DUJASE Vol. 7 (1) 45-57, 2022 (January) *https://doi.org/10.3329/dujase.v7i1.62886*

GDPx: An Application Independent Pruning Technique to Reduce Computation Cost of Max-Product Belief Propagation Algorithm

Md. Mosaddek Khan[1]* and N. V. Q. Trung[2]

[1]*Department of Computer Science and Engineering, University of Dhaka, Dhaka, Bangladesh*

[2]*School of Electronics and Computer Science, University of Southampton, Southampton, United Kingdom*
**Email: mosaddek@du.ac.bd*

Received on 27 October 2021, Accepted for publication on 18 April 2022

ABSTRACT

The Max-Product belief propagation algorithm has been widely used to process constraints associated with optimization problems in a broad range of application domains such as information theory, multi-agent systems, image processing, etc. The constraint optimization of a given problem is typically accomplished by performing inference with the use of a message passing process. During the process, the Max-Product algorithm performs repetitive maximization operation, which has been considered as one of the main reasons the algorithm can be computationally expensive. In more detail, scalability becomes a challenge when Max-Product has to deal with constraint functions with high arity and/or variables with a large domain size. In either case, the ensuing exponential growth of search space can make the maximization operator of the algorithm computationally infeasible in practice. In effect, it is frequently observed that the output of an algorithm becomes obsolete or unusable as the optimization process takes too long. Specifically, the issue of an algorithm taking too long to complete its internal inference process becomes more severe and prevalent as the size of the problem increases. As a result, the practical scalability of such algorithms is constrained. However, it is challenging to maintain the solution quality while reducing the computation cost of the algorithm. This is important because success in doing so will eventually reduce the algorithm's overall completion time without compromising on the quality of its solution. To address this issue, we develop a generic pruning technique that enables the maximization operator of the Max-Product algorithm to operate on a significantly reduced search space of at least around 85% or more (i.e. empirical observation). Additionally, we demonstrate theoretically that the pruned search space obtained through our approach has no negative impact on the algorithm's outcome. Finally, further empirical evidence notably suggests that our proposed approach brings down 50% to around 99% of the time required to complete a single maximization operation.

Keywords: Belief Propagation, Generalized Distributive Law, Max-Product, Maximization Operation

1. Introduction

Belief propagation algorithms, originally invented by Pearl, have been used to solve constraint reasoning problems in a wide range of application domains including error correcting codes, speech recognition, image understanding and multi-agent coordination and constraint recommender systems [1, 2, 3, 4], etc. In general, the belief propagation algorithms, also known as message passing algorithms, deal with the constraint reasoning problems by performing inference on graphical models that have been used to represent such problems [5]. The graphical models, such as Bayesian networks, Markov random field, junction trees or factor graphs, have been used with the same amount of success to articulate problems with deterministic behaviour, as well as in situations involving probability distributions or uncertainty [6, 7, 8]. The former is typically named as the deterministic graphical models and the latter as the probabilistic graphical models.

It is worth mentioning that the initial intention was to employ the belief propagation algorithms only for graphical models without loops or cycles for which they are guaranteed to provide an exact or optimal solution. Nevertheless, enough empirical evidences have been found showing the effectiveness of this class of algorithms on a number of loopy graphical models [9, 10, 11]. Additionally, one very important feature of the message passing algorithms have been identified by Aji and McEliece [12], in which they have famously shown that any algorithm of this type can be seen as a special case of Generalized Distributive Law (GDL) over a couple of specific semiring operators. For example, two semiring operators, "max" and "product", are used to form one of the most studied message passing algorithm named Max-Product.

The Max-Product algorithm has received particular attention amongst all of the existing message passing algorithms. Similar to other such algorithms, Max-Product performs inference on a graphical model by either following a synchronous or an asynchronous message update protocol [6, 13]. The messages here are generated using the GDL framework that has an axiomatic tendency of computational savings [12]. In effect, this class of algorithms make efficient use of constrained computational and communication resources, and effectively represent and communicate complex utility relationships (generated from the constraints) through the graph. In any case, all the nodes of a graphical model continuously generate and exchange messages towards completing the inference process. During the process, the semiring operator "max" serves as the summary operator for the Max-Product algorithm. Moreover, nodes in this particular algorithm (and other similar algorithms of this class) calculate and propagate utilities (or costs)

for each possible value assignment of their neighbouring nodes. Thus, the nodes explicitly share the consequences of choosing non-preferred states with the preferred one during inference through a graphical representation. Eventually, this information helps the algorithm to achieve good solution quality for large and complex problems.

Despite these aforementioned advantages, scalability remains a widely acknowledged challenge for the belief propagation algorithms such as Max-Product [14, 15]. Specifically, they perform repetitive maximization operations (i.e. the semiring operator max) for each constraint function to select the locally best configuration of the associated variables, given the local utility function and a set of incoming messages. To be precise, a constraint function that depends on n variables having domains composed of d values each, will need to perform d^n computations for a maximizationoperation. As the system scales up, either due to discrete random variables with a very large number of possible states or constraint functions with high arity, the complexity of this step grows exponentially. Examples of such problems include massive task allocation in multi-agent settings, disparity estimation in computer vision, tracking problems in sensor networks, and error-control decoding. For such problems, and many other besides, it may be expensive to compute and/or store the messages. In essence, the inference process of the deployed message passing algorithm may take too long to complete, and as such their applicability be limited to only small-scale problem settings.

Motivated by this challenge, researchers have studied a variety of techniques in order to reduce the complexity of belief propagation algorithm in different applications. Specifically, over the past few years, a number of efforts have tried to improve the scalability of message passing algorithms by reducing the cost of the maximization operator. In particular, [15] and [16] reduce the domain size of variables associated with constraint functions for task allocation domains where nodes' action choices are strictly divided into working on a task or not. However, this method is completely application dependent, because it can only be applied to a specific problem formulation of task allocation domain. Moreover, [17] carries a branch and bound search utilizing constraint functions to ensure that the upper and lower bounds can be evaluated using only a subset of variable values. Nevertheless, the bounding function they propose to accomplish this is entirely devoted to mobile sensor coordination. Therefore, it is not directly applicable to general settings. A more general approach to reduce the cost of the maximization operator, called Generalized Fast Belief Propagation (G-FBP), is proposed in [14]. In this approach, they select and sort the top $cd^{\frac{n-1}{2}}$ values of the search space, presuming the maximum value can be found from these ranges. Here, c is a constant. Nevertheless, they also admit that they cannot guarantee in advance whether the presumption is true or false, and in the latter case G-FBP incurs a significant penalty in terms of the computational cost [18]. Recently, [19] develops a generic Function Decomposing and State Pruning (FDSP) technique based on branch-and-bound. FDSP includes Function Decomposing (FD) phase that effectively computes the function estimation with the intent to reduce the over-heads in computing an upper bound of a partial assignment. Moreover, its State Pruning (SP) phase is based on branch and bound that reduces the search space. Besides, they also theoretically prove that these bounds are monotonically non-increasing during the search process.

