Solving Highly Cyclic Distributed Optimization Problems Without Busting the Bank: A Decimation-based Approach

JESÚS CERQUIDES[*], *Artificial Intelligence Research Institute (IIIA-CSIC), Campus de la UAB, 08193 Bellaterra, Spain.*

JUAN ANTONIO RODRÍGUEZ-AGUILAR[**], *Artificial Intelligence Research Institute (IIIA-CSIC), Campus de la UAB, 08193 Bellaterra, Spain.*

RÉMI EMONET[†], *Univ Lyon, UJM-Saint-Etienne, CNRS, Institut d'Optique Graduate School, Laboratoire Hubert Curien UMR 5516, F-42023 Saint-Etienne, France.*

GAUTHIER PICARD[††], *Mines Saint-Etienne, Univ Lyon, Univ Jean Monnet, IOGS, CNRS, UMR 5516 LHC, Institut Henri Fayol, Departement ISI, F-42023 Saint-Etienne, France.*

Abstract

In the context of solving large distributed constraint optimization problems, belief-propagation and incomplete inference algorithms are candidates of choice. However, in general, when the problem structure is very cyclic, these solution methods suffer from bad performance, due to non-convergence and many exchanged messages. As to improve performances of the MaxSum inference algorithm when solving cyclic constraint optimization problems, we propose here to take inspiration from the belief-propagation-guided decimation used to solve sparse random graphs (k-satisfiability). We propose the novel DECIMAXSUM method, which is parameterized in terms of policies to decide when to trigger decimation, which variables to decimate and which values to assign to decimated variables. Based on an empirical evaluation on a classical constraint optimization benchmarks (graph coloring, random graph and Ising model), some of these combinations of policies, using periodic decimation, cycle detection-based decimation, parallel and non parallel decimation, random or deterministic variable selection and deterministic or random sampling for value selection, outperform state-of-the-art competitors in many settings.

Keywords: Distributed constrained optimization, DCOP, decimation, belief propagation.

1 Introduction

In the context of multi-agent systems, distributed constraint optimization problems (DCOPs) are convenient to model the coordination issues agents have to face, like resource allocation, distributed planning or distributed configuration. In a DCOP, agents manage one or more variables they have

[*]E-mail: cerquide@iiia.csic.es
[**]E-mail: jar@iiia.csic.es
[†]E-mail: remi.emonet@univ-st-etienne.fr
[††]E-mail: picard@emse.fr

Vol. 29, No. 1, © The Author(s) 2020. Published by Oxford University Press. All rights reserved.
For permissions, please e-mail: journals.permission@oup.com.
Advance Access published 19 December 2020 doi:10.1093/jigpal/jzaa069

to assign a value to (e.g. a goal, a decision), while taking into account constraints with other agents. Solving a DCOP consists in making agents communicate so as to minimize the violation of these constraints. Several solution methods exist to solve such problems, from complete and optimal solutions, to incomplete ones. When dealing with large scale, incomplete methods are solutions of choice. Indeed, complete methods, like ADOPT or DPOP, suffer from exponential computation and/or communication costs in general settings [13, 18]. As a consequence, in some large settings, incomplete methods are better candidates, as evidenced by the extensive literature on the subject (see [1] for a complete review). One major difficulty for incomplete methods to solve a DCOP is the presence of cycles in the constraint graph (or factor graph). Among the aforementioned methods, inference-based ones (e.g. MaxSum [3]) and its extensions (e.g. [19]), have demonstrated good performance even on cyclic settings. However, there exist some cases, with numerous loops or large induced width of the constraint graph, where they perform badly, which translates into a larger number of messages, a longer time to convergence and a final solution of bad quality.

One approach to cope with cyclic graphs is to break loops by *decimating* variables during the solving process. Decimation is a method inspired by statistical physics and applied in belief-propagation, which consists in fixing the value of a variable, using the marginal values as the decision criteria to select the variable to decimate [16]. The decimation is processed regularly after the convergence of a classical belief-propagation procedure. In [14], decimation has been used for solving centralized k-satisfiability problems [14]. Inspired by this concept, we propose a general framework for applying decimation in the DCOP setting. Other works proposed MaxSum_AD_VP to improve MaxSum performance on cyclic graphs [25]. The idea is to perform the inference mechanism through an overlay directed acyclic graph, to remove loops and to alternate the direction of edges at a fixed frequency.

Against this background, the main goal of this paper is to propose a general framework for incorporating decimation into MaxSum. More precisely, we make the following contributions:

(1) We propose a parametric solution method, namely DECIMAXSUM, to implement decimation in MaxSum. It takes three fundamental parameters for decimation: (i) a policy stating when to trigger decimation, (ii) a policy stating which variables to decimate and (ii) a policy stating which value to assign to decimated variables. The flexibility of DECIMAXSUM comes from the fact that any policy from (1i) can be combined with any policy from (1ii) and (1iii).
(2) We propose a family of decimation policies; some inspired by the state-of-the-art and some original ones. Many combinations of policies are possible, depending on the problem to solve, like periodic decimation, cycle detection based decimation, parallel decimation of several variable at the same time, random or minimum entropy variable selection and deterministic or random sampling value assignment.
(3) We implement and evaluate these combinations of decimation policies on classical constraint optimization benchmarks (graph coloring, random graphs and Ising model), against state-of-the-art methods like standard MaxSum and MaxSum_AD_VP. More specifically, we analyze the impacts of fast decimation, cycle detection-based decimation and parallel decimation. On the one hand, we observe that some decimation policies significantly outperform MaxSum, both in terms of quality and communication cost. On the other hand, other decimation policies show to be very competitive with respect to MaxSum_AD_VP.

The rest of the paper is organized as follows. Section 2 expounds some background on DCOP and the decimation algorithm from which our algorithm DECIMAXSUM is inspired. Section 3 defines the general framework of DECIMAXSUM, and several examples of decimation policies. Section 4 presents results and analyses of experimenting DECIMAXSUM, with different combinations of

decimation policies, against MaxSum and MaxSum_AD_VP. Finally, Section 5 concludes this paper with some perspectives.

2 Background

We start this section by introducing the DCOP framework in Section 2.1. Since our purpose is to extend classical MaxSum using decimation mechanisms, Section 2.2 briefly presents belief-propagation and its DCOP derivation MaxSum, before providing in Section 2.3 a summary of decimation-related methods.

2.1 Distributed constraint optimization problems

One way to model the coordination problem between intelligent agents is to map it into the distributed constraint optimization framework.

DEFINITION 2.1 (DCOP).
A discrete *Distributed Constraint Optimization Problem* (or DCOP) is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, where: $\mathcal{A} = \{a_1, \ldots, a_N\}$ is a set of agents; $\mathcal{X} = \{x_1, \ldots, x_N\}$ are variables (each one owned by one of the agents); $\mathcal{D} = \{\mathcal{D}_{x_1}, \ldots, \mathcal{D}_{x_N}\}$ is a set of finite domains, such that variable x_i takes values in $\mathcal{D}_{x_i} = \{v_1, \ldots, v_{k_i}\}$; $\mathcal{C} = \{c_1, \ldots, c_M\}$ is a set of soft constraints, where each $c_m : \mathcal{D}_{c_m} \rightarrow \mathbb{R} \cup \{+\infty\}$ (where $\mathcal{D}_{c_m}$ is the cartesian product of the domains of the variables in the scope of c_m, namely $\mathcal{X}_{c_m} \subseteq \mathcal{X}$) is a cost function (a constraint is initially known only to the agents involved); A *solution* to the DCOP is an assignment $\mathbf{x} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ to all variables that minimizes the overall sum of costs[1] : $\sum_{m=1}^{M} c_m(\mathbf{x}_{c_m})$, where $\mathbf{x}_{c_m}$ is the projection of $\mathbf{x}$ to the scope of c_m.

As highlighted in [1, 4], DCOPs have been widely studied and applied in many reference domains, and have many interesting features: (i) strong focus on decentralized approaches where agents negotiate a joint solution through local message exchange; (ii) 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 to suboptimal algorithms.

In general, a DCOP can be represented as a factor graph: an undirected bipartite graph in which vertices represent variables and constraints (called factors), and an edge exists between a variable and a constraint if the variable is in the scope of the constraint, as pictured in Figure 1b. When the graph representing the DCOP contains at least a cycle, we call it a *cyclic* DCOP; otherwise, it is *acyclic*.

DEFINITION 2.2 (Factor Graph).
A factor graph of a DCOP as in Def. 2.1, is a bipartite graph $FG = \langle \mathcal{X}, \mathcal{C}, E \rangle$, where the set of variable vertices corresponds to the set of variables $\mathcal{X}$, the set of factor vertices corresponds to the set constraints $\mathcal{C}$ and the set of edges is $E = \{\{x_i, c_j\} | \, x_i \in \mathcal{X}_{c_j}\}$.

A large literature exists on algorithms for solving DCOPs which fall into two categories. On the one hand, *complete algorithms* like ADOPT and its extensions [13], or inference algorithms

[1] Note that the notion of cost can be replaced by the notion of utility. In this case, solving a DCOP is a maximization problem of the overall sum of utilities.

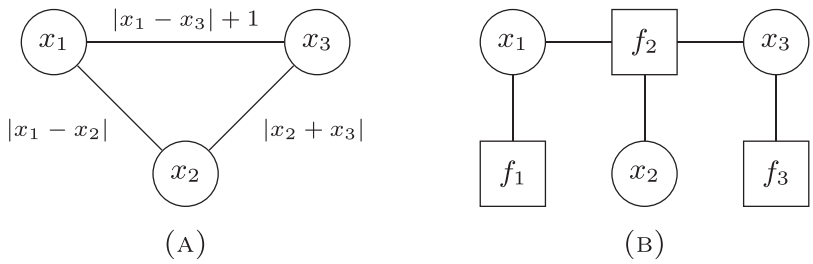

FIGURE 1. Usual DCOP representations: (a) cyclic constraint graph for a binary DCOP and (b) acyclic factor graph for a n-ary DCOP (unary and ternary constraints).

