DeciMaxSum: Using Decimation to Improve Max-Sum on Cyclic DCOPs

Jesús CERQUIDES [a]
Rémi EMONET [b] Gauthier PICARD [c] Juan A. RODRÍGUEZ-AGUILAR [a]
[a] *IIIA-CSIC, Campus UAB, 08193 Cerdanyola, Catalonia, Spain*
e-mail: {cerquide,jar}@iiia.csic.es
[b] *Université de Lyon, Laboratoire Hubert Curien UMR CNRS 5516, France*
[c] *Mines Saint-Etienne, Laboratoire Hubert Curien UMR CNRS 5516, France*
e-mail: remi.emonet@univ-st-etienne.fr,e-mail: picard@emse.fr

Abstract. When solving large distributed constraint optimization problems (DCOP), belief-propagation and incomplete inference algorithms are candidates of choice. However, these methods perform poorly when the factor graph is very loopy (i.e. cyclic) because of non-convergence and high communication costs. As to improving performances of the Max-Sum inference algorithm when solving loopy constraint optimization problems, we propose to take inspiration from the belief-propagation-guided decimation used to solve sparse random graphs (k-satisfiability). We introduce the DeciMaxSum method, parameterized in terms of policies to decide when to trigger decimation, which variables to decimate, and which values to assign to decimated variables. Our empirical analysis on a classical BP benchmark indicates that some of these combinations of policies outperform state-of-the-art competitors.

Keywords. DCOP, max-sum, decimation

1. Introduction

In multi-agent settings, distributed constraint optimization problems (DCOP) are convenient to model coordination issues agents have to face, like resource allocation, distributed planning or distributed configuration. Here, 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 incomplete ones. In some large settings, incomplete methods are better candidates, as evidenced by the extensive literature on the subject (see [2] for a complete review). One major difficulty for incomplete methods 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 [1] and its extensions like [12], have demonstrated good performance even on cyclic settings. However, there exist 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 original 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. The decimation is processed regularly after the convergence of a classical belief-propagation procedure. In [7,11], decimation has been used in the constraint satisfaction framework, for solving centralized k-satisfiability problems. 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 [16]. 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 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, here we propose a general framework for installing decimation in Max-Sum to solve DCOPs. More precisely, we make the following contributions. First, 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 (i) can be combined with any policy from (ii) and (iii). Second, 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. Finally, we implement and evaluate some of these combinations of decimation policies on a classical DCOP benchmarks (Ising model), 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 reviews background on the DCOP and expounds the decimation algorithm on 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

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

Definition 1 (DCOP)**.** A discrete *Distributed Constraint Optimization Problem* (or DCOP) is a tuple $\langle \mathscr{A}, \mathscr{X}, \mathscr{D}, \mathscr{C}, \mu \rangle$, where: $\mathscr{A} = \{a_1, ..., a_{|A|}\}$ is a set of agents; $\mathscr{X} = \{x_1, ..., x_N\}$ are variables owned by the agents; $\mathscr{D} = \{\mathscr{D}_{x_1}, ..., \mathscr{D}_{x_N}\}$ is a set of finite domains, such that variable x_i takes values in $\mathscr{D}_{x_i} = \{v_1, ..., v_k\}$; $\mathscr{C} = \{u_1, ..., u_M\}$ 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 $\mathscr{X}_i \subseteq \mathscr{X}$ (a constraint is initially known only to the agents involved); $\mu : \mathscr{X} \rightarrow \mathscr{A}$ is a function mapping variables to their associated agent. A *solution* to the DCOP is an assignment $\mathscr{X}^* = \{x_1^*, ..., x_N^*\}$ to all variables that maximizes the overall sum of costs : $\sum_{m=1}^{M} u_m(\mathscr{X}_m)$.

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

Definition 2 (Factor Graph)**.** A factor graph of a DCOP is a bipartite graph $FG = \langle \mathscr{X}, \mathscr{C}, E \rangle$ whose variable vertices corresponds to the set of variables $\mathscr{X}$, the set of factor vertices corresponds to the set constraints $\mathscr{C}$, and the set of edges is $E = \{e_{ij} \mid x_i \in \mathscr{X}_j\}$.

Much literature exists on algorithms for solving DCOPs which fall into two categories. On the one hand, *complete algorithms* like ADOPT and its extensions [6], or inference algorithms like DPOP [10] or ActionGDL [15], 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 [1] 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). Here we focus on belief-propagation-based algorithms, like Max-Sum, whose notion of marginal values describes the dependency of the global utility function on variables.