On the other hand, another line of effort has recently been sought to reduce the computation cost of the maximization operator of the Max-Sum message passing algorithm [18]. This is motivated by the aforementioned pre-processing sorting based approach G-FBP. Similar to G-FBP, it is a one-shot pruning technique. Notably, unlike G-FBP, they provide a theoretical guarantee that the reduced search space obtained by using their algorithm always provides the desired outcome from the maximization operator. The algorithm is generic in a sense that it can be applied to any application (or setting) of Max-Sum. To be exact, GDP computes the reduced search space by considering one of its two semiring operators "sum. Hence, GDP cannot be used on the maximization operator of the Max-Product algorithm in its current form, though the other benchmarking algorithm G-FBP is readily applicable to this algorithm. Considering this observation and the vast usability of Max-Product coupled with the theoretical guarantee, generic nature and significant empirical results of GDP leads to the fact that further investigation needs to be undertaken to comprehend whether GDP can be tailored for this particular message passing algorithm.

In light of the above background, this paper proposes a modified version of GDP, that we call that we call GDPx, that is applicable to the Max-Product algorithm, regardless of the application domain. Similar to GDP, GDPx operates as a part of the maximization operator, proveably without affecting its solution quality (see Lemma 1). In other words, we improve the computational efficiency of theMax-Product message passing algorithm by reducing the search space over which the maximization operation is computed. We empirically evaluate the performance of our approach, and we observea significant reduction of search space, ranging from around 85% to 99% by using this technique. More importantly, we show the relative performance gain of GDPx gets better with an increase in the variables' domain size and the constraint functions' arity, in which the maximization operator acts on.

The remainder of this paper is structured as follows. We describe the problem in more detail inthe section that follows. Then, in Section 3, we discuss the complete process of GDPx with a worked example. We end this section by providing theoretical analyses. Subsequently, in Section 4, wepresent the empirical results of our method compared to the current state-of-the-art, and Section 5 concludes.

2. Problem Formulation

A constraint reasoning problem that can be solved using the Max-Product belief propagation algorithm is defined by a tuple X, D, F, where X is a set of discrete variables $\{x_0, x_1, \ldots, x_m\}$ and $D = \{D_0, D_1, \ldots, D_m\}$ is a set of discrete and finite variable domains. Each variable x_i can take value from the states of the corresponding domain D_i. F is a set of constraint functions $\{F_1, F_2, \ldots, F_L\}$, where each $F_i \in F$ is a function associated with a subset of variables $\boldsymbol{x}_i \in X$ defining the relationship among the variables in $\boldsymbol{x}_i$. Thus, thefunction $F_i(\boldsymbol{x}_i)$ denotes the value for each possible assignment ofthe variables in $\boldsymbol{x}_i$. Notably, the dependencies between the variables and the functions generate a bipartite graph, called a factor graph. The max-product algorithm operates directly in this particular graphical representation of a deployed problem. In a factor graph, each constraint function $F_i(\boldsymbol{x}_i)$ is represented by a square node and is connected to each of its associated variable nodes $\boldsymbol{x}_i$ (denoted by circles) by an individual edge. Note that the term function is also known as factor, and they are used interchangeably throughout this paper. Hence, $\boldsymbol{x}_i$ is the arity of $F_i(\boldsymbol{x}_i)$ in this particular graphical representation. Within the model, the objective is to assign values to the variables X from their corresponding domains in order to either maximize or minimize the global objective function, which eventually produces the value of each variable, X^*.

For example, Fig. 1 depicts the relationship among variables and functions in a factor graph representation. Here, we have a set of four variables $X = \{x_0, x_1, x_2, x_3\}$ and a set of two functions $F = \{F_0, F_1\}$. . Moreover, . is a set of discrete and finite variable domains, each variable $x_i \in X$ can take its value from the domain D_i .The ultimate objective is to either maximize or minimize a global objective function $F(x_0, x_1, x_2, x_3)$. Here, the global objective function is aproduct of two local functions $F_0(x_0, x_1, x_2)$ and $F_1(x_1, x_2, x_3)$.

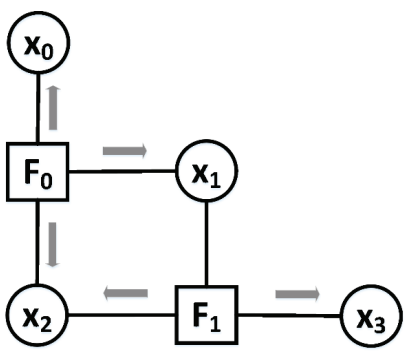


Fig. 1. A sample factor graph representation with two function/factor nodes $\{F_0, F_1\}$ and four variable nodes $\{x_0, x_1, x_2, x_3\}$, illustrating a global objective function $F(x_0, x_1, x_2, x_3)$. In the figure, variables are denoted by circles and factors are squares. Here, the grey arrows are used to highlight the factor-to-variable messages of the Max-Product belief propagation algorithm, each of which requires the maximization operation to be performed.

$$X^* = arg\,max_X \prod_{i=1}^{L} F_i(\boldsymbol{x}_i) \vee \square$$
$$X^* = arg\,min_X \prod_{i=1}^{L} F_i(\boldsymbol{x}_i) \tag{1}$$

In the factor graph, F_0 is associated (i.e. connected) with three variable nodes, and as such, the arity of the constraint function F_0 is 3. Similar to F_0, the arity of constraint function F_1 is 3 in this particular example. Note that the term function is also known as factor, and they are used interchangeably throughout this paper.

As mentioned in the previous section, belief propagation algorithms generally follow a message passing protocol (also known as belief update or summary propagation protocol) to exchange messages (i.e. beliefs) among the nodes of the factor graph representation of the aforementioned formulation. Notably, the Max-Product algorithm uses Equations 2 and 3 for their message passing, and they can be directly applied to the factor graph. Specifically, the variable and function nodes of a factor graph continuously exchange messages (variable x_i to function F_j (Equation 2) and function F_j to variable x_i (Equation 3) to compute an approximation of the impact that each of the variable's value have on the global objective function by building a local objective function $Z_i(x_i)$. In Equations 2 - 4, M_i stands for the set of functions connected to variable x_i and N_j represents the set of variables connected to function F_j. Once the function is built (Equation 4), each variable picks the value that maximizes the function by finding $\arg\max_{x_i}(Z_i(x_i))$.

$$Q_{x_i \to F_j}(x_i) = \prod_{F_k \in M_i \setminus F_j} R_{F_k \to x_i}(x_i) \tag{2}$$

$$R_{F_j \to x_i}(x_i) = \max_{\boldsymbol{x}_j \setminus x_i} \left[F_j(\boldsymbol{x}_j) + \prod_{x_k \in N_j \setminus x_i} Q_{x_k \to F_j}(x_k) \right] \tag{3}$$

$$Z_i(x_i) = \prod_{F_j \in M_i} R_{F_j \to x_i}(x_i) \tag{4}$$

As discussed previously, due to the potentially large parameter domain size and constraint functions with high arity, the maximization operator of the factor-to-variable message is the main reason the max-product belief propagation algorithm can be computationally expensive. This can be visualized from an example where a function node has five variable nodes connected to it, meaning the arity of the function is $n=5$. Here, we assume each of the variables can take its value from 10 possible options (i.e. states of the domain), implying that the domain size is $d=10$ for each of the variables. In this case, the function node has to perform 10^5 or $100,000$ operations to generate a message for one of its neighbouring variable nodes. Now, each of the function nodes in a factor graph has to generate and send a single message to each of its neighbours to complete a single round of message passing [6, 12]. For example, function node F_0 of Figure 1has to send a distinct message (grey arrow) to each of its neighbouring variable nodes x_0, x_1 and x_2. Each of these messages includes the expensive maximization operator. Under such circumstances, it is possible to significantlyreduce the computational cost of this step. Meanwhile, it is essential to ensure that this reduction process does not limit the algorithms' applicability, as well as not affecting the solution quality. We deal with the issue that arises from the trade-off in the remainder of this paper.

