



**HAL**  
open science

## Improving Max-Sum through Decimation to Solve Loopy Distributed Constraint Optimization Problems

Jesús Cerquides, Rémi Emonet, Gauthier Picard, Juan A. Rodríguez-Aguilar

► **To cite this version:**

Jesús Cerquides, Rémi Emonet, Gauthier Picard, Juan A. Rodríguez-Aguilar. Improving Max-Sum through Decimation to Solve Loopy Distributed Constraint Optimization Problems. [Research Report] Ecole des Mines de Saint Etienne (EMSE). 2017. hal-01534066

**HAL Id: hal-01534066**

**<https://hal.science/hal-01534066>**

Submitted on 7 Jun 2017

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Improving Max-Sum through Decimation to Solve Loopy Distributed Constraint Optimization Problems

J. Cerquides<sup>1</sup>, R. Emonet<sup>2</sup>, G. Picard<sup>3</sup>, and J.A. Rodriguez-Aguilar<sup>1</sup>

<sup>1</sup> IIIA-CSIC, Campus UAB, 08193 Cerdanyola, Catalonia, Spain {cerquide,jar}@iiia.csic.es

<sup>2</sup> Université de Lyon, Laboratoire Hubert Curien UMR CNRS 5516, France  
remi.emonet@univ-st-etienne.fr

<sup>3</sup> MINES Saint-Etienne, Laboratoire Hubert Curien UMR CNRS 5516, France  
picard@emse.fr

**Abstract.** In the context of solving large distributed constraint optimization problems (DCOP), belief-propagation and approximate inference algorithms are candidates of choice. However, in general, when the factor graph is very loopy (i.e. cyclic), these solution methods suffer from bad performance, due to non-convergence and many exchanged messages. As to improve performances of the Max-Sum inference algorithm when solving loopy 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 BP benchmark (the Ising model), some of these combinations of policies exhibit better performance than state-of-the-art competitors.

## 1 Introduction

In the context of multi-agent systems, distributed constraint optimization problems (DCOP) are a convenient to model 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 to assign a value (e.g. a goal, a decision), while taking into account constraints with other agents. Solving a DCOP consists in making agents communicate as to minimize the violation of these constraints. Several solution methods exist to solve such problems, from complete and optimal solutions, to approximate ones. When dealing with larger scales (thousands of variables), approximate methods are solutions of choice. Indeed, complete methods, like ADOPT or DPOP, suffer exponential computation and/or communication cost in general settings [10,15]. As a consequence, in some large settings, approximate methods are better candidates, as evidenced by the extensive literature on the subject (see [1] for a complete review). One major difficulty for approximate method to solve DCOP is the presence of cycles in the constraint graph (or factor graph). Among the aforementioned methods, inference-based ones, like Max-Sum [3] and its extensions like [16], have demonstrated good performance even on loopy settings. However, there exists 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 with bad quality.

One original approach to cope with loopy 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 [13]. The decimation is processed regularly after the convergence of a classical belief-propagation procedure. In [11], decimation has been used in the constraint satisfaction framework, for solving centralized  $k$ -satisfiability problems [11]. Inspired by this concept, we propose a general framework for applying decimation in the DCOP setting. Other works proposed Max-Sum\_AD\_VP as to improve Max-Sum performance on loopy graphs [20]. The idea is to perform the inference mechanism through an overlay directed

acyclic graph, to remove loops, and to alternating the direction of edges at a fixed frequency as to improve the sub-optimal solution found with the previous direction. One mechanism within one of these extensions, namely value propagation, can be viewed as a temporary decimation.

Against this background, the main goal of this paper is to propose a general framework for installing decimation in Max-Sum for solving DCOP. More precisely, we make the following contributions:

1. We propose a parametric solution method, namely DECIMAXSUM, to implement decimation in Max-Sum. It takes three fundamental parameters for decimation: (i) a policy stating when to trigger decimation, (ii) a policy stating which variables to decimate, and (iii) 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 library 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.
3. We implement and evaluate some of these combinations of decimation policies on classical DCOP benchmarks (meeting scheduling and Ising models), against state-of-the-art methods like standard Max-Sum and Max-Sum\_AD\_VP.

The rest of the paper is organized as follows. Section 2 expounds some background on DCOP and expounds 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 Max-Sum and Max-Sum\_AD\_VP. Finally, Section 5 concludes this paper with some perspectives.

## 2 Background

This section expounds the DCOP framework and some related belief-propagation algorithms from the literature are discussed concerning the mechanisms to handle cycles in constraint graphs.