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 [5]: $F(x) = \prod_{m=1}^{M} f_m(\mathscr{X}_m)$. 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: $z_n(x_n) = \sum_{\{x'\}, n' \neq n} F(\mathscr{X}_{n'})$. BP operates iteratively propagating messages $m_{i \to j}$ (tables associating marginals to each value of variables) along the edges of the factor graph. These messages depend on the type of the emitter: (a) *from functions to variables*: $q_{n \to m}(x_n) = \prod_{m' \in \mathscr{V}(n) \setminus m} r_{m' \to n}(x_n)$ where $\mathscr{V}$ is the set of indexes of variables connected to the function f_m, and (b) *from variables to functions*: $r_{m \to n}(x_n) = \sum_{\mathscr{X}_m \setminus n} \left(f_m(\mathscr{X}_m) \prod_{n' \in \mathscr{F}(m) \setminus n} q_{n' \to m}(x_{n'}) \right)$ where $\mathscr{F}(n)$ is the set of indexes of functions connected to the variable x_n, and $\mathscr{X}_m \setminus n \stackrel{\text{def}}{=} \{x_{n'} : n' \in \mathscr{V}(m) \setminus n\}$. If the factor graph is a tree, BP computes the exact marginals and converge in a finite number a steps depending on the diameter of the graph [5]. 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 incomplete algorithm to solve DCOP [1]. The main evolution is the way messages are assessed, to pass from product to sum operator through logarithmic translation. In Max-Sum, messages from variables to factor are assessed as $Q_{n \to m}(x_n) = \alpha_{nm} + \sum_{m' \in \mathscr{V}(n) \setminus m} R_{m' \to n}(x_n)$ and messages for factor to variable are defined by $R_{m \to n}(x_n) = \max_{\mathscr{X}_m \setminus n} \left(u_m(\mathscr{X}_m) \sum_{n' \in \mathscr{F}(m) \setminus n} Q_{n' \to m}(x_{n'}) \right)$. And as a consequence, Max-Sum computes an assignment $\mathscr{X}^*$ that maximizes the DCOP objective. 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 variable or a factor receives twice the same message from the same sender it does not propagate); (ii) propagate message for a fixed number of iterations per agent. Max-Sum is optimal on tree-shaped factor graphs, and still performs well on some cyclic settings, but there are problems for which it fails to converge or converges to a sub-optimal state. In fact, on cyclic settings [1] identifies 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 messages slightly change at each update, and thus continue being propagated; (iii) neither the agents' preferred states, nor the messages converge and both display cyclic behavior.

As to improve Max-Sum performance on cyclic graphs, [16] 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.

BP-guided Decimation. In this paper, we propose to take inspiration from work done in computational physics, as to cope with cyclicity in DCOP. Notably, [7] 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-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 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 performances are analysed in [7,11]. 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). BP-guided decimation may eventually fail if random choices assign a value to a variable that is not consistent with other decimated variables.

Some comments apply to 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 [7], this procedure is used to solve satisfiability problems, the approach can easily be implemented to *cope with optimization problems*. For instance, the libDAI inference library proposes an implementation of decimation for discrete approximate inference in graphical models [8].

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. To the best of our knowledge, this approach has never been followed to improve Max-Sum. Here we expound the core contribution of this paper, namely the DeciMaxSum framework and its components.

The main idea is to extend the BP-guided decimation algorithm from [7] 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 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.

We note by FG^t the current state at time t of a factor graph $FG = \langle \mathscr{X}, \mathscr{C}, E \rangle$, 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 $m_{i \to j}$, the set of decimated variables $\mathscr{U}$, and other algorithm-specific data. We consider that many factor graph states may exist per problem. $\mathfrak{S}$ is the set of possible factor graph states, and $\mathfrak{S}(FG) \subset \mathfrak{S}$ is the set of possible states for factor graph FG.

Definition 3 (Decimation Policy)**.** A *decimation policy* is a tuple $\pi = \langle \Theta, \Phi, \Upsilon, \Lambda \rangle$ where: $\Theta : \mathfrak{S} \to \{0,1\}$ is the condition to trigger the decimation process, the *trigger policy*; $\Phi : \mathfrak{S} \to 2^{\mathscr{X}}$ is a *filter policy* that selects candidate variables to decimate; $\Upsilon : \mathscr{X} \times \mathfrak{S} \to \{0,1\}$ is the condition to perform decimation on a variable, namely *perform policy*; and $\Lambda : \mathscr{X} \times \mathfrak{S} \to \mathscr{D}_{\mathscr{X}}$ 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 [7]. However, as many more decimation policies can be defined and combined, we fall into a more general framework generating a whole family of algorithms.