3. The GDPx Algorithm

GDPx (Algorithm 1) works as a part of Equation 3, which represents a function-to-variable message of the Max-Product belief propagation algorithm, in order to reduce the search space over which the maximization needs to be computed. This algorithm requires as inputs a sending function node $F_j\left(\boldsymbol{x}_j\right)$ whose utility depends on a set of variable nodes ($\boldsymbol{x}_j$) associated with it (i.e.neighbours), a receiving variable node $x_i \in \boldsymbol{x}_j$ and all the incoming messages from the neighbour(s) of F_j apart from the receiving node x_i, denoted as $\mathcal{M}_{x_j \setminus x_i}$. Finally, GDPx returns a pruned range of values (i.e. $\mathcal{R}_j$) for each state i of the domains of the variables over which the maximization operation needs to be performed to generate the message from the function node F_j to the variable node x_i (i.e. $R_{F_j \to x_i}\left(x_i\right)$).

In more detail, S stands for a set $\{s_1, s_2, \ldots, s_r\}$ representing each state of the domains corresponding to $\boldsymbol{x}_j$ (line 1 of Algorithm 1). This implies that S is the union ($\cup$) of those sets of states, each of which corresponds to the domain of a variable in $\boldsymbol{x}_j$. Line 2 sorts the local utility of the sending function node F_j independently by each state $s_i \in S$. It is worth noting that this sorting can be carried out at runtime of a belief propagation algorithm without incurring an additional delay [18]. Then the total number of incoming messages received by F_j is represented by n (line 3). Note that, a complete worked example of GDPx is illustrated in Figure 2 where we use a part of the factor graph of Figure 1 to show a factor-to-variable (i.e. F_1 to x_3) message computation (Figure 2a), as well as the operation of GDPx on it (Figure 2b). Here, the local utility of the sending function node F_1 is shown in atable at the left side of Figure 2a, which is based on three domain states $\{R, B, G\}$ (for simplicity red, blue and green colours are used to distinguish the values of the states, respectively) and three neighbouring variable nodes x_1, x_2 and x_3. Moreover, the direction of two incoming messages ($n=2$) received by $F, \{0.06203, 0.05307, 0.09390\}$ and $\{0.08423, 0.06310, 0.04713\}$, from the variable nodes x_1 and x_2, respectively, are indicated using the dotted black arrows. Then, the arrow from node F_1 to variable node x_3 indicates the desired function-to-variable message $R_{F_1 \to x_3}\left(x_3\right) = \{3.603\times10^{-4}, 3.027\times10^{-4}, 2.309\times10^{-4}\}$, and the complete calculation is depicted in a table at the left side of Fig. 2a.

At this point, line 4 computes m which is the multiplication of the maximum values of each of the messages $\mathcal{M}_k \in \mathcal{M}_{x_j \setminus x_i}$ received by the sending function F_j, other than the receiving variable node x_i. Here, $\mathcal{M}_k$ is one ofthe n messages received by F_j. In the worked example of Figure 2b, since the maximum of the received messages $\{0.06203, 0.05307, 0.09390\}$ (i.e. $\mathcal{M}_1$) and $\{0.08423, 0.06310, 0.04713\}$ (i.e. $\mathcal{M}_2$) by F_1 are 0.09390 and 0.08423 respectively, the valueof $m = 0.09390 \times 0.08423 = 7.909\times10^{-3}$. Now, the for loop in lines $5-12$ generates the range of the values for each state $s_i \in S$ fromwhere we will always find the maximum value for the function F_j, and discardthe rest. To this end, the function $sortedVal_{s_i}\left(F_j\left(\boldsymbol{x}_j\right)\right)$ gets the sorted value of s_i from line 2, and stores them in an array $\mathcal{V}_i$ (line 6). Then, line 7 finds p, which is the maximum of the local utility values for the state s_i (i.e. $\max\left(\mathcal{V}_i\right)$). In the worked example, the sorted values of domain state R are stored in ,epicted in the left side of Figure 2b. Hence, the value of $p = \max\left(\mathcal{V}_R\right) = 6.668\times10^{-2}$. Afterwards, line 8 computes b, which is the multiplication of the

corresponding values of p from the incoming messages of F_j (i.e. $val_p(\mathcal{M}_k)$). In the example, the values corresponding to p (i.e. 6.668×10^{-2}) from two incoming messages are 9.390×10^{-2} and 4.713×10^{-2}, thus the value of $b = 9.390\times10^{-2}\times4.713\times10^{-2} = 4.425\times10^{-3}$. This can be seen in the first row of the rightmost table of Figure 2b. The rows related to the computation for the state R are summarized into this table from the rightmost table of Figure 2a, which the complete computation of the function F_1 to variable x_3 message based on domain states R, B and G. Having obtained the value of m and b from lines 4 and 8 respectively, line 9 gets the cut point value c, where we develop an equation to compute the value, which is $c = \frac{p\times b}{m}$. The desired maximum value for the state s_i must always be found by considering the rows corresponding to the values in the range $[p,c]$, denoted by $prunedRange_{s_i}([p,c])$ (lines $10-11$) (See Lemma 1 and its proof for the theoretical guarantee and the intuition behind the choice of the range).

Algorithm 1 The $\mathcal{GDP}$x Algorithm

Input:

$F_j(\mathbf{x}_j)$, local utility of sending function node F_j, where $\mathbf{x}_j$ is the set of variable nodes associated with F_j;
$x_i \in \mathbf{x}_j$, the variable node which is going to receive a message from F_j;
$\mathcal{M}_{\mathbf{x}_j\setminus x_i}$, received messages by F_j from all of its neighbouring variable nodes $\mathbf{x}_j$ other than x_i.

Output:

$\{\mathcal{R}_i \mid i = 1,\ldots,r\}$, pruned ranges of values of the states over which maximization needs to be performed to generate the message for F_j to x_i, where r is the number of states corresponding to the domains of $\mathbf{x}_j$.

1: $\mathbb{S} \leftarrow \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_r\}$; ▷ set of states corresponding to the domain of $\mathbf{x}_j$
2: Sort the local utility $F_j(\mathbf{x}_j)$ independently by each state $\mathbf{s}_j \in \mathbb{S}$;
3: $\mathbf{n} \leftarrow |\mathcal{M}_{\mathbf{x}_j\setminus x_i}|$;
4: $m \leftarrow \prod_{k=1}^{\mathbf{n}} \max(\mathcal{M}_k)$; ▷ $\mathcal{M}_k \in \mathcal{M}_{\mathbf{x}_j\setminus x_i}$ is one of the $\mathbf{n}$ messages received by F_j
5: **for** $\mathbf{s}_i \in \mathbb{S}$ **do**
6: $\mathcal{V}_i \leftarrow \text{sortedVal}_{\mathbf{s}_i}(F_j(\mathbf{x}_j))$; ▷ sort the values for the state $\mathbf{s}_i$ in $F_j(\mathbf{x}_j)$
7: $p \leftarrow \max(\mathcal{V}_i)$; ▷ max local utility value for the state $\mathbf{s}_i$
8: $b \leftarrow \prod_{k=1}^{\mathbf{n}} \text{val}_p(\mathcal{M}_k)$ ▷ product of the corresponding values of p
9: $c \leftarrow \frac{p\times b}{m}$; ▷ the cut-point
10: $\mathcal{R}_i \leftarrow \text{prunedRange}_{\mathbf{s}_i}([p,c])$; ▷ the resulted pruned range
11: **result** $\mathcal{R}_i$
12: **end for**