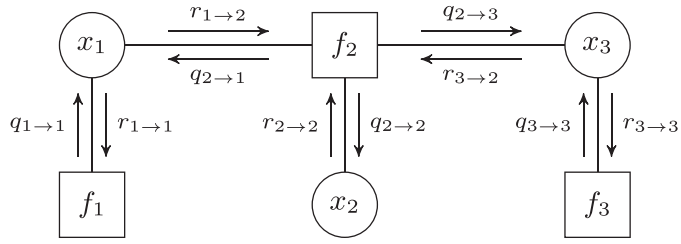

FIGURE 2. BP message flow in a sample factor graph.

like DPOP [18] or ActionGDL [23], are optimal, but mainly suffer from expensive memory (e.g. exponential for DPOP) or communication (e.g. exponential for ADOPT) load –which we may not be able to afford in a constrained infrastructure, like in sensor networks. On the other hand, *incomplete algorithms* like MaxSum [3] or MGM [12] have the great advantage of being fast with a limited memory print and communication load, but losing optimality in some settings –e.g. MaxSum is optimal on acyclic DCOPs and may achieve good quality guarantee on some settings.

The aforementioned algorithms mainly exploit the fact that an agent's utility (or constraint's cost) depends only on a subset of other agents' decision variables, and that the global utility function (or cost function) is a sum of each agent's utility (constraint's cost). In this paper, we are especially interested in belief-propagation-based algorithms, like MaxSum, where the notion of marginal values describes the dependency of the global utility function on variables.

2.2 From belief-propagation to MaxSum

Belief-propagation (BP), i.e. sum–product message passing method, is a potentially distributed algorithm for performing inference on graphical models, and can operate on factor graphs representing a product of M factors [11]:

$$F(\mathbf{x}) = \prod_{m=1}^{M} f_m(\mathbf{x}_{f_m}) \tag{1}$$

For instance, factor graph in Figure 2 represents the factorization $f_1(\mathbf{x}_1)f_2(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)f_3(\mathbf{x}_3)$.

The sum-product algorithm provides an efficient local message passing procedure to compute the marginal functions of all variables simultaneously. The marginal function, z_n describes the total dependency of the global function F on variable x_n:

$$z_n(\mathbf{x}_n) = \sum_{\mathbf{x}_{-n} \in \mathcal{D}_{-n}} F(\mathbf{x}_{-n}, \mathbf{x}_n) \tag{2}$$

where $\mathcal{D}_{-n}$ is the cartesian product of the domains of every variable other than x_n.

BP operates iteratively, propagating messages $m_{i \to j}$ (tables associating marginals to each value of variables) along the edges of the factor graph, as illustrated in Figure 2. These messages depend on the type of the emitter:

(a) *from variables to functions*:

$$q_{n \to m}(\mathbf{x}_n) = \prod_{m' \in \mathcal{V}(n) \setminus m} r_{m' \to n}(\mathbf{x}_n) \tag{3}$$

where $\mathcal{V}(n)$ is the set of factors connected to variable x_n, and (b) *from functions to variables*:

$$r_{m \to n}(\mathbf{x}_n) = \sum_{\mathbf{y} \in \mathcal{D}_{f_m \setminus n}} \left(f_m(\mathbf{y}, \mathbf{x}_n) \prod_{n' \in \mathcal{F}(m) \setminus n} q_{n' \to m}(\mathbf{y}_{n'}) \right) \tag{4}$$

where $\mathcal{F}(m)$ is the set of indexes of variables connected to function f_m and $\mathcal{D}_{f_m \setminus n}$ is the cartesian product of the domains of every variable in the scope of f_m other than x_n.

When the factor graph is a tree, BP algorithm computes the exact marginals and converges in a finite number a steps depending on the diameter of the graph [11]. Max-product is an alternative version of sum-product which computes the maximum value instead of the sum in Equation 4:

$$r_{m \to n}(\mathbf{x}_n) = \max_{\mathbf{y} \in \mathcal{D}_{f_m \setminus n}} \left(f_m(\mathbf{y}, \mathbf{x}_n) \prod_{n' \in \mathcal{F}(m) \setminus n} q_{n' \to m}(\mathbf{y}_{n'}) \right) \tag{5}$$

and uses a scaler α_{nm}, such that $\sum_{\mathbf{x}_n} q_{n \to m}(\mathbf{x}_n) = 1$ to cope with cyclic graphs in Equation 3:

$$q_{n \to m}(\mathbf{x}_n) = \alpha_{nm} \prod_{m' \in \mathcal{V}(n) \setminus m} r_{m' \to n}(\mathbf{x}_n) \tag{6}$$

Thus, max-product computes $z_n(\mathbf{x}_n)$ as:

$$z_n(\mathbf{x}_n) = \max_{\mathbf{x}_{-n} \in \mathcal{D}_{-n}} F(\mathbf{x}_{-n}, \mathbf{x}_n) \tag{7}$$

and the marginal value for variable x_n is $\mathbf{x}_n^* = \arg\max_{\mathbf{x}_n \in \mathcal{D}_n} z_n(\mathbf{x}_n)$.

Built as a derivative of max-product, MaxSum is an incomplete algorithm to solve DCOP [3]. The main evolution is the way messages are assessed, to pass from product to sum operator through logarithmic translation. In MaxSum, message from variable to factor are assessed as

$$Q_{n \to m}(\mathbf{x}_n) = \alpha_{nm} + \sum_{m' \in \mathcal{V}(n) \setminus m} R_{m' \to n}(\mathbf{x}_n) \tag{8}$$

and messages from factor to variable are defined by

$$R_{m\to n}(\mathbf{x}_n) = \max_{\mathbf{y}\in\mathcal{D}_{fm}\setminus n}\left(c_m(\mathbf{y},\mathbf{x}_n)\sum_{n'\in\mathcal{F}(m)\setminus n} Q_{n'\to m}(\mathbf{y}_{n'})\right) \tag{9}$$

And as a consequence, MaxSum computes an assignment $\mathbf{x}^*$ that maximizes the DCOP objective. Depending on the DCOP to solve, MaxSum may be used with two different termination rules: (i) continue until convergence (no more exchanged messages, because when a variable or a factor receives twice the same message from the same emitter it does not propagate it); (ii) propagate message for a fixed number of iterations per agent. MaxSum is optimal on tree-shaped factor graphs and still performs well on cyclic settings. But there exist problems for which MaxSum does not converge or converge to a sub-optimal state. In fact, on cyclic settings [3] identify the following behaviors: (i) agents converge to fixed states that represent either the optimal solution, or a solution close to the optimal, and the propagation of messages ceases; (ii) agents converge as above, but the messages continue to change slightly at each update, and thus continue to be propagated around the network; (iii) neither the agents' preferred states, nor the messages converge and both display cyclic behavior.

As to improve MaxSum performance on cyclic graphs, [25] proposed two extensions to MaxSum: (i) MaxSum_AD which operates MaxSum on a directed acyclic graph built from the factor graph and alternates direction at a fixed rate (parameter k of the algorithm); (ii) MaxSum_AD_VP which operates MaxSum_AD and propagates current values of variables when sending MaxSum messages so that factors receiving the value only consider this value instead of the whole domain of the variable. These two extensions, especially the second one, greatly improve the quality of the solution: MaxSum_AD_VP found solutions that approximate the optimal solution by a factor of roughly 1.1 on average, on random graphs and graph coloring problems. However, the study does not consider the number of exchanged messages, or the time required to converge and terminate MaxSum_AD_VP.

2.3 BP-guided decimation

In this paper, we propose to take inspiration from work done in computational physics [16], as to cope with cyclicity in DCOP. Notably, [8] introduced the notion of decimation in constraint satisfaction, especially k-satisfiability, where variables are binary, $\mathcal{D}_i = \{0, 1\}$, and each constraint requires k of the variables to be different from a specific k-tuple. Authors proposed a class of algorithms, namely *message passing-guided decimation procedure*, which consists in iterating the following steps: (1) run a message passing algorithm, like BP; (2) use the result to choose a variable index i and a value $\mathbf{x}_i^*$ for the corresponding variable; (3) replace the constraint satisfaction problem with the one obtained by fixing x_i to $\mathbf{x}_i^*$. The BP-guided decimation procedure is shown in Algorithm 1, and its performance is analysed in [14, 16].

BP-guided decimation operates on the factor graph representing the k-satisfiability problem to solve. At each step, the variable to decimate is randomly chosen among the remaining variables. The chosen variable x_i is assigned a value that is determined by random sampling from its marginal z_i. After decimation, the factor graph is simplified: some edges are no more relevant, and factors can be sliced (columns corresponding to removed variables are deleted).

Some comments can be made on this approach. First, *relying on marginal values* is a key feature, and it is the core of the "BP-guided" nature of this method. Marginal values are exploited to prune the factor graph. Second, while in the seminal work of [14], this procedure is used to solve satisfiability problems, the approach can be adapted to *cope with optimization problems*.