### 2.1 Distributed Constraint Optimization Problems

One way to model the coordination problem between smart objects is to formalize the problem as a distributed constraint optimization problem.

**Definition 1 (DCOP).** A discrete Distributed Constraint Optimization Problem (or DCOP) is a tuple  $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C}, \mu \rangle$ , where:  $\mathcal{A} = \{a_1, \dots, a_{|\mathcal{A}|}\}$  is a set of agents;  $\mathcal{X} = \{x_1, \dots, x_N\}$  are variables owned by the agents;  $\mathcal{D} = \{\mathcal{D}_{x_1}, \dots, \mathcal{D}_{x_N}\}$  is a set of finite domains, such that variable  $x_i$  takes values in  $\mathcal{D}_{x_i} = \{v_1, \dots, v_k\}$ ;  $\mathcal{C} = \{u_1, \dots, u_M\}$  is a set of soft constraints, where each  $u_i$  defines a utility  $\in \mathbb{R} \cup \{-\infty\}$  for each combination of assignments to a subset of variables  $\mathcal{X}_i \subseteq \mathcal{X}$  (a constraint is initially known only to the agents involved);  $\mu : \mathcal{X} \rightarrow \mathcal{A}$  is a function mapping variables to their associated agent. A solution to the DCOP is an assignment  $\mathcal{X}^* = \{x_1^*, \dots, x_N^*\}$  to all variables that maximizes the overall sum of costs<sup>4</sup>:

$$\sum_{m=1}^M u_m(\mathcal{X}_m) \quad (1)$$

<sup>4</sup> Note that the notion of cost can be replaced by the notion of cost  $\in \mathbb{R} \cup \{+\infty\}$ . In this case, solving a DCOP is a minimization problem of the overall sum of costs.

As highlighted in [1], 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.

A binary DCOP can be represented as a constraint graph, where vertices represent variables, and edge represent binary constraints. In the case of n-ary constraints, 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.

**Definition 2 (Factor Graph).** A factor graph of a DCOP as in Def. 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  $E = \{e_{ij} \mid x_i \in \mathcal{X}_j\}$ .

When the graph representing the DCOP contains at least a cycle, we call it a *cyclic* DCOP; otherwise, it is *acyclic*.

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 [9], or inference algorithms like DPOP [15] or ActionGDL [19], 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, *approximate algorithms* like Max-Sum [3] or MGM [8] have the great advantage of being fast with a limited memory print and communication load, but losing optimality in some settings –e.g. Max-Sum 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 Max-Sum, where the notion of marginal values describes the dependency of the global utility function on variables.

## 2.2 From Belief-Propagation to Max-Sum

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 [7]:  $F(x) = \prod_{m=1}^M f_m(\mathcal{X}_m)$ . The sum-product algorithm provides an efficient local message passing procedure to compute the marginal functions of all variables simultaneously. The marginal function,  $z_n(x_n)$  describes the total dependency of the global function  $F(x)$  on variable  $x_n$ :  $z_n(x_n) = \sum_{\{x'\}, n' \neq n} F(\mathcal{X}_{n'})$ .

BP operates iteratively propagating messages  $m_{i \rightarrow j}$  (tables associating marginals to each value of variables) along the edges of the factor graph. When the factor graph is a tree, BP algorithm computes the exact marginals and converge in a finite number a steps depending on the diameter of the graph [7]. Max-product is an alternative version of sum-product which computes the maximum value instead of the sum.

Built as a derivative of max-product, Max-Sum is an approximate algorithm to solve DCOP [3]. The main evolution is the way messages are assessed, to pass from product to sum operator through logarithmic translation. And as a consequence, Max-Sum computes an assignment  $\mathcal{X}^*$

**Algorithm 1:** The BP-guided decimation algorithm from [11]

---

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

```

1 initialize BP messages
2  $\mathcal{U} \leftarrow \emptyset$ 
3 for  $t = 1, \dots, n$  do
4     run BP until the stopping criterion is met
5     choose  $x_i \in \mathcal{X} \setminus \mathcal{U}$  uniformly at random
6     compute the BP marginal  $z_i(x_i)$ 
7     choose  $x_i^*$  distributed according to  $z_i$ 
8     fix  $x_i = x_i^*$ 
9      $\mathcal{U} \leftarrow \mathcal{U} \cup \{x_i\}$ 
10    simplify the factor graph
11    if a contradiction is found, return FAIL
12 return  $\mathcal{X}^*$ 