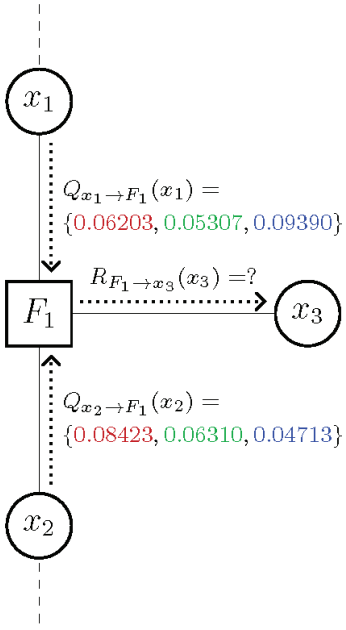
In the worked example of Figure 2b, the value $c = \frac{(6.668\times10^{-2})\times(4.425\times10^{-3})}{7.909\times10^{-3}} = 3.371\times10^{-2}$, given $p = 6.668\times10^{-2}$, $b = 4.425\times10^{-3}$ and $m = 7.909\times10^{-3}$. Hence, the resultant range for the state R of this particular example is $\mathcal{R}_{\mathcal{R}} = [6.668\times10^{-2}, 3.371\times10^{-2}]$. This implies that the desired maximum value will be found by considering the rows corresponding to the values in the range $\mathcal{R}_{\mathcal{R}}$. As can be seen in the right most table of Figure 2b, only considering the top three rows are sufficient to obtain the desired value of R; hence, it is not necessary to consider the remaining 6 rows for this particular instance. To be exact, the value for the state R after maximization is 3.603×10^{-4}, which isobtained from the row corresponding to the local utility value of 4.555×10^{-2}. In this way, GDPx reduces the computational cost of the expensive maximization operator. The grey colour is used to mark the discarded rows of the table. We can see that even for such a small instance, having

domain size $d = 3$ and arity $n = 3$, GDPx prunes more than 65% of the search space during the maximization of a state in computing the function-to-variable message.

As argued above, it is important to ensure that combining GDPx with Equation 3 does not make the computation of a function-to-variable message prohibitively expensive. In this regard, the original GDP algorithm proposed for the Max-Sum algorithm shows that its overall time complexity is $\mathcal{O}\left(r\, log\left|\mathcal{V}_i\right|\right)$ [18]. Thus, GDP is able to reduce the search space significantly at the expense of a quasi-linear computation cost of its own. Here, r stands for the number of states of the variables' domain associated with the sending function node (line 5).Then, $\left|\mathcal{V}_i\right|$ is the size of the array $\mathcal{V}_i$, hence log $\left|\mathcal{V}_i\right|$ is the time complexity to do the binary search on $\mathcal{V}_i$, which is required for their approach. On the other hand, our proposed approach GDPx, which works on the the Max-Product belief propagation algorithm, does not require performing binary search on $\mathcal{V}_i$ to find the cut-point. Therefore, the overall complexity of GDPx is $\mathcal{O}(r)$. This significantly indicates that GDPx can perform at the expense of a linear computation cost of its own.

x_1	x_2	x_3	F_1 ($\times 10^{-2}$)
R	R	R	2.841
R	R	G	3.968
R	R	B	2.982
R	G	R	2.630
R	G	G	2.958
R	G	B	3.405
R	B	R	4.226
R	B	G	3.498
R	B	B	2.653
G	R	R	3.146
G	R	G	2.630
G	R	B	2.606
G	G	R	3.639
G	G	G	4.391
G	G	B	3.522
G	B	R	3.076
G	B	G	3.733
G	B	B	3.522
B	R	R	4.555
B	R	G	3.827
B	R	B	2.841
B	G	R	3.639
B	G	G	4.109
B	G	B	3.898
B	B	R	6.668
B	B	G	5.893
B	B	B	5.142

Utility (cost) table for the function node $F_1(x_1, x_2, x_3)$



x_1 ($\times 10^{-2}$)	x_2 ($\times 10^{-2}$)	F_1 ($\times 10^{-2}$)	Product ($\times 10^{-4}$)	$R_{F_1 \to x_3}(x_3)$ ($\times 10^{-4}$)
6.203	8.423	2.841	1.484	
6.203	8.423	3.968	2.073	
6.203	8.423	2.982	1.558	
6.203	6.310	2.630	1.029	
6.203	6.310	2.958	1.158	
6.203	6.310	3.405	1.333	
6.203	4.713	4.226	1.236	
6.203	4.713	3.498	1.023	
6.203	4.713	2.653	0.776	
5.307	8.423	3.146	1.406	
5.307	8.423	2.630	1.176	
5.307	8.423	2.606	1.165	
5.307	6.310	3.639	1.219	{3.603, 3.027, 2.309}
5.307	6.310	4.391	1.470	
5.307	6.310	3.522	1.179	
5.307	4.713	3.076	0.769	
5.307	4.713	3.733	0.934	
5.307	4.713	3.522	0.881	
9.390	8.423	4.555	3.603	
9.390	8.423	3.827	3.027	
9.390	8.423	2.841	2.247	
9.390	6.310	3.639	2.156	
9.390	6.310	4.109	2.435	
9.390	6.310	3.898	2.309	
9.390	4.713	6.668	2.951	
9.390	4.713	5.893	2.608	
9.390	4.713	5.142	2.276	

Computation of $R_{F_1 \to x_3}(x_3)$

Fig. 2. (a) Computation of a factor-to-variable message (i.e. F_1 to x_3.)

$\mathcal{M}_1 = Q_{x_1 \to F_1}(x_1) = \{0.06203, 0.05307, 0.09390\}$

$\mathcal{M}_2 = Q_{x_2 \to F_1}(x_2) = \{0.08423, 0.06310, 0.04713\}$

$m = \max(\mathcal{M}_1) \times \max(\mathcal{M}_2) = 7.909 \times 10^{-3}$

$\mathcal{V}_i = \text{sortedVal}_R\big(F_1(x_1, x_2, x_3)\big) = \{6.668, 4.555, 4.226, 3.639, 3.639, 3.146, 3.076, 2.841, 2.630\}$

$p = \max(\mathcal{V}_R) = 6.668 \times 10^{-2}$; hence, $b = 9.390 \times 10^{-2} \times 4.713 \times 10^{-2} = 4.425 \times 10^{-3}$

$c = pb/m = 3.731 \times 10^{-2}$

$\Longrightarrow \mathcal{R}_R = [6.668 \times 10^{-2}, 3.639 \times 10^{-2}]$

That means, we have to look only for the rows within the range $\mathcal{R}_R$ which contains the first three rows of the table.