For instance, the inference library libDAI proposes an implementation of decimation for discrete approximate inference in graphical models [15], which was amongst the three winners of the UAI 2010 Approximate Inference Challenge.[2]

Algorithm 1: The BP-guided decimation algorithm from [14]

Data: A factor graph representing a k-satisfiability problem
Result: A feasible assignment $\mathbf{x}^*$ or `FAIL`

initialize BP messages
$\mathcal{U} \leftarrow \emptyset$
for $t = 1, \ldots, n$ **do**
  run BP until the stopping criterion is met
  choose $x_i \in \mathcal{X} \setminus \mathcal{U}$ uniformly at random
  compute the BP marginal z_i
  choose $\mathbf{x}_i^*$ distributed according to z_i
  fix $x_i = \mathbf{x}_i^*$
  $\mathcal{U} \leftarrow \mathcal{U} \cup \{x_i\}$
  simplify the factor graph
  if a contradiction is found, return `FAIL`
return $\mathbf{x}^*$

3 DECIMAXSUM: extending MaxSum with decimation

While mainly designed as a centralized algorithm and studied on k-SAT problems, BP-guided decimation can be adapted to solve DCOPs. Here we detail the core contribution of this paper, namely the DECIMAXSUM framework and its components.

3.1 Principles

The main idea is to extend the BP-guided decimation algorithm from [14] in order to define a more general framework, in which other BP-based existing algorithms could fit. The main focus of our framework is *decimation*, which means assigning a value to a variable to remove it from the problem. As the name suggests, there is no way back when a variable has been decimated –unlike search algorithms, where variable assignments can be revised following a backtrack, for instance. Therefore, triggering decimation has a major impact. This is why our framework is mainly based on answering three questions: (i) when to trigger decimation, (ii) which variable(s) to decimate and (iii) which value to assign to the decimated variable(s). Several criteria can be defined for answering each question, and DECIMAXSUM specifies such criteria as *decimation policies*, which are fundamental parameters of the decimation procedure.

We note by FG^t the current state at time t of a factor graph $FG = \langle \mathcal{X}, \mathcal{C}, E \rangle$, that is the composition of all the current states of the data structures used by the BP-based algorithm to operate on the related factor graph, including the marginal values z_i, the messages $m_{i \to j}$ and the set of decimated variables $\mathcal{U}$. We can consider that for a given problem, many factor graph states may exist. We denote by $\mathfrak{S}$ the set of possible factor graph states.

[2] http://www.cs.huji.ac.il/project/UAI10/

DEFINITION 3.1 (Decimation Policy).
A *decimation policy* is a tuple $\pi = \langle \Theta, \Omega, \Lambda \rangle$ where:

- $\Theta : \mathfrak{S} \to \{0, 1\}$ is the condition to trigger the decimation process, namely the *trigger policy*,
- $\Omega = \langle \Phi, \Upsilon \rangle$ stands for the *variable selection policy*, where $\Phi : \mathfrak{S} \to 2^{\mathcal{X}}$ is a *filter policy* which selects some candidate variables to decimate, and $\Upsilon : \mathcal{X} \times \mathfrak{S} \to \{0, 1\}$ is the condition to perform decimation on a variable, namely *perform policy*,
- $\Lambda : \mathcal{X} \times \mathfrak{S} \to \mathcal{D}_{\mathcal{X}}$ is the *assignment policy*, which assigns a value to a given variable.

A wide range of decimation-based algorithms can be represented into this framework by defining particular decimation policies. For instance, one may consider a DECIMAXSUM instance, which (i) triggers decimation once BP has converged, (ii) chooses randomly a variable to decimate within the whole set of non-decimated variables and (iii) samples the value of the decimated variable depending on its marginal values (used as probability distribution). By doing so, we obtain the classical BP-guided decimation algorithm from [14].

3.2 DECIMAXSUM *as an algorithm*

We can summarize the DECIMAXSUM framework using Algorithm 2. It is a reformulation of BP-guided decimation, parameterized with a decimation policy. Here decimation is not necessarily triggered at convergence (or time limit) of BP. Criterion Θ may rely on other components of the state of the factor graph. Contrary to classical BP-guided decimation, there may be several variables to decimate simultaneously (like in some variants of DSA [24] or MGM [12]) and those variables can be chosen in an informed manner (and not randomly), using criterion Υ. Values assigned to decimated variables are not necessarily chosen stochastically, but they are assigned using function Λ, which can be deterministic (still depending on the current state of the FG). Since, here we are not in the k-satisfiability case, but in an optimization case, there is no failure (only suboptimality), contrary to Algorithm 1.

Algorithm 2: The DECIMAXSUM framework as an algorithm

Data: A factor graph $FG = \langle \mathcal{X}, \mathcal{C}, E \rangle$, a decimation policy $\pi = \langle \Theta, \Phi, \Upsilon, \Lambda \rangle$
Result: A feasible assignment $\mathbf{x}^*$

```
1  initialize BP messages
2  U ← ∅
3  initialize x* to a void assignment
4  while U ≠ X or maximum number of iterations reached do
5      run BP until decimation triggers, i.e. Θ(FG^t) = 1          // Sect. 3.3
6      choose vars to decimate, X' = {x_i ∈ Φ(FG^t) | Υ(x_i, FG^t)}   // Sect. 3.4
7      for x_i ∈ X' do                                             // Sect. 3.5
8          x*_i ← Λ(x_i, FG^t)
9          U ← U ∪ {x_i}
10         simplify FG^t                          // remove variables, slice factors
11 if maximum number of iterations reached then
12     complete x* by assigning values to variables in X \ U as in line 8
13 return x*
```

The rest of the section details and illustrates each of these decimation policy components with some examples.

3.3 Triggering decimation (criterion)

In the original approach proposed by [14], decimation is triggered once BP has converged. In a distributed setting and diffusing algorithms like BP, this can be implemented using termination detection techniques.

$$\Theta_{\texttt{converge}}(s) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } s \text{ is } \textit{quiescent} \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

This trigger consists of detecting the *quiescence* of the current state of the factor graph. This means no process is enabled to perform any locally controlled action and there are no messages in the channels [10].

In some settings with strong time or computation constraints (e.g. sensor networks [3], Internet-of-Things [21]), waiting for convergence is not affordable. Indeed, BP may generate a lot of messages. Therefore, we may consider decimating before convergence at a fixed rate (e.g. each 10 iterations), or by sharing a fixed iteration budget amongst the variables (e.g. each 1000 iterations divided by the number of variables). We can even consider a varying decimation speed (e.g. faster at the beginning, and lower at the end, as observed in neural circuits in the brain [17]). Formally, the condition to trigger is defined as follows,

$$\Theta_{\nu-\texttt{periodic}}(s) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } \textit{time}(s) \mod \nu = 0 \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

where ν is a predefined decimation frequency.

Finally, another approach is to trigger decimation once a cycle in the FG is detected. Indeed, decimation is used here to cope with loops, so decimating variables that might potentially break loops, seems a good approach. Detecting cycles in the FG can be implemented during BP by adding some meta-data to the BP messages, as done in the DFS-tree construction phase of algorithms like DPOP or ADOPT. When an agent sends a message, it attaches a token with its identifier and the identifier of the destination. Tokens are aggregated all along the message propagations. When a variables receives a message with a token containing its identifier, but coming from another agents than the destination of the token, it signifies a cycle has been detected. Once, a variable is decimated, all the tokens containing its identifier are removed (for communication load concern). Formally, the condition to trigger decimation is:

$$\Theta_{\texttt{cycle}}(s) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } \exists x_i \in \mathcal{X}, |\textit{cycle}(x_i)| > 1 \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

where $\textit{cycle}(x_i)$ is the set of agents in the same first loop that x_i just discovered.

3.4 Deciding the subset of variables to decimate (and criteria)

Now that our algorithm is capable of deciding when to trigger decimation, the following question is "which variables to decimate?" In [14], the choice of a variable is random (in a uniform manner), while [15] selects the variable with the minimum entropy over its marginal values (the most *determined* variable). Obviously, exploiting the marginal values built throughout propagation is a good idea.

3.4.1 Choosing the candidate variables to decimate from. Both [14] and [15] select the only variable to decimate amongst the whole set of non-decimated variables (cf. line 5 in Algorithm 1). Here, the criterion is specified as follows:

$$\Phi_{\texttt{all}}(s) \stackrel{\text{def}}{=} \mathcal{X} \setminus \mathcal{U} \tag{13}$$

An interesting alternative approach is to exploit cycle detection to filter out the variables to decimate, when using criterion $\Theta_{\texttt{cycle}}$ from Eq. 12. Formally, the following filter policy defines this approach:

$$\Phi_{\texttt{cycle}}(s) \stackrel{\text{def}}{=} \{x \in \mathcal{X} \setminus \mathcal{U} \mid |cycle(x)| > 1\} \tag{14}$$

3.4.2 Criteria to decide the variable(s) to decimate. Now, we have to specify the criterion to decide the candidate variables to decimate. In [14], the criterion is fully random: it does not depend on the current state of variables. Our criterion will randomly select k variables as follows:

$$\Upsilon_{\texttt{rand\_k}}(x_i, s) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } x_i \in \mathcal{R}_k \\ 0, & \text{otherwise} \end{cases} \tag{15}$$

where $\mathcal{R}_k$ is a set of k variables selected at random out of the variables picked up by the filter policy.

In [15], the variable with the lowest entropy over its marginal values is selected. This means that the variable for which marginal values seem to be the most informed ones, in a Shannon's Information Theory sense, is chosen:

$$\Upsilon_{\texttt{min\_entropy\_k}}(x_i, s) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } x_i \in \mathcal{M}_k \\ 0, & \text{otherwise} \end{cases} \tag{16}$$

with $\mathcal{M}_k$ stands for the set of k variables with smaller marginal entropy value out of the variables picked up by the filter policy.