```

---

that maximizes the DCOP objective in Equation 1. Depending on the DCOP to solve, Max-Sum may be used with two different termination rules: (i) continue until convergence (no more exchanged messages, because when a variables or a factor receives twice the same message from the same emitter it does not propagates); (ii) propagate message for a fixed number of iterations per agent. Max-Sum is optimal on tree-shaped factor graphs, and still perform well on cyclic settings. But there exist problems for which Max-Sum 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 Max-Sum performance on cyclic graphs, [20] proposed two extensions to Max-Sum: (i) Max-Sum\_AD which operates Max-Sum on a directed acyclic graph built from the factor graph, and alternates direction at a fixed rate (a parameter of the algorithm); (ii) Max-Sum\_AD\_VP which operates Max-Sum\_AD and propagates current values of variables when sending Max-Sum 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 improves the quality of the solution: Max-Sum\_AD\_VP found solutions that approximate the optimal solution by a factor of roughly 1.1 on average. However, the study does not consider the number of exchanged messages, or the time required to converge and terminate Max-Sum\_AD\_VP.

### 2.3 BP-guided Decimation

In this paper, we propose to take inspiration from work done in computational physics [13], as to cope with cyclicity in DCOP. Notably, [5] introduced the notion of decimation in constraint satisfaction, especially  $k$ -satisfiability, where variables are binary,  $x_i \in \{0, 1\}$ , and each constraint requires  $k$  of the variables to be different from a specific  $k$ -uple. 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  $x_i^*$  for the corresponding variable; (3) replace the constraint satisfaction problem with the one obtained by fixing  $x_i$  to  $x_i^*$ . The BP-guided decimation procedure is shown in Algorithm 1, whose performances are analysed in [11,13].

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 determined by random sampling according to 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). In some settings, BP-guided decimation may fail, if random choices assign a value to a variable which is not consistent with other decimated variables.

Some comments can be made on this approach. First, *relying on marginal values* is a key feature, and 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 [11], this procedure is used to solve satisfiability problems, the approach can easily be implemented to *cope with optimization problems*. For instance, the inference library libDAI proposes an implementation of decimation for discrete approximate inference in graphical models [12], which was amongst the three winners of the UAI 2010 Approximate Inference Challenge<sup>5</sup>.

## 2.4 State of a Factor Graph Representation

The previous BP-based algorithm operates on factor graph representing the problem. “Operates” means that the algorithms create a data structure representing the factor graph which evolves with time : marginal values change, variables disappear, messages are sent/received, etc. Commonly, the logical representation of a factor graph is a set of nodes connected depending on the connectivity of the graph. Each such node has a state which stores some useful values.

**Definition 3.** *The current state  $FG^t$  at time  $t$  of a factor graph  $FG = \langle \mathcal{X}, \mathcal{C}, E \rangle$  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  $m_{i \rightarrow j}$ , the set of decimated variables  $\mathcal{U}$ , and other algorithm-specific data.*

We can consider that for a given problem, many factor graph states may exist. We denote  $\mathfrak{S}$  the set of possible factor graph states, and  $\mathfrak{S}(FG) \subset \mathfrak{S}$  the set of possible states for the factor graph  $FG$ .

## 3 DECIMAXSUM: Extending Max-Sum with Decimation

While mainly designed as a centralized algorithm and studied on  $k$ -SAT problems, BP-guided decimation could be utilized for solving DCOP with a few modifications. To the best of our knowledge, this approach has never been proposed for improving Max-Sum algorithm. Here we expound 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 [11] in order to define a more general framework, in which other BP-based existing algorithms could fit. First, the main focus is *decimation*, which means assigning a value to a variable as 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 is an impacting decision. This is why our framework

<sup>5</sup> <http://www.cs.huji.ac.il/project/UAI10/>

is mainly based on answering three questions: (i) when is decimation triggered, (ii) which variable(s) to decimate, (iii) which value to assign to the decimated variable(s)? Several criteria can be defined for answering each question, and the `DECIMAXSUM` specifies such criteria as *decimation policies*, that are fundamental parameters of the decimation procedure.

**Definition 4 (Decimation Policy).** A decimation policy is a tuple  $\pi = \langle \Theta, \Phi, \Upsilon, \Lambda \rangle$  where:

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

A rich population of decimation-based algorithm can be modeled through this framework by combining decimation policies. For instance, one can consider a `DECIMAXSUM` instance, which (i) triggers decimation once BP has converged, (ii) chooses randomly a variable to decimated 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 result in the classical BP-guided decimation algorithm from [11]. However, as many more decimation policies can be defined and combined, we fall into a more general framework generating a whole family of algorithms.

