAC-ng algorithm mainly backtracks from the last agent on the ordering to the second last that returns the token to the last agent for seeking new solutions and so on.

Comparing the solution quality (Tables 2 and 3), for instances **pr5**, AGAC-ng improvement over the baseline is insignificant. The improvement over the baseline ranges between 0.1-0.26%, which represents less than a euro per day. For other instances, this improvement is more significant mainly for **pr2** where the improvement over the baseline ranges between 3.71% and 5.71%, which represents between 7.32 and 11.27 euros per day. This is an interesting result mainly because agents are solving problems only by passing messages between them, without sharing their constraints or their prices, and keeping their information private.

Comparing two different agents ordering, running AGAC-ng using agent ordering **o2** always improves AGAC-ng using **o1** for pricing. However, this is not always the case for the number of message exchanges to solve the problem. The average cost over all instances when running AGAC-ng using **o2** is 366 and 369 when using **o1**. For communication load, AGAC-ng (**o2**) requires an average of $27,775$ messages over all instances while AGAC-ng (**o2**) requires $26,561$ messages.

Regarding the migration limit, the results suggests that having a smaller migration limit is better for the distributed solving process regarding the number messages exchanges while the solution quality is slightly affected by the change on the migration limits. In the distributed problems having larger domains leads to more messages when the filtering power is limited.

6 Related Work

Many distributed algorithms for solving DisCSP/DCOP have been designed in the last two decades. Synchronous Backtrack (SBT) is the simplest DisCSP search algorithm. SBT performs assignments sequentially and synchronously. In SBT, only the agent holding a current partial assignment (CPA) performs an assignment or backtrack [41]. Meisels and Zivan ([21]) extended SBT to Asynchronous Forward Checking (AFC), an algorithm in which the forward checking propagation [14] is performed asynchronously [21].

In AFC, whenever an agent succeeds to extend the CPA, it sends the CPA to its successor and it sends copies of this CPA to the other unassigned agents in order to perform the forward checking asynchronously. The Nogood-Based Asynchronous Forward Checking algorithm (AFC-ng), which is an improvement of AFC, has been proposed in [34]. Unlike AFC, AFC-ng uses no-goods as justification of value removals and allows several simultaneous backtracks coming from different agents and going to different destinations. AFC-ng was shown to outperform AFC. The pioneering asynchronous algorithm for DisCSP was *asynchronous backtracking* (ABT) [38,6]. ABT is executed autonomously by each agent, and is guaranteed to converge to a global consistent solution (or detect inconsistency) in finite time.

The synchronous branch and bound (SyncBB) [15] is the basic systematic search algorithm for solving DCOP. In SyncBB, only the agent holding the token is allowed to perform an assignment while the other agents remain idle. Once it assigns its variables, it passes on the token and then remains idle. Thus, SyncBB does not make any use of concurrent computation. No-Commitment Branch and Bound (NCBB) is another synchronous polynomial-space search algorithm for solving DCOPs [10]. To capture independent sub-problems, NCBB arranges agents in constraint tree ordering. NCBB incorporates, in a synchronous search, a concurrent computation of lower bounds in non-intersecting areas of the search space based on the constraint tree structure. Asynchronous Forward Bounding (AFB) has been proposed in [12] for DCOP to incorporate a concurrent computation in a synchronous search. AFB can be seen as an improvement of SyncBB where agents extend a partial assignment as long as the lower bound on its cost does not exceed the global upper bound. In AFB, the lower bounds are computed concurrently by unassigned agents. Thus, each synchronous extension of the CPA is followed by an asynchronous forward bounding phase. Forward bounding propagates the bounds on the cost of the partial assignment by sending to all unassigned agents copies of the extended partial assignment. When the lower bound of all assignments of an agent exceeds the upper bound, it performs a simple backtrack to the previous assigned agent. Later, the AFB has been enhanced by the addition of a backjumping mechanism, resulting in the AFB_BJ algorithm [13]. The authors report that AFB_BJ, especially combined with the minimal local cost value ordering heuristic performs significantly better than other DCOP algorithms. The pioneer complete asynchronous algorithm for DCOP is Adopt [23]. Later on, the closely related BnB-Adopt [36] was presented. BnB-Adopt changes the nature of the search from Adopt best-first search to a depth-first branch-and-bound strategy, obtaining better performance.