We can summarize the DeciMaxSum framework using Algorithm 1. 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 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 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 Υ. Values assigned to decimated variables, are not necessarily chosen stochastically, but are assigned using the function Λ 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 [7]. 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.

3.1. Triggering Decimation (Θ criterion)

In the original approach proposed by [7], decimation is triggered once BP has converged. In a distributed settings and diffusing algorithms like BP, this can be implemented using termination detection techniques. Such 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 [4]. Algorithms like *DijkstraScholten* can detect such global state by implementing a send/receive network

Algorithm 1: 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 $\mathcal{X}^*$

initialize BP messages
$\mathcal{U} \leftarrow \emptyset$
while $\mathcal{U} \neq \mathcal{X}$ **do**
  run BP until decimation triggers, i.e. $\Theta(FG^t) = 1$ // *Sect. 3.1*
  choose variables to decimate, $\mathcal{X}' = \{x_i \in \Phi(FG^t) \mid \Upsilon(x_i, FG^t)\}$ // *Sect. 3.2*
  for $x_i \in \mathcal{X}'$ **do** // *Sect. 3.3*
    $x_i \leftarrow \Lambda(x_i, FG^t)$
    $\mathcal{U} \leftarrow \mathcal{U} \cup \{x_i\}$
    simplify FG^t // *remove variables, slice factors*
return $\mathcal{X}^*$ by decoding $\mathcal{U}$

algorithm, based on the same graph than FG [4]. Note that such techniques generate extra communication load for termination detection-dedicated messages.

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

Due to the Max-Sum behavior on cyclic factor graphs, convergence may not be reached [16]. 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. In synchronous settings (all variables and factors are executed synchronously, step by step), getting the iteration number of the current state of the FG, $time(s)$, can be done in a distributed manner, as usually be done in Max-Sum. In the asynchronous case, one can either (i) use a shared clock, or (ii) count locally outcoming messages within each variables, and once a variable has sent a limit number of messages, decimation is triggered.

$$\Theta_{\texttt{time}}(s) \stackrel{\text{def}}{=} \begin{cases} 1, \text{ if } time(s) = \texttt{LIMIT} \\ 0, \text{ otherwise} \end{cases} \tag{2}$$

In some settings with strong time or computation constraints (e.g. sensor networks [1], internet-of-things [13]), 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 [9]).

$$\Theta_{\texttt{frequency}}(s) \stackrel{\text{def}}{=} \begin{cases} 1, \text{ if } time(s) \bmod f(s) = 0 \\ 0, \text{ otherwise} \end{cases} \tag{3}$$

where f is a function of the current state of the system, for instance : $f(s) = \texttt{RATE}$, with a predefined decimation frequency; $f(s) = \texttt{BUDGET}/|\mathcal{X}|$, with a predefined computation budget; $f(s) = 2 \times time(s)$, for an decreasing decimation frequency. Finally, another approach could be to trigger decimation once a loop in the FG is detected, such that decimation could potentially break the detected loop.

$$\Theta_{\texttt{loop}}(s) \stackrel{\text{def}}{=} \begin{cases} 1, \text{ if } \exists x_i \in \mathcal{X}, |loop(x_i)| > 1 \\ 0, \text{ otherwise} \end{cases} \tag{4}$$

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.

3.2. Deciding the Subset of Variables to Decimate (Φ and Υ criteria)

In [7], the variable to decimate is chosen randomly in a uniform manner, while in [8], the variable with the lowest entropy over its marginal values (the most *determined* variable) is selected. Obviously, exploiting the marginal values, build throughout propagation seems relevant.