### 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 the convergence (or time limit) of BP. Criterion  $\Theta$  may relies on other components of the state of the factor graph. Contrary to classical BP-guided decimation, there may be several variables to decimate at the same time (like in some variants of DSA or MGM) and that variables can be chosen in an informed manner (and not randomly), using criterion  $\Upsilon$ . Values assigned to decimated variables, are not necessarily chosen stochastically, but are assigned using the function  $\Lambda$  that can be deterministic (still depending on the current state of the FG). Since, here we're not in the  $k$ -satisfiability case, but in an optimization case, there is no failure (only suboptimality), contrary to Algorithm 1. Finally, once all variables have been decimated, the output consists in decoding the state  $FG^t$ , i.e. getting the values assigned to decimated variables. This means that finally `DECIMAXSUM` is performing decoding while solving the problem, which is not a common feature in other DCOP algorithms, like classical Max-Sum or DSA. Indeed, once these algorithms halt, a decoding phase must be performed to extract the solution from the variables' states.

While presented as a classical algorithm, let us note that decimation is meant to be implemented in a distributed and concurrent manner, depending on the decimation policy components. The rest of the section details and illustrates each of these decimation policies component with some examples.

### 3.3 Triggering Decimation ( $\Theta$ criterion)

In the original approach proposed by [11], decimation is triggered once BP has converged. In a distributed settings and diffusing algorithms like BP, this can be implemented using termination detection techniques.

$$\Theta_{\text{converge}}(s) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } s \text{ is quiescent} \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

---

**Algorithm 2:** The DECI-MAX-SUM framework as an algorithm
 

---

**Data:** A factor graph  $FG = \langle \mathcal{X}, \mathcal{C}, E \rangle$ , a decimation policy  $\pi = \langle \Theta, \Phi, \mathcal{T}, \Lambda \rangle$

**Result:** A feasible assignment  $\mathcal{X}^*$

```

1 initialize BP messages
2  $\mathcal{U} \leftarrow \emptyset$ 
3 while  $\mathcal{U} \neq \mathcal{X}$  do
4     run BP until decimation triggers, i.e.  $\Theta(FG^t) = 1$  // Sect. 3.3
5     choose variables to decimate,  $\mathcal{X}' = \{x_i \in \Phi(FG^t) \mid \mathcal{T}(x_i, FG^t)\}$  // Sect. 3.4
6     for  $x_i \in \mathcal{X}'$  do // Sect. 3.5
7          $x_i \leftarrow \Lambda(x_i, FG^t)$ 
8          $\mathcal{U} \leftarrow \mathcal{U} \cup \{x_i\}$ 
9         simplify  $FG^t$  // remove variables, slice factors
10 return  $\mathcal{X}^*$  by decoding  $\mathcal{U}$ 
    
```

---

This trigger consists in 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 [6]. Algorithms like *DijkstraScholten* can detect such global state by implementing a send/receive network algorithm, based on the same graph than  $FG$  [6]. Note that such techniques generate extra communication load for termination detection-dedicated messages.

Due to the Max-Sum behavior on loopy factor graphs, convergence may not be reached [20]. The common workaround is to run BP for a fixed number of iterations in case there is no convergence. Setting this time limit (namely LIMIT) might be really problem-dependent.

$$\Theta_{\text{time}}(s) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } \text{time}(s) = \text{LIMIT} \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

In synchronous settings (all variables and factors are executed synchronously, step by step), getting the iteration number of the current state of the FG,  $\text{time}(s)$ , can be done in a distributed manner, as usually done in Max-Sum. In the asynchronous case, one can either (i) use a shared clock, or (ii) count locally outgoing messages within each variable, and once a variable has sent a limit number of messages, decimation is triggered.

In some settings with strong time or computation constraints (e.g. sensor networks [3], internet-of-things [17]), waiting 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 [14]).

$$\Theta_{\text{frequency}}(s) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } \text{time}(s) \bmod f(s) = 0 \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

where  $f$  is a function of the current state of the system, for instance :

- $f(s) = \text{RATE}$ , with a predefined decimation frequency,
- $f(s) = \text{BUDGET}/|\mathcal{X}|$ , with a predefined computation budget,
- $f(s) = 2 \times \text{time}(s)$ , for an decreasing decimation frequency.