7 Conclusions

In this paper we studied the geographically distributed data centres problem where the objective is to optimize the allocation of workload across a set of DCs such that the energy cost is minimized. We introduced a model of this problem using the distributed constraint optimization framework. We presented AGAC-ng, nogood-based asynchronous generalized arc-consistency, a new semi-asynchronous algorithm for DCOPs with multiple variables per agent and with non-binary and hard constraints. AGAC-ng can find the optimal solution, or a solution within a user-specified distance form the optimal using polynomial space at each agent. We showed empirically the benefits of the new method for solving large-scale DCOPs.

References

1. America's Data Centres Consuming and Wasting Growing Amounts of Energy, 2015. https://www.nrdc.org/resources/americas-data-centers-consuming-and-wasting-growing-amounts-energy.
2. Data centres to consume three times as much energy in next decade, experts warn, 2016. http://www.independent.co.uk/environment/global-warming-data-centres-to-consume-three-times-as-much-energy-in-next-decade-experts-warn-a6830086.html.
3. Aaron A. Armstrong and Edmund H. Durfee. Dynamic prioritization of complex agents in distributed constraint satisfaction problems. In *Proceedings of AAAI'97/IAAI'97*, pages 822–822, 1997.
4. Anton Beloglazov and Rajkumar Buyya. Energy efficient resource management in virtualized cloud data centers. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 826–831. IEEE Computer Society, 2010.
5. Christian Bessiere, Ismel Brito, Patricia Gutierrez, and Pedro Meseguer. Global constraints in distributed constraint satisfaction and optimization. *The Computer Journal*, 2013.
6. Christian Bessiere, Arnold Maestre, Ismel Brito, and Pedro Meseguer. Asynchronous backtracking without adding links: a new member in the ABT family. *Artif. Intel.*, 161:7–24, 2005.
7. Olivier Bonnet-Torrés and Catherine Tessier. Multiply-constrained dcop for distributed planning and scheduling. In *AAAI Spring Symposium: Distributed Plan and Schedule Management*, pages 17–24, 2006.
8. Ismel Brito, Amnon Meisels, Pedro Meseguer, and Roie Zivan. Distributed Constraint Satisfaction with Partially Known Constraints. *Constraints*, 14:199–234, 2009.
9. David A. Burke and Kenneth N. Brown. Efficient handling of complex local problems in distributed constraint optimization. In *Proceedings of ECAI'2006, 2006, Riva del Garda, Italy*, pages 701–702, 2006.
10. Anton Chechetka and Katia Sycara. No-Commitment Branch and Bound Search for Distributed Constraint Optimization. In *Proceedings of AAMAS'06*, pages 1427–1429, 2006.
11. Boi Faltings and Makoto Yokoo. Editorial: Introduction: Special issue on distributed constraint satisfaction. *Artif. Intell.*, 161(1-2):1–5, January 2005.
12. Amir Gershman, Amnon Meisels, and Roie Zivan. Asynchronous Forward-Bounding for Distributed Constraints Optimization. In *Proceedings of ECAI'06*, pages 103–107, 2006.
13. Amir Gershman, Amnon Meisels, and Roie Zivan. Asynchronous Forward-Bounding for Distributed COPs. *JAIR*, 34:61–88, 2009.
14. Robert M. Haralick and Gordon L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artif. Intel.*, 14(3):263–313, 1980.
15. Katsutoshi Hirayama and Makoto Yokoo. Distributed partial constraint satisfaction problem. In *Principles and Practice of Constraint Programming*, pages 222–236, 1997.
16. Katsutoshi Hirayama and Makoto Yokoo. The Distributed Breakout Algorithms. *Artif. Intel.*, 161:89–116, 2005.
17. Thomas Léauté and Boi Faltings. Coordinating Logistics Operations with Privacy Guarantees. In *Proceedings of the IJCAI'11*, pages 2482–2487, 2011.
18. Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Series, 1997.
19. Arnold Maestre and Christian Bessiere. Improving Asynchronous Backtracking for Dealing with Complex Local Problems. In *Proceedings of ECAI'04*, pages 206–210, 2004.
20. Rajiv T. Maheswaran, Milind Tambe, Emma Bowring, Jonathan P. Pearce, and Pradeep Varakantham. Taking DCOP to the Real World: Efficient Complete Solutions for Distributed Multi-Event Scheduling. In *Proceedings of AAMAS'04*, pages 310–317, Washington, DC, USA, 2004. IEEE Computer Society.
21. Amnon Meisels and Roie Zivan. Asynchronous Forward-checking for DisCSPs. *Constraints*, 12(1):131–150, 2007.