3.5 Deciding the values to assign to decimated variables (criterion)

Once the variables to decimate have been selected, the question is: "which values shall we assign to decimated variables?" Usually, in BP-based algorithms, the simplest variable assignment mechanism, after propagation, is based on selecting those values with maximal marginal value (or utility). In fact, [15] uses such deterministic criterion for inference:

$$\Lambda_{\texttt{deterministic}}(x_i, s) \stackrel{\text{def}}{=} \arg\max_{\mathbf{d} \in \mathcal{D}_i} z_i(\mathbf{d}) \tag{17}$$

While the policy above is deterministic, in [14] the choice of the value is random by using marginal values as a probability distribution:

$$\Lambda_{\texttt{sampling}}(x_i, s) \stackrel{\text{def}}{=} sample(z_i) \tag{18}$$

Once again, these are only two examples of policies exploiting BP.

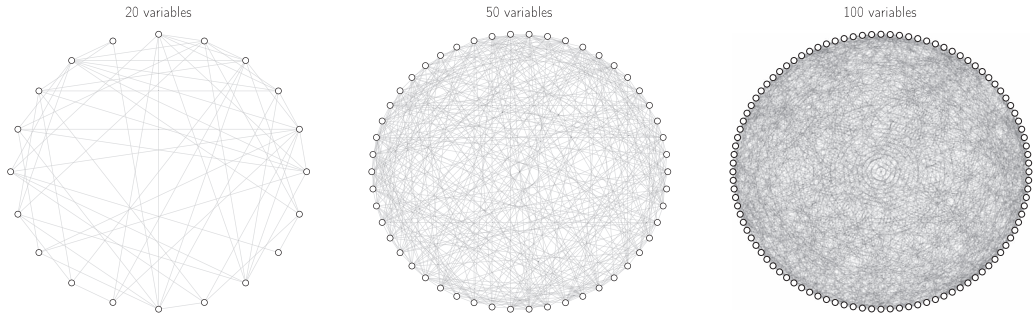

FIGURE 3. Sample constraint graph structures used for graph coloring and random graph problems.

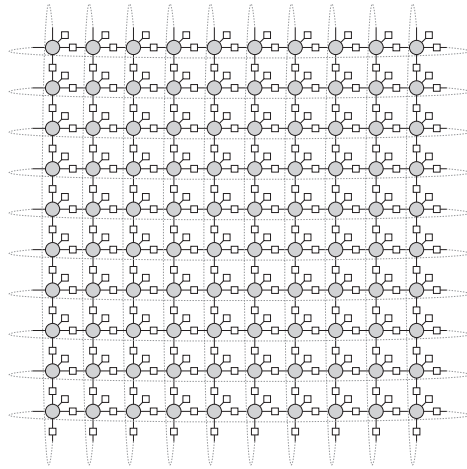

FIGURE 4. Ising model factor graph, i.e. a 10×10 toroidal grid: x_i variables as circles, unary (r_i) and binary (r_{ij}) factors as squares.

4 Experiments

This section evaluates the performance of different combinations of decimation policies in DECIMAXSUM on a classical optimization model against classical Max-Sum [3] and its extension Max-Sum_AD_VP [25]. First the different benchmark problems used to evaluate the performances of decimation policies are described in Section 4.1. Then, the implemented algorithms and decimation policies are listed in Section 4.2. Finally, the results of these experiments are presented and analyzed in Section 4.3.

4.1 Benchmark problems

This study evaluates DECIMAXSUM performance to solve three different kinds of optimization problems: 1. graph coloring problems, 2. random graphs problems, 3. Ising models.

4.1.1 Graph Coloring Graph coloring is a classical benchmark in constraint satisfaction and optimization. Here are considered coloring randomly structured graphs with 4 colors, density equals

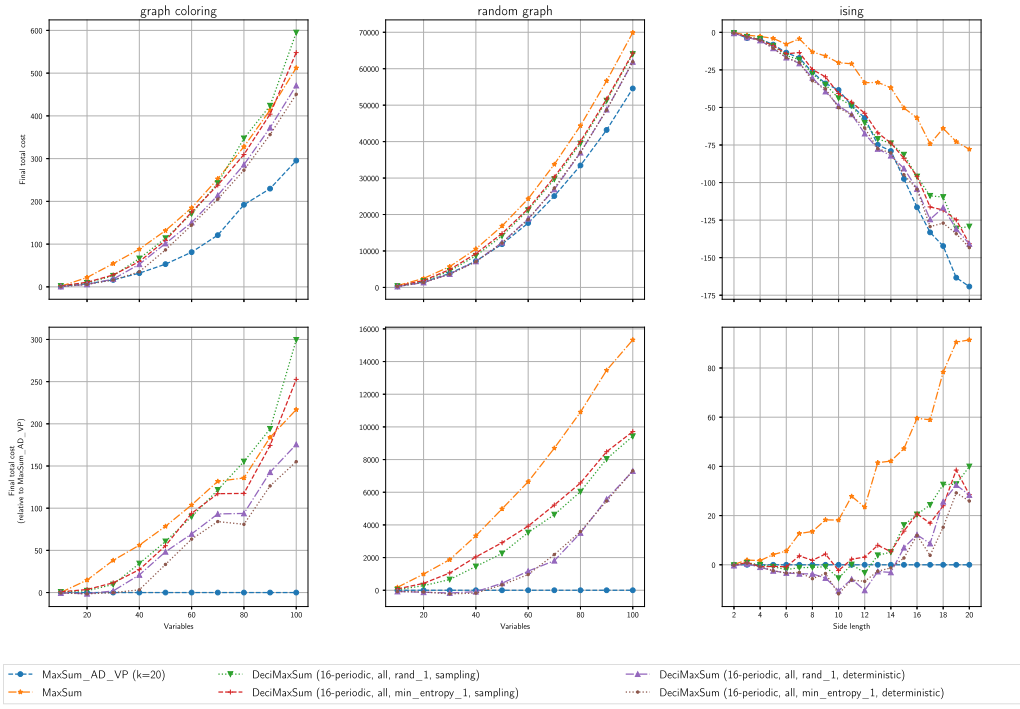

FIGURE 5. Absolute final total cost (top) and relative to MaxSum_AD_VP (bottom), for DECIMAX-SUM variable and value selection policies with a periodic decimation trigger.

to 0.3 (probability for each pair of variables to be connected in the graph), tightness equals to 0.5 (the fraction of the variable assignments that are infeasible) and with a number of variables in [10, 100]. All constraints are binary (between two variables) and the cost of violating a constraint is set to 1, and 0 otherwise. Thus, the problem is encoded as an minimization problem. Figure 3 illustrates the structure of such graphs, which are dense and contain many cycles.

4.1.2 Random Graphs Built as our graph coloring graphs (density 0.3), random graphs are more general constraint optimization problems with larger domains (size 10) and binary soft constraints with random uniform cost in $U[0, 100]$ for each pair of values between constrained variables. Here again we consider problems of size in [10, 100], which are pure minimization problems.

4.1.3 Ising Model Since we are interested in evaluating the different algorithms in the presence of strong dependencies among the values of variables, they are also evaluated on the Ising model, which is a widely used benchmark in statistical physics [7]. The same settings than [22] are used. Constraint graphs are rectangular grids where each binary variable x_i is connected to its four closer neighbors (with toroidal links that connect opposite sides of the grid) and is constrained by a unary cost r_i. The weight of each binary constraint r_{ij} is determined by first sampling a value κ_{ij} from a

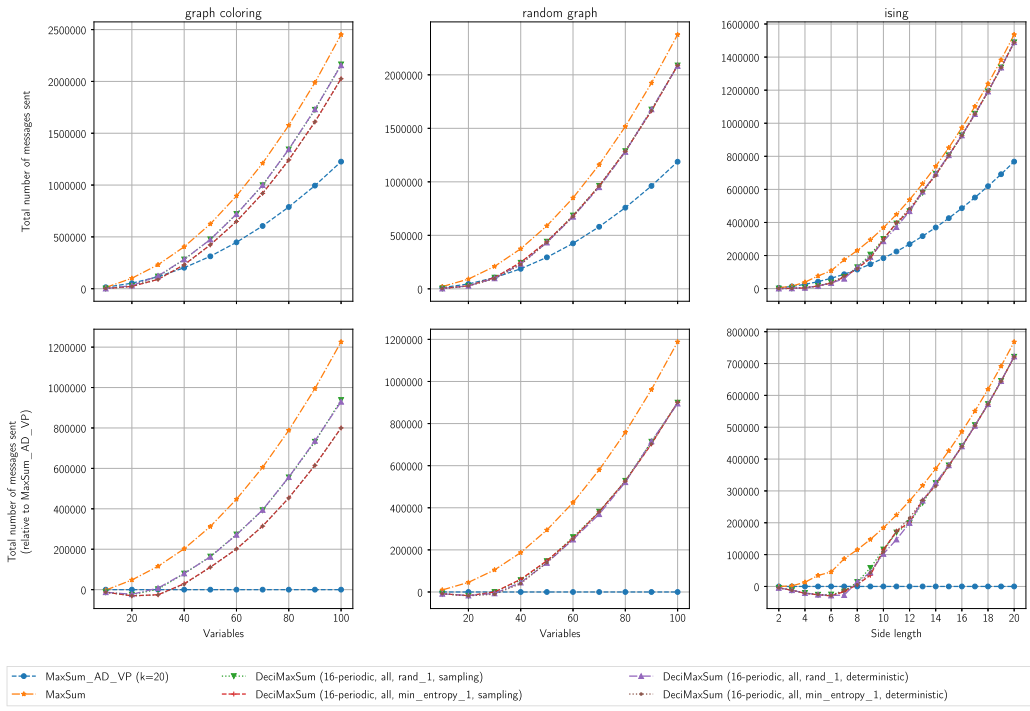

FIGURE 6. Absolute final number of messages (top) and relative to MaxSum_AD_VP (bottom), for DECIMAXSUM variable and value selection policies with a periodic decimation trigger.

uniform distribution $U[-\beta, \beta]$ and then setting

$$r_{ij}(x_i, x_j) = \begin{cases} \kappa_{ij} & \text{if } x_i = x_j \\ -\kappa_{ij} & \text{otherwise.} \end{cases}$$

The β parameter controls the average strength of interactions. In the following experiments β is set to 1.6. The weight for each unary constraint r_i is determined by sampling κ_i from a uniform distribution $U[-0.05, 0.05]$ and then setting $r_i(0) = \kappa_i$ and $r_i(1) = -\kappa_i$. Here, generated problems have side sizes varying in $[2, 20]$ (i.e. 4 to 400 variables).