Finally, another approach could be to trigger decimation once a loop in the FG is detected. Indeed, decimation is used here to cope with loops, so decimating variables, which could potentially break loops, seems a good approach.



$$\Theta_{\text{loop}}(s) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } \exists x_i \in \mathcal{X}, |loop(x_i)| > 1 \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

where  $loop(x_i)$  is the set of agents in the same first loop that  $x_i$  just discovered. Detecting loops in the FG can be implemented during BP, by adding some metadata on the BP messages, like done in the DFS-tree construction phase of algorithms like DPOP or ADOPT.

### 3.4 Deciding the Subset of Variables to Decimate ( $\Phi$ and $\Upsilon$ criteria)

Now our system has detected decimation should be triggered, the following question is “which variables to decimate?” In [11], the variable is chosen randomly in a uniform manner, while in [12], the variable with a the maximum entropy over its marginal values (the most *determined* variable) is selected. Obviously, exploiting the marginal values, build throughout propagation is a good idea.

**From which subset choosing the candidate variables to decimate?** Both [11] and [12] select the only variable to decimate amongst the whole set of non-decimated variables (cf. line 5 in Algorithm 1). Here,  $\Phi$  criterion is specified as follows:

$$\Phi_{\text{all}}(s) \stackrel{\text{def}}{=} \mathcal{X} \setminus \mathcal{U} \quad (6)$$

However, this selection on the whole set of variables can be discussed when using local decimation triggers, like loop detection. In such case, selecting the variables to decimate within the agents in the loop, or the one which detected the loop sounds better. Another approach is to consider selecting agents depending on the past state of the system. For instance, if a variable has been decimated, good future candidates for decimation could be its direct neighbors in the FG:

$$\Phi_{\text{neighbors}}(s) \stackrel{\text{def}}{=} \{x \in \mathcal{X} \setminus \mathcal{U} \mid neighbors(x) \cap \mathcal{U} \neq \emptyset\} \quad (7)$$

with  $neighbors(x_i) = \{x_j \in \mathcal{X} \mid j \neq i, \exists e_{ik}, e_{kj} \in E\}$ .

**Which criteria to decide whether the variable decimate?** Now, we have to specify the  $\Upsilon$  criterion used to decide which candidates decimate. In [11], it is fully random: it does not depends on the current state of the variables. It corresponds to make each variable roll a dice and choosing the greatest draw:

$$\Upsilon_{\text{max\_rand}}(x_i, s) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } \forall x_j \neq x_i \in \mathcal{X}, rand(x_i) > rand(x_j) \\ 0, & \text{otherwise} \end{cases} \quad (8)$$

where  $rand(x)$  stands for the output of a random number generator (namely *sample*) using a uniform distribution (e.g.  $U[0, 1]$ ).

In [12], the variable with the maximal entropy over its marginal values is selected. This means the variable for which marginal values seems to be the most informed, in the Shannon’s Information Theory sense, is chosen:

$$\Upsilon_{\text{max\_entropy}}(x_i, s) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } \forall x_j \neq x_i \in \mathcal{X}, H(z_i(x_i)) > H(z_j(x_j)) \\ 0, & \text{otherwise} \end{cases} \quad (9)$$

with  $H(z_k(x_k)) = - \sum_{d \in \mathcal{D}_k} z_k(x_k)(d) \log(z_k(x_k)(d))$ .

From this, other criteria can be derived. For instance, instead of using entropy, one can consider the maximal normalized marginal value:

$$\mathcal{Y}_{\max\_marginal}(x_i, s) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } \forall x_j \neq x_i \in \mathcal{X}, \max_{d \in \mathcal{D}_i}(z_i(x_i)(d)) > \max_{d \in \mathcal{D}_i}(z_j(x_j)(d)) \\ 0, & \text{otherwise} \end{cases} \quad (10)$$

If several variables can be decimated at the same time, one may consider selecting the set of variable having an entropy or a normalized marginal value greater than a given threshold, to only decimate variable which are “sufficiently” determined. Hence, this approach requires setting another parameter (namely THRESHOLD):

$$\mathcal{Y}_{\text{threshold\_entropy}}(x_i, s) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } H(z_i(x_i)) > \text{THRESHOLD} \\ 0, & \text{otherwise} \end{cases} \quad (11)$$

Of course, many combination of the aforementioned criteria, and other criteria could be considered in our framework. We don’t discuss here criteria like in DSA which does not rely on marginal values, but on stochastic decision.