$Q_{x_1 \to F_1}(x_1)$ ($\times 10^{-2}$)	$Q_{x_2 \to F_1}(x_2)$ ($\times 10^{-2}$)	F_1 ($\times 10^{-2}$)	Product ($\times 10^{-4}$)	
9.390	4.713	6.668	2.951	
9.390	8.423	4.555	3.603	(max)
6.203	4.713	4.226	1.236	
5.307	6.310	3.639	1.219	
9.390	6.310	3.639	2.156	
5.307	8.423	3.146	1.406	
5.307	4.713	3.076	0.769	
6.203	8.423	2.841	1.484	
6.203	6.310	2.630	1.029	

Fig. 2. (b) Complete operation of GDPx on $R_{F_1 \to x_3}(x_3)$

Figure 2. Worked example of GDPx in computing a factor-to-variable message, F_1 to x_3 or $R_{F_1 \to x_3}(x_3)$, within the factor graph shown in Figure 1. In this example, for simplicity, we show that part of the original factor graph which is necessary for this particular message computation. In the figure, red, blue and green coloured values are used to distinguish the domain states R, B and G respectively for each of the variables involved in the computation, and arrows between the nodes of the factor graph are used to indicate the direction of the corresponding messages.

Lemma 1. During the function-to-variable message computation, the desired maximum value for a state $s_i \in S$ must always be found from the rows corresponding to the values ranging from p to c.

Proof. We prove this by contradiction. Assume there exists a row r_u that resides outside the range from which the maximum value for s_i can be found. That means:

$$p_u \prod_{k=1}^{n} \text{val}_{p_u}(M_k) > p \times b \tag{5}$$

where p_u is the local utility value for s_i which corresponds to the row r_u and $\text{val}_{p_u}(M_k)$ ($\forall k = 1, \ldots, \boldsymbol{n}$) is the corresponding value of p_u from the k^{th} incoming message of F_j. However, as the row r_u is outside the pruned range, we have $p_u < c$, or:

$$p_u < \frac{p \times b}{m} \tag{6}$$

From Equations 5 and 6, we have:

$$\frac{p \times b}{m} \prod_{k=1}^{n} \text{val}_{p_u}(M_k) > p \times b. \tag{7}$$

Replace m in Equation 7 by the product identified in Line 4 of Algorithm 1, we have:

$$\prod_{k=1}^{n} \text{val}_{p_u}(M_k) > \prod_{k=1}^{n} \max(M_k) \tag{8}$$

We can see that Equation 8 is false. Hence, there exists no such row as r_u.

4. Empirical Results

Given the detailed description in the previous section, we now empirically evaluate how much speed-up can be achieved using GDPx and compare this with the performance of G-FBP. In so doing, we run our experiments on corresponding factor graphs representing different instances of the benchmarking graph colouring problem. It is obvious from the discussion of Lemma 1 that our approach neither improves nor affects the solution quality of Max-Product; rather its sole objective is to reduce its computation cost while maintaining the same solution quality. Therefore, we focus on the computation aspect of the algorithm. More specifically, both the approaches, G-FBP and our proposed GDPx, intend to reduce the computation cost of the most expensive phase of the Max-Product belief propagation algorithm, that is the maximization operator. This particular operator, as discussed in Section 2, depends on two factors: i) domain size of the associated (i.e. neighbouring) variable nodes of the sending function nodes and ii) density of the factor graph, which can be apprehended from the values of arity/degree of the sending function nodes. Moreover, it is also observed from the literature that a pruning algorithm's performance often varies with the size of the problem setting [14, 18]. In light of the aforementioned discussion, we perform our experiments on varying these three parameters. Note that all of the experiments were performed on a simulator implemented in an Intel $i7$ Quadcore $i7$ GHz machine with 16 GB of RAM.

In our first experiment, as illustrated in Figure 3, we consider factor graphs having a number of function nodes randomly taken from the range 5 to 50, and that each of the factor graphs is generated by randomly connecting a number of variable nodes per function node. Specifically, this number of variable nodes connected to each function node, termed the arity n of a function node (i.e. density), is randomly

chosen from the range $[2-8]$. In Figure 3, we report the percentage of search space pruned by GDPx and G-FBP during the computation of function-to-variable messages as the values of the domain size of the variables (i.e. d) increases. Notably, G-FBP is based on an intuition that the maximum value can be found from the partially sorted top $cd^{\frac{n-1}{2}}$ values (see Section 1 for details). When this presumption is false, it incurs a significant penalty in terms of the computation cost (i.e. search space). Nevertheless, we always consider that their assumption is true while reporting the performance of G-FBP for all of the experiments in this paper. Moreover, the developers of G-FBP specifically admitted in [14] that the chosen value of the constant c can

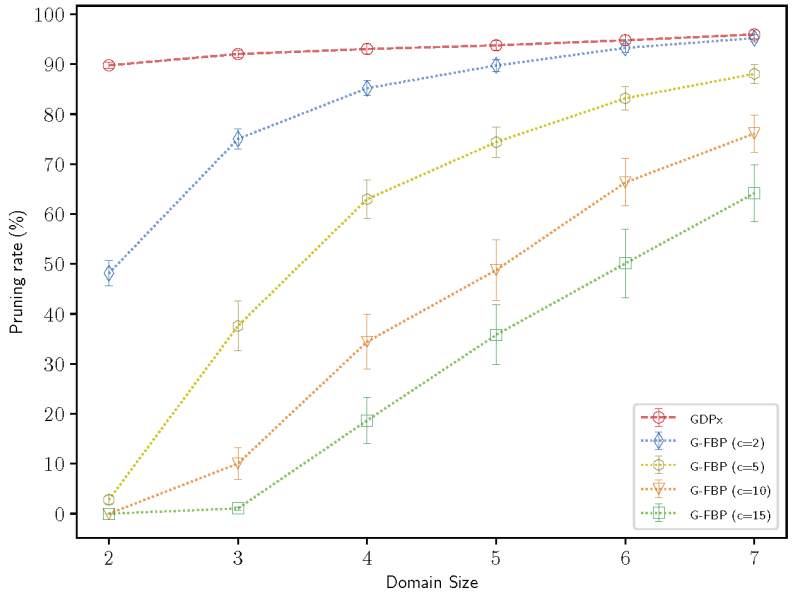


Fig. 3. Percentage of search space pruned as the domain size increases, GDPx vs G-FBP, for the factor graph representations of different instances of the graph colouring problem. Here the values of dependent variables, density and number of function nodes, are randomly taken from the ranges $[2,8]$ and $[5,50]$, respectively. In the figure, we use four different values of the constant c in evaluating the performance of G-FBP. Error bars are calculated using standard error of the mean.

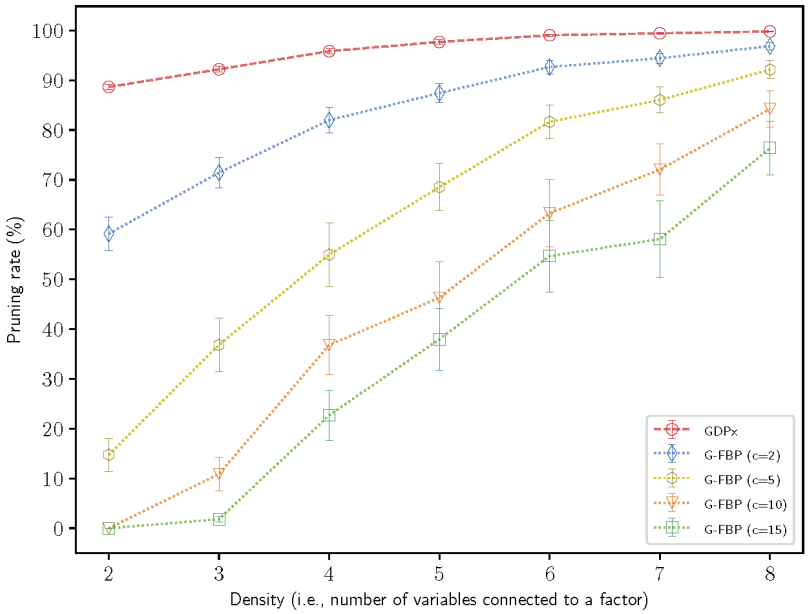


Fig. 4. Percentage of search space pruned as the density increases, GDPx vs G-FBP, for the factor graph representations of different instances of the graph colouring problem. Here the values of dependent variables, domain size and number of function nodes, are randomly taken from the ranges $[2,7]$ and $[5,50]$, respectively. In the figure, we use four different values of the constant c in evaluating the performance of G-FBP. Error bars are calculated using standard error of the mean.