4.2 Evaluated algorithms and decimation policies

The following state-of-the-art solution methods have been implemented: MaxSum, as defined in [3], with a maximum number of iterations set to 400, MaxSum_AD_VP, as defined in [25], with $k = 20$ (i.e. the edge directions alternate every 20 iterations) and a maximum number of iterations set to 400.

The following DECIMAXSUM policies have also been implemented and evaluated:

- with different triggers:
  - $\Theta_{\texttt{cycle}}$, i.e. each time an agent/variable detects a loop, noted cycle,
  - $\Theta_{\nu\texttt{-periodic}}$ with different frequencies, noted 4-periodic, 16-periodic, 64-periodic,

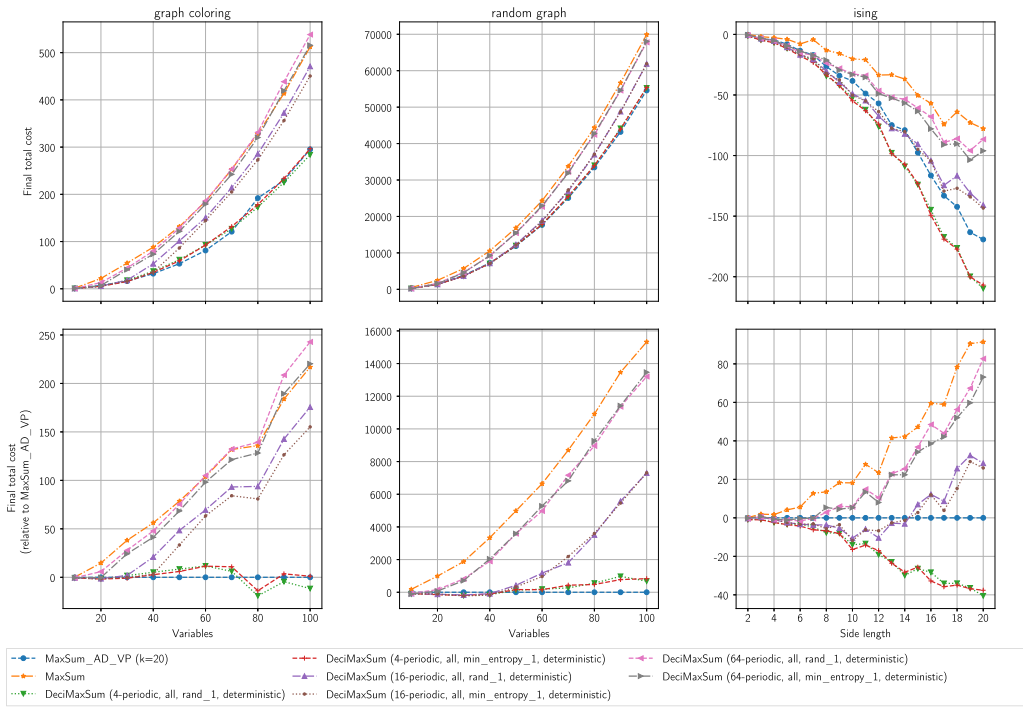

FIGURE 7. Absolute final total cost (top) and relative to MaxSum_AD_VP (bottom), for DECIMAX-SUM with periodic trigger.

- with different sets of variables as potential variables to decimate in parallel:
  - $\Phi_{\texttt{all}}$, the whole set of non decimated variables,
  - $\Phi_{\texttt{cycle}}$, the variables involved in a cycle,
- with different variable selection policies:
  - $\Upsilon_{\texttt{rand\_k}}$, randomly selects k variables out of the candidate variables,
  - $\Upsilon_{\texttt{min\_entropy\_k}}$, selects the k variables with the smaller entropy values out of the candidate variables,
- with different value selection policies:
  - $\Lambda_{\texttt{deterministic}}$, noted deterministic and
  - $\Lambda_{\texttt{sampling}}$ noted sampling.

Thus, in the following figures, decimation policies are noted as DeciMaxSum(<trigger>, <set>, <variable>, <value>). For instance, DeciMaxSum(4-periodic, all, rand_2, deterministic) means that decimation is triggered each 4 iterations and 2 randomly chosen variables are decimating in parallel by choosing their value as the one which maximizes the marginal value. Note that other state-of-the-art decimation techniques which inspired this work would be implemented like follows: Montanari's decimation, as defined in [14], is equivalent to DeciMax-Sum(converge, all, rand_1, sampling); Mooij's decimation, as defined in [15], is equivalent to

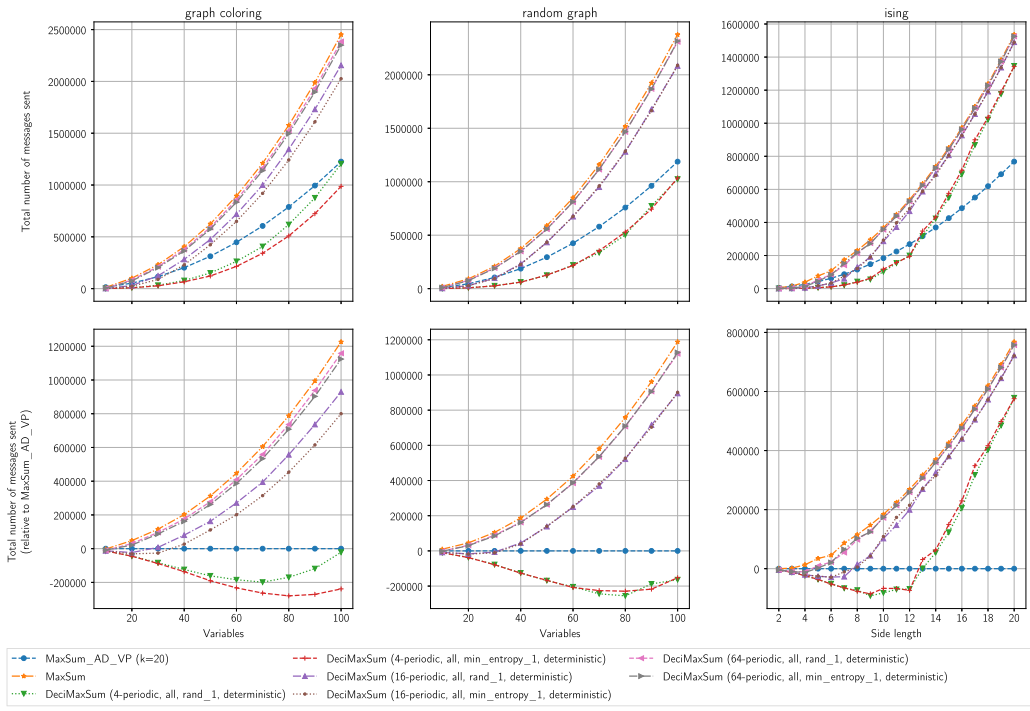

FIGURE 8. Absolute final number of messages (top) and relative to MaxSum_AD_VP (bottom), for DECIMAXSUM with periodic trigger.

DeciMaxSum(converge, all, min_entropy_1, deterministic). In the reviewed benchmarks, convergence cannot be reached in reasonable time (due to high cyclicity), so we do not compare against such techniques.

4.3 Results and analysis

The following analysis is based on two performance metrics: final total cost (i.e. the quality of the resulting solutions) and the total number of exchanged messages (to assess the communication load of each technique). For both metrics, the absolute value and the value relative to the best state-of-the-art competitor, namely MaxSum_AD_VP, are displayed. All the evaluated algorithms are allowed to run for 400 iterations. For each problem size set, 20 problems are generated and each algorithm is ran 3 times on each problem, for which the average values are plotted.

4.3.1 Impact of Variable and Value Selection Criteria Let's first analyse the impact of the Υ and Λ criteria, i.e. the policies to choose the variables to decimate and the values to assign to decimated variables. Here are only considered the case of periodic decimation (every 16 iterations) of one variable at a time, and the combinations of $\Upsilon_{\texttt{rand\_k}}$ or $\Upsilon_{\texttt{min\_entropy\_k}}$ criteria for choosing the variable, and $\Lambda_{\texttt{deterministic}}$ or $\Lambda_{\texttt{sampling}}$ for choosing the value of the decimated variables.