**Which subset of variables the decision to decimate a variable depends on?** Behind this question lies the question of coordinating the variable selection. Indeed, if computing criterion  $\mathcal{Y}$  does not depend on the decision of other variables, the procedure is fully distributable at low communication cost, as for policies like (11). At the contrary, if the decision requires to be aware of the state of other variables, as for policies like (8), (9) and (10), the procedure will require some system-scale coordination messages. In [11] and [12], decimation only concerns all the variables, from which only one will be chosen. This requires a global coordination, or a distributed leader election protocol which may require an underlying network (ring, spanning tree, etc.), like the one used for quiescence detection, to propagate election messages [6].

In some cases, the decimation decision might be at local scale, when variables will make their decision depending on the decision of their direct neighbors, or variables in the same loop. In this case, less coordination messages will be required. For instance, if considering decimating variables in a loop, only variables in the loop will implement a leader election protocol. All policies, from (8) to (10), could be extended in the same manner, by replacing  $\mathcal{X}$  by  $loop(x_i)$ ,  $neighbours(x_i)$ , or any subset of  $\mathcal{X}$ . For instance:

$$\mathcal{Y}_{\max\_rand\_loop}(x_i, s) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } \forall x_j \neq x_i \in loop(x_i), rand(x_i) > rand(x_j) \\ 0, & \text{otherwise} \end{cases} \quad (12)$$

### 3.5 Deciding the Values to Assign To Decimated Variables (A criterion)

Now variables to decimate have been selected, the question is “which values to assign?” Usually, in BP-based algorithms, the simplest way to select values for variables, after propagation, is to assign values with maximal marginal value (or utility). [12] is using such a criterion for inference:

$$A_{\max\_marginal}(x_i, s) \stackrel{\text{def}}{=} \underset{d \in \mathcal{D}_i}{\operatorname{argmax}} z_i(x_i)(d) \quad (13)$$

While, the policy is deterministic, in [11] the choice of the value is a random choice using the marginal values as a probability distribution:

$$A_{\text{sample\_marginal}}(x_i, s) \stackrel{\text{def}}{=} \text{sample}(z_i(x_i)) \quad (14)$$

Once again, these are only some examples of policies exploiting BP, and one can easily specify many more.

## 4 Experiments

In this section we evaluate the performance of different combinations of decimation policies in DECIMAXSUM, on a classical optimization model (Ising model), against classical Max-Sum [3] and its extension Max-Sum\_AD\_VP [20], we have implemented in our own framework.

### 4.1 Ising Model

Since we are interested in evaluating our algorithms in the presence of strong dependencies among the values of variables, we evaluate them on Ising model which is a widely used benchmark in statistical physics [4]. We use here the same settings than [18]. Here, constraint graphs are rectangular grids where each binary variable  $x_i$  is connected to its four closer neighbors (with toroidal links which 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  $\kappa_{ij}$  from a uniform distribution  $U[-\beta, \beta]$  and then assigning

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

The  $\beta$  parameter controls the average strength of interactions. In our experiments we set  $\beta$  to 1.6. The weight for each unary constraint  $r_i$  is determined by sampling  $\kappa_i$  from a uniform distribution  $U[-0.05, 0.05]$  and then assigning  $r_i(0) = \kappa_i$  and  $r_i(1) = -\kappa_i$ .

### 4.2 Results and Analysis

In this section we analyse results of different DECIMAXSUM combinations to solve squared-shape Ising problems with side size varying from 10 to 20 (e.g. 100 to 400 variables). We implemented the following combinations:

- 11 DECIMAXSUM instances with different decimation policies using the following criteria:
  - *trigger policies* ( $\Theta$  criterion):
    - \*  $\Theta_{\text{converge}}$  (from equation 2, noted converge),
    - \* rate-based  $\Theta_{\text{frequency}}$  (from equation 4, noted 2-periodic, 3-periodic, 5-periodic, 10-periodic, 20-periodic, and 100-periodic),
    - \* budget-based  $\Theta_{\text{frequency}}$  (from equation 4, noted periodic),
  - *filter policy* ( $\Phi$  criterion):
    - \* the one that selects the whole set of variables as potential variables to decimate (i.e.  $\Phi_{\text{all}}$  from equation 6),
  - *perform policies* ( $\Upsilon$  criterion):
    - \*  $\Upsilon_{\text{max\_rand}}$  (from equation 8, noted random),
    - \*  $\Upsilon_{\text{max\_entropy}}$  (from equation 9, noted max\_entropy),
  - *assignment policies* ( $\Lambda$  criterion):
    - \* deterministic  $\Lambda_{\text{max\_marginal}}$  (from equation 13, noted deterministic),
    - \* sampled  $\Lambda_{\text{sample\_marginal}}$  (from equation 14, noted sampling),
- MaxSum, as defined in [3],
- MaxSum\_AD, as defined in [20],
- MaxSum\_AD\_VP, as defined in [20],
- Montanari-Decimation, as defined in [11],
- Mooij-Decimation, as defined in [12].