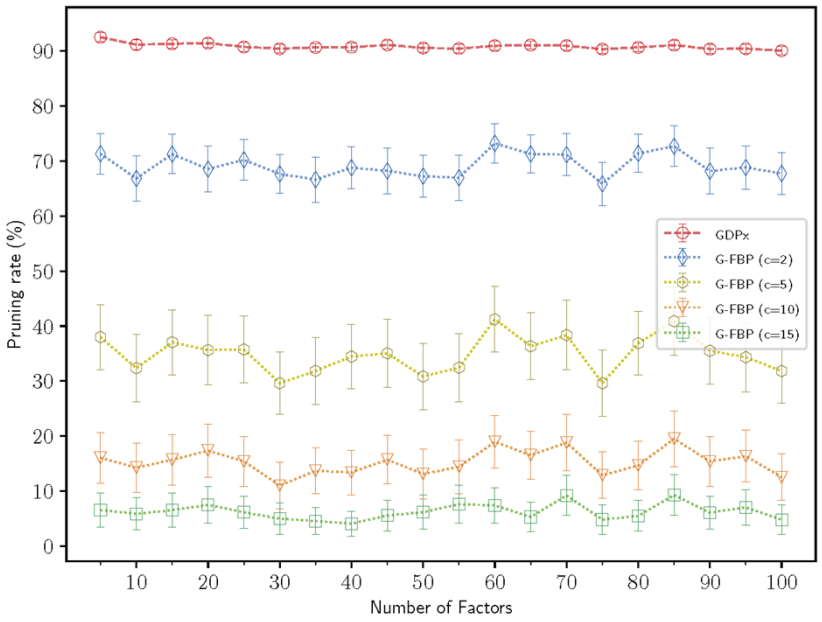


Fig. 5. Percentage of search space pruned as the number of function node increases, GDPx vs G-FBP, for the factor graph representations of different instances of the graph colouring problem. Here the values of dependent variables, domain size and density, are randomly taken from the ranges $[2,7]$ and $[2,8]$, respectively. In the figure, we use four different values of the constant c in evaluating the performance of G-FBP. Error bars are calculated using standard error of the mean.

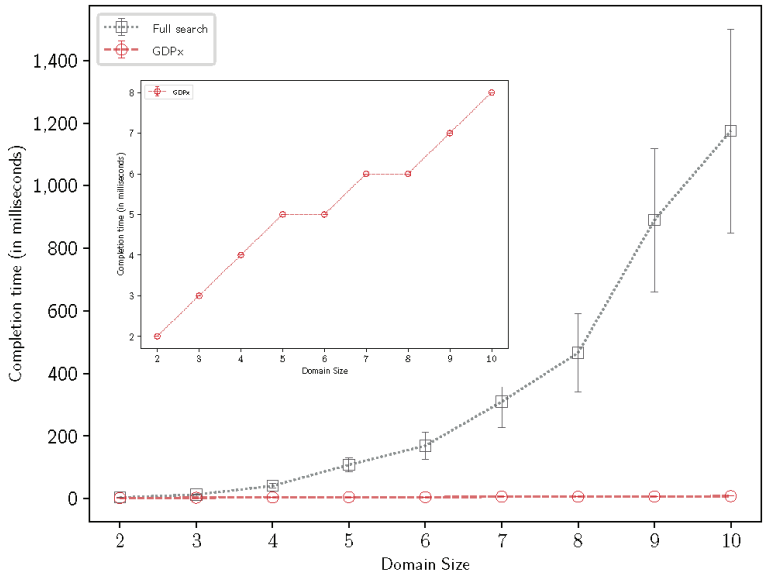


Fig. 6. Completion time (single message) of GDPx compared to full-search in the Max-Product algorithm. In this experiment, we use the same setting as Figure 3. Error bars are calculated using standard error of the mean.

make notable difference in its overall performance, and hence they discussed the significance of a range of values of c. By taking their observation into account, we consider 4 values (i.e. $2,5,10$ and 15) of c for all of the experiments presented in this paper. It is worth noting that the local utility tables (i.e. probability distribution tables) for the function nodes of a factor graph are generated randomly. Now, to get the results based on the aforementioned setting, we initially compute the percentage of the search space pruned (i.e. speed-up) by the algorithms for a function node by taking the average of the speed-ups of all the messages sent by that function node. Afterwards, we take the average of the speed-ups of all the nodes in a factor graph. Finally, we report the results of each factor graph averaged over 100 test runs in Figure 3, recording standard errors to ensure statistical significance.

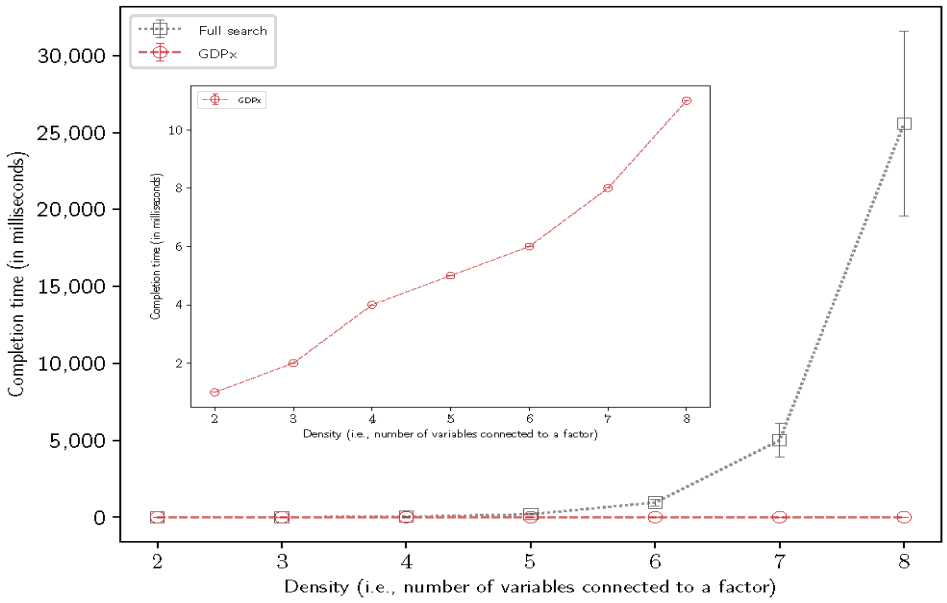


Fig. 7. Completion time (single message) of GDPx compared to full-search in the Max-Product algorithm. In this experiment, we use the same setting as Figure 4. Error bars are calculated using standard error of the mean.