Figure 5 shows the final cost of solutions resulting from our decimation policies against MaxSum and MaxSum_AD_VP. On the three benchmarks, DeciMaxSum(16-periodic, all, rand_1,

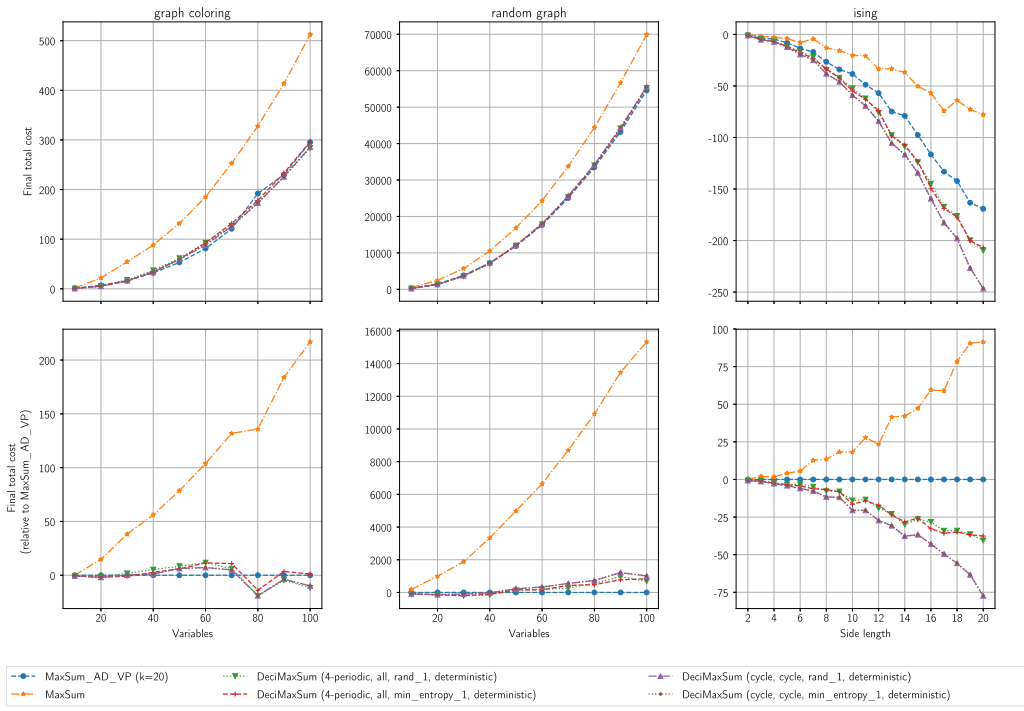

FIGURE 9. Absolute final total cost (top) and relative to MaxSum_AD_VP (bottom), for DECIMAX-SUM with cycle detection-based trigger.

deterministic) and DeciMaxSum(16-periodic, all, min_entropy_1, deterministic) are better than DeciMaxSum(16-periodic, all, rand_1, sampling) and DeciMaxSum(16-periodic, all, min_entropy_1, sampling). Note that we implemented experiments for other decimation triggers and different decimation frequencies. In the end, the following conclusion always holds on the three different benchmarks considered here: choosing the values to assign in a deterministic manner guided by the marginal values is better than choosing randomly by sampling. On the other hand, the difference between choosing the variables in a deterministic way and choosing randomly is not very significant.

With these settings, our decimation policies always outperform classical MaxSum, but only beats MaxSum_AD_VP on lower scale instances, especially on Ising models. Even looking at Figure 6 which shows the number of messages propagated until the end of the processes, decimation generates less messages than MaxSum but more than MaxSum_AD_VP (up to 92% more messages on large Ising models). This is mainly due to the fact that with a decimation frequency set to 16, for a maximum number of iterations set to 400, all the variables are not decimated before the end. So, the current decimation policies improve classical MaxSum with few decimations (only 6 with 100 variables in random structures, 25 with 400 variables in Ising models), but not enough to outperform MaxSum_AD_VP. This highlights the fact that the decimation frequency should be set depending on the time budget one can afford. The next section will thus analyze the effects of faster decimation.

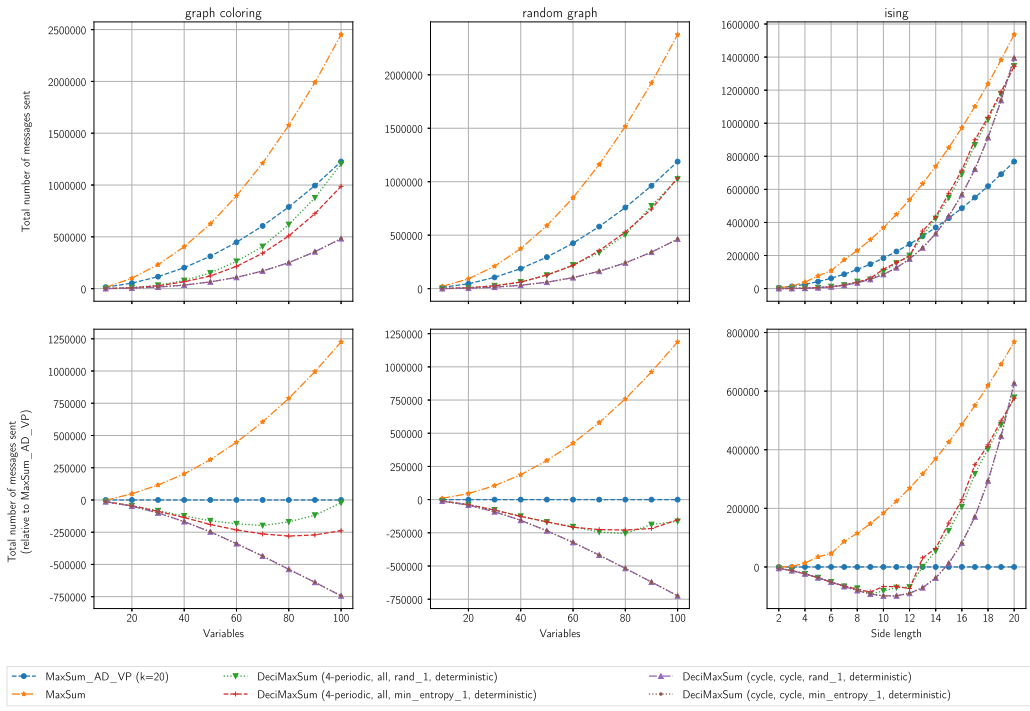

FIGURE 10. Absolute final number of messages (top) and relative to MaxSum_AD_VP (bottom), for DECIMAXSUM with cycle detection-based trigger.

4.3.2 Impact of Fast Periodic Decimation Now the fact that deterministic value selection performs better than sampling has been identified, let's increase the frequency to analyse the impact of fast decimation. Note that only DeciMaxSum(_, all, rand_1, deterministic) and DeciMaxSum(_, all, min_entropy_1, deterministic) policies will be kept for the further analyses. Figure 7 shows the quality of solutions obtained by different decimation policies with different decimation frequencies (4, 16 and 64). Clearly DeciMaxSum(4-periodic, all, rand_1, deterministic) and DeciMaxSum(4-periodic, all, min_entropy_1, deterministic) outperform low frequency decimation policies. At this rate, all the variables are decimated at the end of the process. While on graph coloring and random graphs, these fast decimation policies are equivalent to MaxSum_AD_VP (less than 4% difference), on Ising models they greatly outperform MaxSum_AD_VP (approx. 23% improvement on cost). However, when looking at Figure 8, this good quality is at the cost of a higher communication load (approx. 75% more messages), especially on larger instances (Ising models with sides larger than 13). This is mainly due to the fact that only one variable is decimated every 4 iterations, which removes 5 edges and thus poorly reduces communication load, while MaxSum_AD_VP only considers half of the total number of edges to propagate by only considering one direction per edge.

4.3.3 Impact of Cycle Detection-based Decimation Up to now, the best decimation policies identified so far rely on random variable selection over the whole set of non decimated variables. As a consequence, some decimated variables might not be part of any cycle, which might not impact

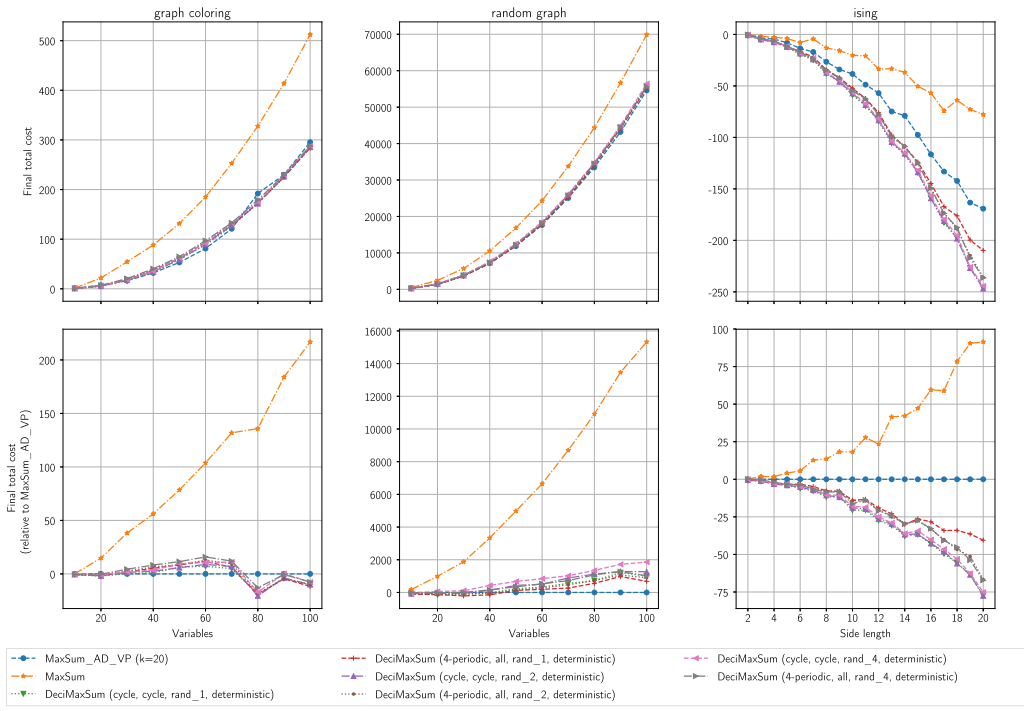

FIGURE 11. Absolute final total cost (top) and relative to MaxSum_AD_VP (bottom), for DECIMAXSUM with parallel decimation.

positively the solution method, while MaxSum_AD_VP breaks cycles by directing the edges. Let's now look at different policies where the decimation is triggered upon cycle detection.