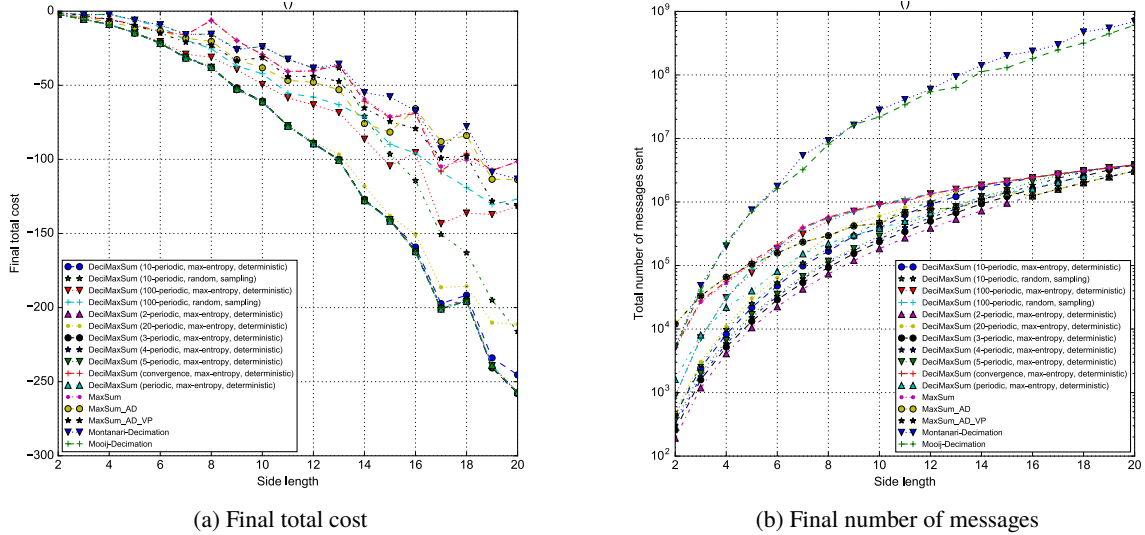


Fig. 1: Performances of DECIMAXSUM and other solution methods on Ising problems (average of 10 problems per generator’s parameter set, and average of 3 runs per problem).

Figure 1 presents two performance metrics (final total cost and total number of exchanged messages). Considering optimality of the final solutions obtained by the different solution methods and DECIMAXSUM instances, what appears is that very fast decimation combined with a deterministic decimation of the most determined variable (max\_entropy) presents the best cost. Besides, very fast decimation also imply that few messages are exchanged compared to other solution methods, since decimation cuts message propagations. However, all the solution methods (except Montanari-Decimation and Mooij-Decimation) tend to a comparable number of exchanged messages.

## 5 Conclusions

In this paper we have investigated how to extend Max-Sum method for solving distributed constraint optimization problems, by taking inspiration from the decimation mechanisms used to solve  $k$ -satisfiability problems by belief-propagation. We propose a parametric method, namely DECIMAXSUM, which can be set up with different decimation policies stating when to trigger decimation, which variables to decimate, and which value to assign to decimated variables. In this paper, we propose a library of such policies that can be combined to produce different versions of DECIMAXSUM. Our empirical results on different benchmarks show that some combinations of decimation policies outperform classical Max-Sum and its extension Max-Sum\_AD\_VP, specifically design to handle loops. DECIMAXSUM outputs better quality solutions in a reasonable number of message propagation.

There are several paths to future research. First, we only explore a limited set of decimation policies. We wish to investigate more complex ones, especially policies trigger when loops are detected by agents. In fact, since our overarching goal is to cope with loops, detecting them at the agent level seems a reasonable approach to initiate decimation in a cyclic network. This approach will require agents to implement cycle-detection protocol, by sending message history, while propagating marginals. In such a setting, several decimation election may arise concurrently in the graph. Second, we would like to generalize DECIMAXSUM framework to consider Max-Sum\_AD\_VP as a particular case of decimation: iterated decimation. Finally, we plan to applied

DECIMAXSUM on real world applications, with strong loopy nature, like the coordination of smart objects in IoT [17] or decentralized energy markets in the smart grid [2].