From which subset choosing the candidate variables to decimate? Both [7] and [8] select the only variable to decimate amongst the whole set of non-decimated variables, as follows:

$$\Phi_{\texttt{all}}(s) \stackrel{\text{def}}{=} \mathscr{X} \setminus \mathscr{U} \tag{5}$$

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

$$\Phi_{\texttt{neighbors}}(s) \stackrel{\text{def}}{=} \{x \in \mathscr{X} \setminus \mathscr{U} \mid neighbors(x) \cap \mathscr{U} \neq \emptyset\} \tag{6}$$

Which criteria to decide whether the variable decimate? Now, we have to specify the Υ criterion used to decide which candidates decimate. In [7], 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:

$$\Upsilon_{\texttt{max\_rand}}(x_i,s) \stackrel{\text{def}}{=} \begin{cases} 1, \text{ if } \forall x_j \neq x_i \in \mathscr{X}, rand(x_i) > rand(x_j) \\ 0, \text{ otherwise} \end{cases} \tag{7}$$

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

In [8], the variable with the lowest 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, $H(z_k(x_k)) = -\sum_{d \in \mathscr{D}_k} z_k(x_k)(d) \log(z_k(x_k)(d))$, is chosen:

$$\Upsilon_{\texttt{min\_entropy}}(x_i,s) \stackrel{\text{def}}{=} \begin{cases} 1, \text{ if } \forall x_j \neq x_i \in \mathscr{X}, H(z_i(x_i)) < H(z_j(x_j)) \\ 0, \text{ otherwise} \end{cases} \tag{8}$$

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

$$\Upsilon_{\texttt{max\_marginal}}(x_i,s) \stackrel{\text{def}}{=} \begin{cases} 1, \text{ if } \forall x_j \neq x_i \in \mathscr{X}, \max_{d \in \mathscr{D}_i}(z_i(x_i)(d)) > \max_{d \in \mathscr{D}_i}(z_j(x_j)(d)) \\ 0, \text{ otherwise} \end{cases} \tag{9}$$

Of course, many combinations of the aforementioned criteria, and other criteria could be considered in our framework.

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 Υ does not depend on the decision of other variables, the procedure is fully distributable at low communication cost. At the contrary, if the decision requires to be aware of the state of other variables, as for policies like (7), (8) and (9), the procedure will require some system-scale coordination messages. In [7,8], decimation 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 [4].

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 (7) to (9), could be extended in the same manner, by replacing $\mathscr{X}$ by $loop(x_i)$, $neighours(x_i)$, or any subset of $\mathscr{X}$. For instance:

$$\Upsilon_{\texttt{max\_rand\_loop}}(x_i,s) \stackrel{\text{def}}{=} \begin{cases} 1, \text{ if } \forall x_j \neq x_i \in loop(x_i), rand(x_i) > rand(x_j) \\ 0, \text{ otherwise} \end{cases} \tag{10}$$

3.3. Deciding the Values to Assign To Decimated Variables (Λ 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). [8] is using such a criterion for inference:

$$\Lambda_{\texttt{max\_marginal}}(x_i,s) \stackrel{\text{def}}{=} \underset{d \in \mathscr{D}_i}{\text{argmax}} z_i(x_i)(d) \tag{11}$$

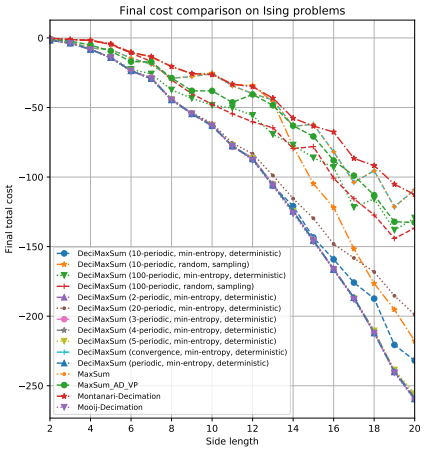
While, the policy is deterministic, in [7] the choice of the value is a random choice using the marginal values as a probability distribution:

$$\Lambda_{\texttt{sample}}(x_i,s) \stackrel{\text{def}}{=} sample(z_i(x_i)) \tag{12}$$

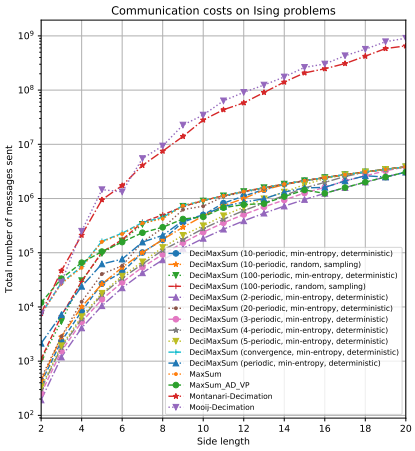
These are only few example policies exploiting BP, and one can easily specify many more.