22. Sam Miller, Sarvapali D. Ramchurn, and Alex Rogers. Optimal Decentralised Dispatch of Embedded Generation in the Smart Grid. In *Proceedings of AAMAS'12*, pages 281–288. International Foundation for Autonomous Agents and Multiagent Systems, 2012.
23. Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees. *Artif. Intel.*, 161:149–180, 2005.
24. Adrian Petcu and Boi Faltings. A Value Ordering Heuristic for Distributed Resource Allocation. In *Proceedings of Joint Annual Workshop of ERCIM/CoLogNet on CSCLP'04*, pages 86–97, 2004.
25. Adrian Petcu and Boi Faltings. DPOP: A Scalable Method for Multiagent Constraint Optimization. In *Proceedings of IJCAI'05*, pages 266–271, 2005.
26. Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016.
27. Asfandyar Qureshi, Rick Weber, Hari Balakrishnan, John Guttag, and Bruce Maggs. Cutting the electric bill for internet-scale systems. In *ACM SIGCOMM computer communication review*, volume 39, pages 123–134. ACM, 2009.
28. Ashikur Rahman, Xue Liu, and Fanxin Kong. A survey on geographic load balancing based data center power management in the smart grid environment. *IEEE Communications Surveys & Tutorials*, 16(1):214–233, 2014.
29. Lei Rao, Xue Liu, Le Xie, and Wenyu Liu. Minimizing electricity cost: optimization of distributed internet data centers in a multi-electricity-market environment. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.
30. Jean-Charles Régin. A filtering algorithm for constraints of difference in csps. In *Proceedings of AAAI'94*, pages 362–367, 1994.
31. Pascal Van Hentenryck, Yves Deville, and Choh-Man Teng. A generic arc-consistency algorithm and its specializations. *Artif. Intel.*, 57(2-3):291–321, 1992.
32. Mohamed Wahbi and Kenneth N. Brown. Global Constraints in Distributed CSP: Concurrent GAC and Explanations in ABT. In *Proceedings of CP'2014*, pages 721–737, Lyon, France, 2014. Springer International Publishing.
33. Mohamed Wahbi, Redouane Ezzahir, Christian Bessiere, and El Houssine Bouyakhf. DisChoco 2: A Platform for Distributed Constraint Reasoning. In *Proceedings of workshop on DCR'11*, pages 112–121, 2011.
34. Mohamed Wahbi, Redouane Ezzahir, Christian Bessiere, and El Houssine Bouyakhf. Nogood-Based Asynchronous Forward-Checking Algorithms. *Constraints*, 18(3):404–433, 2013.
35. Richard J. Wallace and Eugene C. Freuder. Constraint-based reasoning and privacy/efficiency tradeoffs in multi-agent problem solving. *Artif. Intel.*, 161:209–228, 2005.
36. William Yeoh, Ariel Felner, and Sven Koenig. BnB-ADOPT: An Asynchronous Branch-and-Bound DCOP Algorithm. *J. Artif. Intell. Res. (JAIR)*, 38:85–133, 2010.
37. Makoto Yokoo. *Distributed Constraint Satisfaction: Foundation of Cooperation in Multi-agent Systems*. Springer Berlin Heidelberg, London, UK, 2001.
38. Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *Proceedings of 12th IEEE Int'l Conf. Distributed Computing Systems*, pages 614–621, 1992.
39. Makoto Yokoo and Katsutoshi Hirayama. Distributed Constraint Satisfaction Algorithm for Complex Local Problems. In *Inter. Conf. Multi Agent Systems*, pages 372–379, 1998.
40. Weixiong Zhang, Guandong Wang, Zhao Xing, and Lars Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artif. Intel.*, 161:55–87, 2005.
41. Roie Zivan and Amnon Meisels. Synchronous vs Asynchronous Search on DisCSPs. In *Proceedings of EUMAS'03*, 2003.