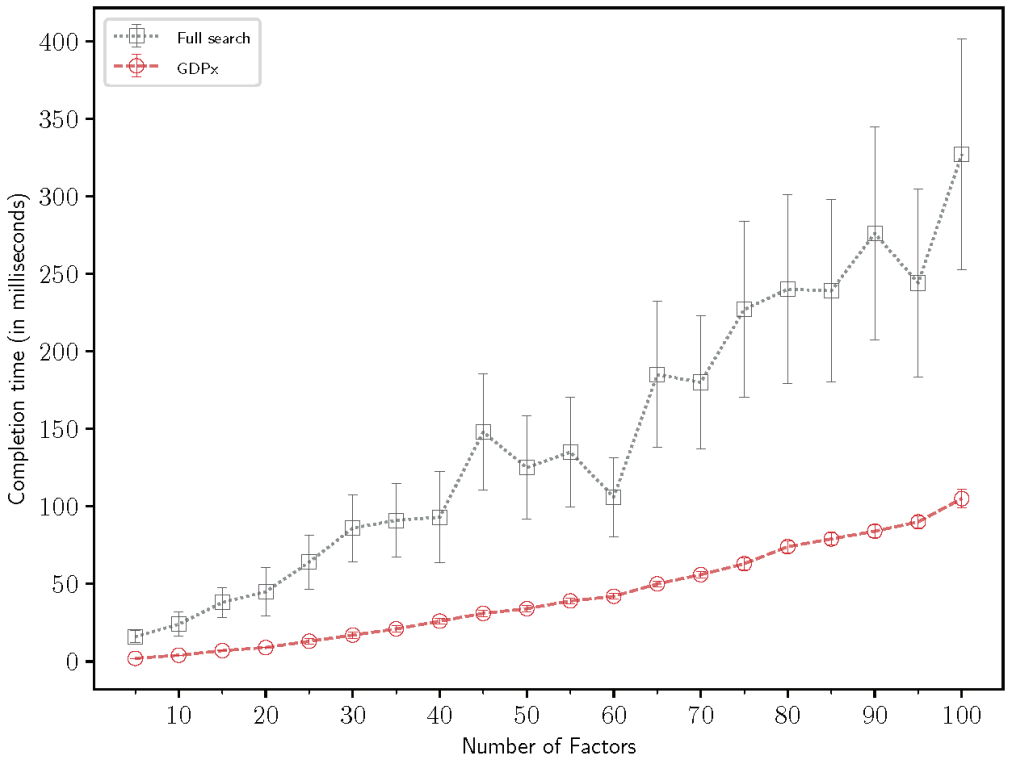


Fig. 8. Completion time (single message) of GDPx compared to full-search in the Max-Product algorithm. In this experiment, we use the same setting as Figure 5. Error bars are calculated using standard error of the mean.

In Figure 3, the green line illustrates the performance of G-FBP with the value of $c = 15$. For this setting, it can be seen from the trend of the line that G-FBP's performance is only notable for domain size 4 or more. To be exact, this algorithm prunes around 18% of the search space during the computation of the maximization operation when $d = 4$. While in the same setting G-FBP reduces around $35 - 64\%$ of the search space for the domain size $5 - 7$. Although G-FBP's performance is slightly better with c value 10, the trend is similar to the previous one (orange line). Notably, its

pruning rate is around $48-75\%$ for the value of the domain size $5-7$. On the other hand, as depicted in the orange line of Figure 3, for $c=5$, G-FBP's pruning rates are around 37%, 60%, 74% and 85% for the d values $3,4,5$ and 6, respectively (light-yellow line). The pruning rate reaches the maximum of 88% with $d=7$ for this setting. Nevertheless, it is clear from the results shown in blue line that G-FBP performs even better with lower values of c (i.e. $c=2$). To be precise, it reaches at its peak with around 95% pruning rate for $d=7$. Having stated that, there is no theoretical guarantee that G-FBP will always provide the above performance, which as aforementioned, are generated considering their presumption always true. In this context, [18] shows that due to this phenomenon (i.e. the lack of theoretical guarantee), G-FBP produces severely inconsistent performance. Despite this issue, we consider such to show how our proposed GDPx performs compared to the best possible (although unrealistic) performance of the state-of-the-art algorithm, G-FBP. Significantly, the red line of Figure 3 illustrates that GDPx always performs better than all the versions of G-FBP. Similar to what we observed from the trend of G-FBP's results, the performance GDPx is better when the variables take their values from a larger domain size, given that the other parameters remain identical. Nevertheless, unlike G-FBP, GDPx prunes around 90% of the search space for values of d as small as 2. Overall, the pruning rate of GDPx always lies within the range of $90-96\%$, and is correspondingly better than any version of the G-FBP algorithm. Note that neither all the nodes, nor all the function-to-variable messages experience similar performance from the proposed approach, due to their differences in the content of the utility tables and incoming messages.

Figure 4 illustrates the comparative performance of GDPx and G-FBP (four versions) for factor graphs representing different instances of the graph colouring problem with function nodes' density/arity (i.e. n) ranging from 2 to 8. Similar to the previous experimental setting, we consider factor graphs having a number of function nodes randomly taken from the range 5 to 50. However, we report the pruning rate while increasing the value of n to observe how the algorithms perform for the factor graphs with different density. Here, we randomly choose the values of the variables' domain size from the range $[2,7]$. Finally, we report the results of each factor graph averaged over 100 test runs and record standard errors to ensure statistical significance. It is observed from Figure4 that GDPx prunes at least 88% or more of the search space during the computation of the maximization operation for the Max-Product algorithm (red-line). Surprisingly, GDPx's pruning rate reaches around 99% for the factor graphs with n's value 6, 7 or 8 for this setting. This trend coupled with the previous experiment's observation is remarkably important because it gives us a clear indication that GDPx is able to prune the maximum amount of search space when the values of n and d becomes larger. On the other hand, even the best-case of the G-FBP algorithm never outperforms GDPx, though its performance is getting better with the lower value of c. However, it is worth noting that with a lower value of c there is a higher possibility that their presumption is false, which would force G-FBP to consider the full search space again.

The final experiments for the metric pruning rate is shown in Figure 5 which reports the performance of GDPx and G-FBP (four versions) as the number of function node increases from as small as 5 to the maximum 100. For this experiment, the values of dependent variables, domain size d and nodes' density n, are randomly taken from the ranges $[2,7]$ and $[2,8]$, respectively. Similar to the previous two experiments, we initially compute the percentage of the search space pruned by the algorithms for a function node by taking the average of the speed-ups of all the messages sent by that function node. Then, we take the average of the pruning rate (%) of all the nodes in a factor graph. Finally, we report the results of each factor graph averaged over 100 test runs in Figure 5, recording standard errors to ensure statistical significance. On the one hand, the best case of G-FBP (i.e. $c=2$) prunes around $67-70\%$ of the search space (blue-line) in this particular experiment. On the other hand, GDPx prunes around $90-92\%$ of the search, and more importantly in a steady rate. In addition, for all three of these experiments, we run the one-way ANOVA with post-hoc Tukey HSD test. While doing so, we consider GDPx, G-FBP ($c=2$), G-FBP($c=5$), G-FBP ($c=10$) and G-FBP ($c=15$) as treatments, each of which illustrates the percentage of the search space pruned. For each experiment, the observed p-value corresponding to the F-statistic of one-way ANOVA is lower than 0.05, suggesting that the one or more treatments are significantly different. Subsequently, we employ a post-hoc test (Tukey HSD) that also suggests that the performance of GDPx is significantly different from each of the remains, individually (i.e. $p<0.01$).