Figure 9 shows the performance of such cycle detection-based decimation policies, namely DeciMaxSum(cycle, cycle, rand_1, deterministic) and DeciMaxSum(cycle, cycle, min_entropy_1, deterministic). Obviously, these two policies are equivalent because the first variable to detect a cycle is decimated, whatever is its marginal value. On graph coloring and random graphs, cycle detection is equivalent to fast decimation, namely DeciMaxSum(4-periodic, all, rand_1, deterministic) and DeciMaxSum(4-periodic, all, min_entropy_1, deterministic). But, on Ising model, cycle detection greatly improves the quality of solutions (approx. 19% cost improvement on 20 side length Ising models). This is mainly due to the fact that the toroidal and regular cyclic structure of Ising models allows to detect cycles very fast and relevantly. Any cycle broken on the Ising structure is equivalent to another, while in graph coloring and random graphs, cycles are not equivalent; some have more impact on message propagation than others.

Interestingly, by looking at the number of exchanged messages, as shown in Figure 10, cycle detection-based decimation generates far less messages than other decimation policies analyzed so far, except on the largest Ising models (20 side length), where it generates up to 75% more messages than MaxSum_AD_VP. Clearly, basing decimation on cycle detection is better than decimation at a fixed rate.

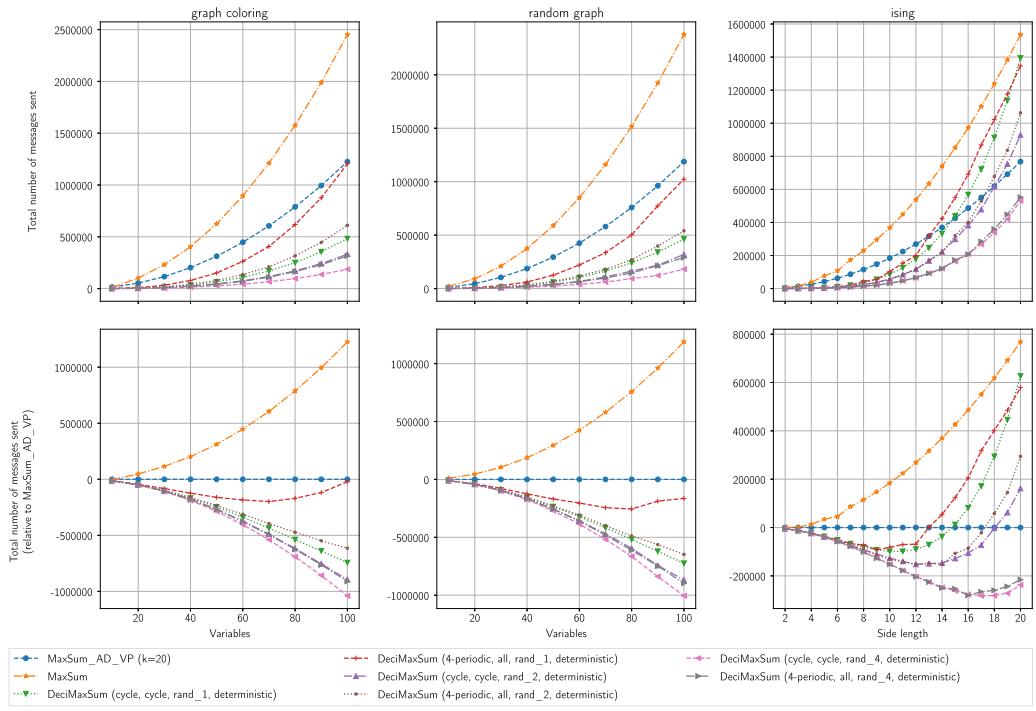

FIGURE 12. Absolute final number of messages (top) and relative to MaxSum_AD_VP (bottom), for DECIMAXSUM with parallel decimation.

4.3.4 Impact of Parallel Decimation The policies we have considered so far only decimate one variable at a time. However, it seems relevant to consider decimating several variables in parallel, especially in large settings where several non overlapping cycles may exist. In the following experiments, we look at decimation policies able to decimate 1, 2 or 4 variables amongst either the full set of non-decimated variables, or the set of variables detecting cycles.

Figure 11 shows the final solution cost for these decimation policies, against the best identified so far. Parallel decimation, especially DeciMaxSum(cycle, cycle, rand_2, deterministic) and DeciMaxSum(cycle, cycle, rand_4, deterministic) improves performances on Ising models, mostly because it is a regular and mesh model where several cycles can be detected and broken independently in different parts of the graph. Note that the diameter (i.e. the maximum path length between two nodes) of an Ising model equals its side size (so up to 20 in these experiments), while it tends to 3 for graph coloring and random graphs. Thus, two parallel decimations on randomly structured graphs may result on breaking twice the same cycle, and thus decimate some variables too early in the process. Therefore, sequential decimation performs better on small diameter graphs.

Figure 12 shows the communication load reduction obtained by parallelizing decimation. Indeed, decimating in parallel reduces faster the number of edges than decimating a single variable at a time. This leads to a reduction on the number of exchanged messages. Parallel decimation of 4 variables at the same time generates approximately 85% less messages than MaxSum_AD_VP on graph coloring and random graphs while providing equivalent solutions (maximum 4% more costly). On Ising, this

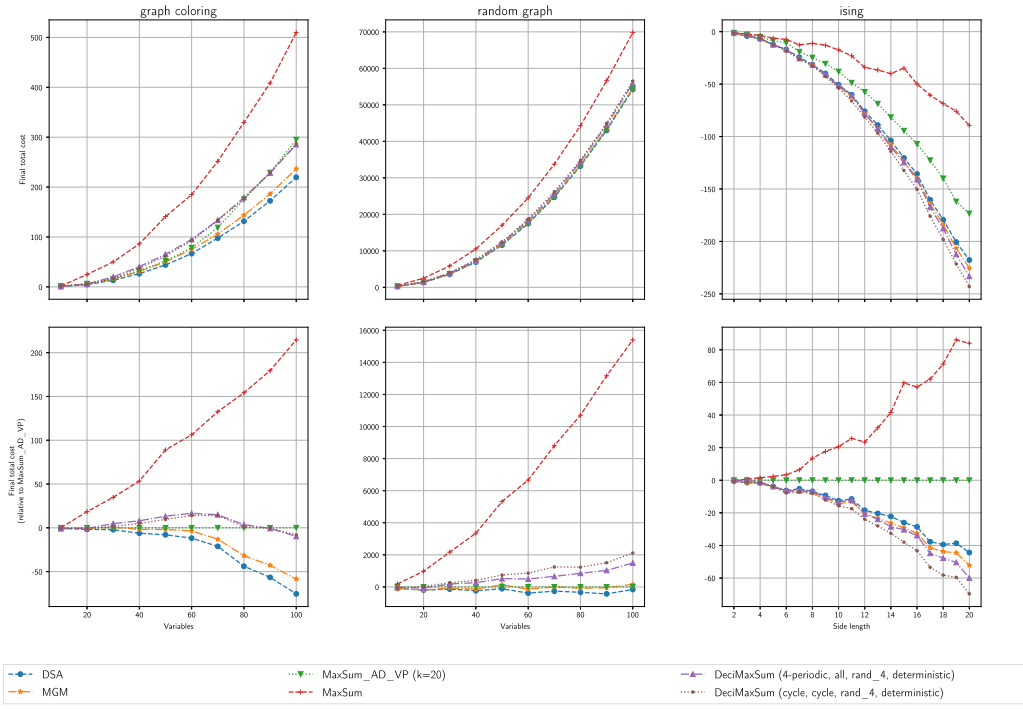

FIGURE 13. Absolute final total cost (top) and relative to MaxSum_AD_VP (bottom), for best DECIMAXSUM policies, DSA and MGM.

very same parallel decimation policies generate roughly 45% less messages than MaxSum_AD_VP for an improvement of about 45% in solution cost.

4.3.5 Comparison with other Incomplete DCOP Solution Methods. While Max-Sum is a well known solution method to solve DCOPs, other incomplete methods exists to cope with large instances. Here we focus on classical DSA [24] and MGM [12], which are not inference algorithms (thus not based on marginal values), but very fast incomplete and performant search-based algorithms. DSA is a stochastic algorithm which relies on random decisions to explore the search space. MGM is an anytime algorithm, close to DSA, but where decisions to change a variable is directly based on the minimization of the costs: for a given neighbourhood, up to k variables providing the best improvement in cost will change their value. Other solutions have been developed so far, like Distributed Large Neighborhood Search (or DLNS) [6], which iteratively splits larges problems in subproblems (neighborhoods), solves sub-problems and then merges solutions to subproblems, or Bounded MaxSum (BMS) which extends Max-Sum to provide quality bounds [20]. We didn't consider them since either the code is not publicly available at the time we ran experiment, or they have been outperformed by MaxSum_AD_VP [25]. Moreover, both DLNS and BMS focus on providing bounds which is not the intent of our study, which aims at improving MaxSum performances on cyclic settings.