4. Experiments

Here we evaluate the performance of different combinations of decimation policies in DeciMaxSum, on a classical optimization model, against classical Max-Sum [1] and its extension Max-Sum_AD_VP [16], we have implemented in our own framework. 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 [3]. We use here the same settings than [14]. 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 κ_{ij} from a uniform distribution $U[-\beta,\beta]$ and then assigning $r_{ij}(x_i,x_j) = \kappa_{ij}$ if $x_i = x_j$ and $r_{ij}(x_i,x_j) = -\kappa_{ij}$ otherwise. The β parameter controls the average strength of interactions. In our experiments we set β 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 assigning $r_i(0) = \kappa_i$ and $r_i(1) = -\kappa_i$.



(a) Final total cost 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).



(b) Final number of messages 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).

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 state-of-the-art solution methods: MaxSum [1], MaxSum_AD_VP, as defined in [16], Montanari-Decimation, as defined in [7], Mooij-Decimation, as defined in [8]. We also implemented several DeciMaxSum policies, and evaluate the following ones:

- with different triggers (Θ criterion): `converge`, rate-based `frequency` (noted 2-periodic, 3-periodic, 5-periodic, 10-periodic, 20-periodic, and 100-periodic), budget-based `frequency` (noted periodicn with a total budget of 1000),
- considering the whole set of variables as potential variables to decimate ($\Phi_{\texttt{all}}$),
- with different variable selection policies (Υ criterion): `max_rand` (noted random) and `min_entropy` (noted min_entropy),
- with different value selection policies (Λ criterion): `deterministic` and `sampling`.

Figures 1a and 1b 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. Indeed, periodic, 2-periodic, 3-periodic, 5-periodic, and 10-periodic while choosing the variables with the lowest entropy and choosing the value with the maximum marginal value are twice as good as state-of-the-art solution methods (Montanari-Decimation and Max-Sum's variants), and as good as Mooij-Decimation (which select the variable with the lowest entropy, but after convergence, or maximum budget). Communication-wise, very fast decimation in DeciMaxSum also implies that few messages are exchanged compared to other decimation solution methods (Montanari-Decimation and Mooij-Decimation), since there is no waiting time for convergence before decimation. However, all the other solution methods tend to a comparable number of exchanged messages. In short, DeciMaxSum with fast and deterministic marginal-value-guided decimation provides very good quality, with no communication extra-cost, on regular DCOPs like Ising.

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 also 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 designed to handle loops. DeciMaxSum outputs better quality solutions in a reasonable number of message propagation.

Acknowledgements

Funded by projects Collectiveware (MINECO/FEDER, TIN2015-66863-C2-1-R) and LOGISTAR(H2020 769142).

References

[1] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS'08)*, pages 639–646, 2008.

[2] F. Fioretto, E. Pontelli, and W. Yeoh. Distributed constraint optimization problems and applications: A survey. *Journal of Artificial Intelligence Research*, 61:623–698, 2018.

[3] V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1568–1583, Oct 2006.

[4] N. Lynch. *Disributed Algorithms*. Morgan Kaufmann, 1996.

[5] D. J. C. Mackay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003.

[6] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees. *Artificial Intelligence*, 161(2):149–180, 2005.

[7] A. Montanari, F. Ricci-Tersenghi, and G. Semerjian. Solving constraint satisfaction problems through belief propagation-guided decimation. *CoRR*, abs/0709.1667, 2007.

[8] 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, Aug. 2010.

[9] 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(7):1–23, 07 2015.

[10] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 266–271, 2005.

[11] F. Ricci-Tersenghi and G. Semerjian. On the cavity method for decimated random constraint satisfaction problems and the analysis of belief propagation guided decimation algorithms. *Journal of Statistical Mechanics: Theory and Experiment*, 2009(09):P09001, 2009.

[12] A. Rogers, A. Farinelli, R. Stranders, and N. Jennings. Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence*, 175(2):730 – 759, 2011.

[13] P. Rust, G. Picard, and F. Ramparany. Using message-passing DCOP algorithms to solve energy-efficient smart environment configuration problems. In *IJCAI'16*. AAAI Press, 2016.

[14] M. Vinyals, M. Pujol, J. Rodríguez-Aguilar, and J. Cerquides. Divide and coordinate: solving dcops by agreement. In *International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'10)*, pages 149–156, Canada, 2010. IFAAMAS.

[15] M. Vinyals, J. 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(3):439–464, 2010.

[16] 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. *JAAMAS*, 31(5):1165–1207, 2017.