When taken together the above empirical results, it is obvious that GDPx significantly reduces the computation cost of the maximization operator of the Max-Product algorithm by reducing the search space upon which the maximization operator acts on. It can also be claimed based on the theoretical analysis that GDPx does not compromise on the solution quality in doing so. Whereas its counterpart G-FBP cannot provide this theoretical guarantee. Moreover, GDPx reduces more of the search spaces compared to the best cases of all the versions of G-FBP, and this is true for all the cases that we have considered in our empirical evaluation. Now, in our final set of experiments, we examine what does this reduction of search space actually mean in reducing the completion time (Figures 6, 7, 8). To do so, we

report the time GDPx takes to compute a single function-to-variable message (i.e. the completion time), and compare this with the completion time of a single message though full-search. Note that, since the search space obtained by G-FBP is always larger than what is achieved by GDPx, its completion time can never be smaller than GDPx. We therefore to avoid redundancy did not consider G-FBP in this particular experiment.

Specifically, Figure 6 illustrates the performance gain of GDPx in terms of completion time (red line), and compare this to the standard way of computing the maximization operation (dotted-black line) in a single function-to-variable message for the same experimental setting (i.e. the values of arity and the number of function nodes) used in Figure 3. To report the result for each domain size d, we take the average of 100 different observations and record standard errors to ensure statistical significance. It can be seen from Figure 6 that GDPx saves 50% and 75% completion time of the full search when domain sizes are 2 and 3, respectively. Then, the impact of GDPx is getting larger with the value of d. To be precise, the performance gain reaches around 90.25% to 99.32% for the domain size of the variable nodes 3 to 10 in this setting.

In Figure 7, the same metrics (i.e. the completion time) is considered to measure GDPx's performance. However, the experimental setting for this experiment is identical to what we considered in Figure 4. It can be observed from the trend of comparative results depicted in the figure that GDPx consumes 50% to 90% less time than its counterpart for the value of n (i.e. number of variables connected to a function node) 3 to 10. Notably, the trend is identical to the previous experiment, and this is important because it gives us a clear indication that GDPx is able to reduce the maximum amount of completion time when the values of n and d becomes larger.

Finally, using the same experimental setting as depicted in Figure 5, Figure 8 illustrates the completion time of a single message through GDPx as well as with the standard approach. We do this to observe how GDPx performs for different size of factor graphs. Notably, GDPx experiences 67% to 88% reduction of the completion time in this setting. More importantly, it is obvious from the figure that GDPx performance does not affected by the increase of the problem size, instead it gets better for larger settings. Moreover, it is worth noting from the results depicted in Figures6, 7 and 8 that GDPx's own runtime is also negligible. This is expected because from its complexity analysis we find that only a linear time is required to execute the proposed GDPx algorithm (see Section 3).

5. Conclusion and Future Work

In this paper, we tailor the GDP algorithm and develop a new algorithm GDPx that is capable of significantly reducing the computation cost of the maximization operator of Max-Product algorithm. Our extensive empirical evidence observes a significant reduction of search space, ranging from around 85% to 99% by using this technique. We demonstrate that the relative performance gain of GDPx improves with increasing the domain size of the variables and the arity of the constraint functions on which the maximize operator acts. These findings also serve as empirical proof for scaling up. In the future, we intend to study whether branch and bound based FDSP can be used effectively to accelerate Max-Product in real-world applications and to compare its performance to that of our GDPx.

Acknowledgement

This paper builds on our previous work presented at the Seventeenth International Conference on Autonomous Agents and Multiagent Systems, held in Stockholm, Sweden, from July 10-15, 2018 [18]. This work is primarily funded by the Centennial Research Grant (CRG) of University of Dhaka.

References

1. J. Pearl, Probabilistic reasoning in intelligent systems: Networks of plausible inference (*representation and reasoning*), 1988.
2. J. Sun, N.-N. Zheng, H.-Y. Shum, Stereo matching using belief propagation, *IEEE Transactions on pattern analysis and machine intelligence*, 25 (7), 787–800, 2003.
3. M. P. Fossorier, M. Mihaljevic, H. Imai, Reducecomplexity iterative decoding of low-density parity check codes based on belief propagation, *IEEE Transactions on communications*, 47 (5), 673–680, 1999.
4. A. Farinelli, A. Rogers, A. Petcu, N. R. Jennings, Decentralised coordination of low-power embedded devices using the max-sum algorithm, in: *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Vol. 2, pp. 639–646, 2008.
5. R. Dechter, Reasoning with graphical models (2007).
6. F. R. Kschischang, B. J. Frey, H. Loeliger, Factor graphs and the sum-product algorithm, IEEE Transactions on Information Theory, 47 (2), 498–519, 2001.
7. A. R. Leite, F. Enembreck, J. A. Barth`es, Distributed constraint optimization problems: *review and perspectives, Expert Systems with Applications*, 41 (11), 5139–5157, 2014.
8. F. Fioretto, E. Pontelli, W. Yeoh, Distributed constraint optimization problems and applications: *A survey, Journal of Artificial Intelligence Research*, 61, 623–698, 2018.
9. K. P. Murphy, Y. Weiss, M. I. Jordan, Loopy belief propagation for approximate inference: An empirical study, in: *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence, Morgan Kaufmann Publishers Inc*., pp.467–475, 1999.

10. P. F. Felzenszwalb, D. P. Huttenlocher,Efficient belief propagation for early vision, *International journal of computer vision*, 70 (1), 41–54, 2006.
11. A. Farinelli, A. Rogers, N. R. Jennings, Agent-based decentralised coordination for sensor networks using the max-sum algorithm, *Autonomous agents and multi-agent systems*, 28 (3), 337–380, 2014.
12. S. M. Aji, R. McEliece, The generalized distributive law, *IEEE Transactions on Information Theory*, 46 (2), 325–343, 2000.
13. M. M. Khan, L. Tran-Thanh, S. D. Ramchurn, N. R. Jennings, Speeding up gdl-based message passing algorithms for large-scale dcops, The Computer Journal, 61 (11), 1639 –1666, 2018.
14. Y. Kim, V. Lesser, Improved max-sum algorithm for dcop with n-ary constraints, in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 191–198, 2013.
15. S. D. Ramchurn, A. Farinelli, K. S. Macarthur, N. R. Jennings, Decentralized coordination in robocup rescue, *Computer Journal*, 53, 1447–1461, 2010.
16. K. S. Macarthur, R. Stranders, S. D. Ramchurn, N. R. Jennings, A distributed anytime algorithm for dynamic task allocation in multi-agent systems, in: *Proceedings of the 25th AAAI Conference on Artificial Intelligence*, pp. 701–706, 2011.
17. R. Stranders, A. Farinelli, A. Rogers, N. R. Jennings, Decentralised coordination of mobile sensors using the max-sum algorithm, in: *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, Vol. 9, pp. 299–304, 2009.
18. M. M. Khan, L. Tran-Thanh, N. R. Jennings, A generic domain pruning technique for gdl-based dcop algorithms in cooperative multi-agent systems, in: *Proceedings of the 17th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), IFAAMAS, Stockholm, Sweden*, pp. 1595–1603, 2018.
19. Z. Chen, X. Jiang, Y. Deng, D. Chen, Z. He, A generic approach to accelerating belief propagation based incomplete algorithms for dcops via a branch-and-bound technique, in: *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33, pp. 6038–6045, 2019.