We ran DSA (variant C with probability 0.5) and MGM (variant with $k = 1$ to minimize the number of messages) on the very same instances than our decimation policies. We plot in

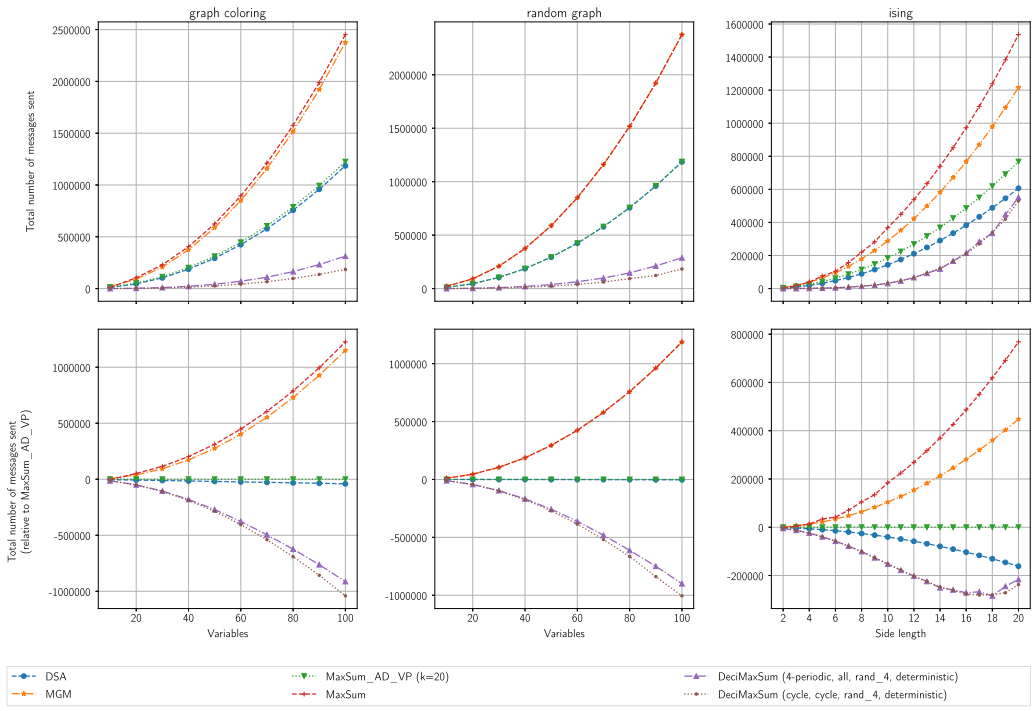

FIGURE 14. Absolute final number of messages (top) and relative to MaxSum_AD_VP (bottom), for best DECIMAXSUM policies, DSA and MGM.

Figures 13 and 14 the cost and the number of messages produced by the benchmarks solutions methods (MaxSum, MaxSum_AD_VP, DSA and MGM) and the best decimation policies identified in the previous experiments. DSA and MGM show their good performance on randomly-structured and dense large graphs (random and graph coloring), providing good quality with reasonable amount of messages. DSA provides better solutions on all problems except Ising model. However, on a more structured problem like Ising model, our best decimation policies clearly outperform both DSA and MGM (in quality and communication load) with 9% better solution quality and 54% less messages than MGM.

In the end, DSA, which is a stochastic method, performs very well on randomly-structured graph (which is an already known property), MGM has very good performances on Ising model, compared to MaxSum_AD_VP, but does not outperform parallel and cycle-detection-based decimation policies.

5 Conclusions

In this paper we have investigated how to extend MaxSum to solve distributed constraint optimization problems by taking inspiration on the decimation mechanisms used for solving k-satisfiability problems by belief-propagation. We propose a parametric method, namely DECIMAXSUM, which can be set up with different decimation policies to decide when to trigger decimation, which variables to decimate and which value to assign to decimated variables. We propose a family of policies

that can be combined to produce different versions of DECIMAXSUM: periodic decimation, cycle detection-based decimation, parallel decimation of several variables at the same time, random or minimum entropy variable selection and deterministic or random value assignment.

Our empirical results on different benchmarks (graph coloring, random graphs and Ising models) show that some combinations of decimation policies outperform classical MaxSum and are competitive with its efficient extension, MaxSum_AD_VP. DECIMAXSUM yields better quality solutions with a reasonable amount of message propagation. More precisely, single-variable cycle detection-based decimation with deterministic value selection produces equivalent quality solutions (less than 4% difference) than MaxSum_AD_VP whilst requiring lower communication load (58% less messages) on graph coloring and random graphs. Single-variable cycle detection-based decimation with deterministic value selection produces better quality solutions (19%) than MaxSum_AD_VP, but requires more messages (75% more) on the Ising model. Finally, parallel decimation is interesting on mesh graphs with large diameter like Ising model, where it improves solution quality by 47%, while reducing the number of messages by 30% on larger instances, by taking advantage of the potential independent locations in the graph. It has also been shown that parallel and cycle detection-based decimation outperform non-inference-based algorithms, DSA and MGM, on Ising models.

There are several paths to future research. First, we would like to generalize DECIMAXSUM framework to consider MaxSum_AD_VP as a particular case of decimation: iterated decimation. Second, since we shown that decimation is relevant for binary domains (like Ising models) and problems with larger domains (like for random graphs), we envision to use such techniques in other problems, close to constraint satisfaction and optimization like Max-SAT or Bayesian inference. Third, since DSA and MGM provide very good solutions on random graphs, but do not outperform parallel and cycle detection-based decimation on more structured graphs, one investigation direction we will follow is to integrate decimation into such search-based frameworks, still not being based on marginal values to decimate. Finally, we plan to apply DECIMAXSUM on real world applications, with strong cyclic nature, like the coordination of smart objects in IoT [21] or decentralized energy markets in the smart grid [2]. Since some of these application domains may require to consider privacy issues, we will tackle such issues by taking inspiration on existing work on the topic in the DCOP literature (e.g.[5, 9]) .

Funding

Research supported by project Collectiveware TIN2015-66863-C2-1-R (MINECO/FEDER). This work was also supported by project CI-SUSTAIN funded by the Spanish Ministry of Science and Innovation (PID2019-104156GB-I00).

References

[1] J. Cerquides, A. Farinelli, P. Meseguer and S. D. Ramchurn. A tutorial on optimization for multi-agent systems. *The Computer Journal*, **57**, 799–824, 2014.

[2] J. Cerquides, G. Picard and J. A. Rodríguez-Aguilar. Designing a marketplace for the trading and distribution of energy in the smart grid. In *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1285–1293. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

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

[4] F. Fioretto, E. Pontelli and W. Yeoh. Distributed constraint optimization problems and applications: A survey. *CoRR*, abs/1602.06347, 2016.
[5] T. Grinshpoun and T. Tassa. A privacy-preserving algorithm for distributed constraint optimization. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, pp. 909–916. International Foundation for Autonomous Agents and Multiagent Systems, 2014.
[6] K. Hoang, F. Fioretto, W. Yeoh, E. Pontelli and R. Zivan. A large neighboring search schema for multi-agent optimization. *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP)*, 688–706, 2018.
[7] V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **28**, 1568–1583, 2006.
[8] F. Krzakala, A. Montanari, F. Ricci-Tersenghi, G. Semerjian and L. Zdeborova. Gibbs states and the set of solutions of random constraint satisfaction problems. *Proceedings of the National Academy of Science*, **104**, 10318–10323, 2007.
[9] T. Léauté. Distributed constraint optimization: privacy guarantees and stochastic uncertainty. In *PhD thesis*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne (EPFL), 2011.
[10] N. A. Lynch. *Disributed Algorithms*. Morgan Kaufmann, 1996.
[11] D. J. C. Mackay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 1st edn., 2003.
[12] R. T. Maheswaran, J. P. Pearce and M. Tambe. Distributed algorithms for dcop: A graphical-game-based approach. In *Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems (PDCS)*, pp. 432–439, San Francisco, CA, 2004.
[13] P. J. Modi, W. Shen, M. Tambe and M. Yokoo. ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees. *Artificial Intelligence*, **161**, 149–180, 2005.
[14] A. Montanari, F. Ricci-Tersenghi and G. Semerjian. Solving constraint satisfaction problems through belief propagation-guided decimation. *CoRR*, abs/0709.1667, 2007.
[15] J. M. Mooij. libDAI: A free and open source C++ library for discrete approximate inference in graphical models. *Journal of Machine Learning Research*, **11**, 2169–2173, 2010.
[16] M. Mèzard and A. Montanari. *Information, Physics, and Computation*. Oxford University Press, 2009.
[17] S. Navlakha, A. L. Barth and Z. Bar-Joseph. Decreasing-rate pruning optimizes the construction of efficient and robust distributed networks. *PLoS Computational Biology*, **11**, 1–23, 2015.
[18] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. *International Joint Conference on Artificial Intelligence (IJCAI'05)*, 266–271, 2005.
[19] A. Rogers, A. Farinelli, R. Stranders and N. R. Jennings. Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence*, **175**, 730–759, 2011.
[20] A. Rogers, A. Farinelli, R. Stranders and N. R. Jennings. Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence*, **175**, 730–759, 2011.
[21] P. Rust, G. Picard and F. Ramparany. Using message-passing DCOP algorithms to solve energy-efficient smart environment configuration problems. In *International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI Press, 2016.
[22] M. Vinyals, M. Pujol, J. A. Rodríguez-Aguilar and J. Cerquides. Divide and coordinate: solving dcops by agreement. In *International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pp. 149–156. IFAAMAS, Canada, 2010.
[23] M. Vinyals, J. A. Rodriguez-Aguilar and J. Cerquides. Constructing a unifying theory of

dynamic programming dcop algorithms via the generalized distributive law. *Autonomous Agents and Multi-Agent Systems*, **22**, 439–464, 2010.

[24] W. Zhang, G. Wang, Z. Xing and L. Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Journal of Artificial Intelligence Research (JAIR)*, **161**, 55–87, 2005.

[25] R. Zivan, T. Parash, L. Cohen, H. Peled and S. Okamoto. Balancing exploration and exploitation in incomplete min/max-sum inference for distributed constraint optimization. *Autonomous Agents and Multi-Agent Systems*, **31**, 1165–1207, 2017.

Received 12 November 2020