## References

1. Cerquides, J., Farinelli, A., Meseguer, P., Ramchurn, S.D.: A tutorial on optimization for multi-agent systems. *The Computer Journal* 57(6), 799–824 (2014), <http://dx.doi.org/10.1093/comjnl/bxt146>
2. Cerquides, J., Picard, G., Rodríguez-Aguilar, J.: Designing a marketplace for the trading and distribution of energy in the smart grid. In: 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS). pp. 1285–1293. International Foundation for Autonomous Agents and Multiagent Systems (2015), <http://www.aamas-conference.org/Proceedings/aamas2015/forms/contents.htm#I4>
3. Farinelli, A., Rogers, A., Petcu, A., Jennings, N.R.: Decentralised coordination of low-power embedded devices using the max-sum algorithm. In: International Conference on Autonomous Agents and Multiagent Systems (AAMAS’08). pp. 639–646 (2008), <http://dl.acm.org/citation.cfm?id=1402298.1402313>
4. Kolmogorov, V.: Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28(10), 1568–1583 (Oct 2006)
5. Krzakala, F., Montanari, A., Ricci-Tersenghi, F., Semerjian, G., Zdeborova, L.: Gibbs states and the set of solutions of random constraint satisfaction problems. *Proceedings of the National Academy of Science* 104, 10318–10323 (Jun 2007)
6. Lynch, N.: *Distributed Algorithms*. Morgan Kaufmann (1996)
7. Mackay, D.J.C.: *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, first edition edn. (Jun 2003)
8. Maheswaran, R., Pearce, J., Tambe, M.: Distributed algorithms for dcop: A graphical-game-based approach. In: Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems (PDCS), San Francisco, CA. pp. 432–439 (2004)
9. Modi, P.J., Shen, W., Tambe, M., Yokoo, M.: ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees. *Artificial Intelligence* 161(2), 149–180 (2005)
10. Modi, P., Shen, W., Tambe, M., Yokoo, M.: ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence Journal* (2005)
11. Montanari, A., Ricci-Tersenghi, F., Semerjian, G.: Solving constraint satisfaction problems through belief propagation-guided decimation. *CoRR abs/0709.1667* (2007), <http://arxiv.org/abs/0709.1667>
12. Mooij, J.M.: libDAI: A free and open source C++ library for discrete approximate inference in graphical models. *Journal of Machine Learning Research* 11, 2169–2173 (Aug 2010), <http://www.jmlr.org/papers/volume11/mooij10a/mooij10a.pdf>
13. Mézard, M., Montanari, A.: *Information, Physics, and Computation*. Oxford University Press (2009)
14. Navlakha, S., Barth, A.L., Bar-Joseph, Z.: Decreasing-rate pruning optimizes the construction of efficient and robust distributed networks. *PLOS Computational Biology* 11(7), 1–23 (07 2015), <http://dx.doi.org/10.1371/journal.pcbi.1004347>
15. Petcu, A., Faltings, B.: A scalable method for multiagent constraint optimization. In: International Joint Conference on Artificial Intelligence (IJCAI’05). pp. 266–271 (2005)
16. Rogers, A., Farinelli, A., Stranders, R., Jennings, N.: Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence* 175(2), 730 – 759 (2011), <http://www.sciencedirect.com/science/article/pii/S0004370210001803>
17. Rust, P., Picard, G., Ramparany, F.: Using message-passing DCOP algorithms to solve energy-efficient smart environment configuration problems. In: International Joint Conference on Artificial Intelligence (IJCAI). AAAI Press (2016)
18. Vinyals, M., Pujol, M., Rodríguez-Aguilar, J., Cerquides, J.: Divide and coordinate: solving dcops by agreement. In: International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS’10). pp. 149–156. IFAAMAS, Canada (2010)
19. Vinyals, M., Rodríguez-Aguilar, J., Cerquides, J.: Constructing a unifying theory of dynamic programming dcop algorithms via the generalized distributive law. *Autonomous Agents and Multi-Agent Systems* 22(3), 439–464 (2010), <http://dx.doi.org/10.1007/s10458-010-9132-7>
20. Zivan, R., Peled, H.: Max/min-sum distributed constraint optimization through value propagation on an alternating DAG. In: van der Hoek, W., Padgham, L., Conitzer, V., Winikoff, M. (eds.) International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 Volumes). pp. 265–272. IFAAMAS (2012), <http://dl.acm.org/citation.cfm?id=2343614>