Improved Max-Sum Algorithm For DCOP with n-ary Constraints

Yoonheui Kim
University of Massachusetts at Amherst, MA
01003, USA
ykim@cs.umass.edu

Victor Lesser
University of Massachusetts at Amherst, MA
01003, USA
lesser@cs.umass.edu

ABSTRACT

Many distributed constraint optimization (DCOP) algorithms include nodes' local maximization operation that searches for the optimal variable assignment in a limited context. When the variable domain is discrete, this operation is exponential in the number of associated variables and thus computationally challenging. McAuley's recent work on efficient inference implements this maximization operator such that in most cases only a small set of values is examined without loss of accuracy. We increase the applicability of such approach to DCOP in the three following ways. First, we extend it to non-pairwise graphs with better computational expected complexity. Second, we remove the requirement for offline sorting, which often is not realistic in many DCOP domains, while keeping the same complexity. Third, we provide a correlation measure to determine dynamically the appropriate cases to apply the technique since its efficiency is sensitive to characteristics of the data sets. We combine this technique with the Max-Sum algorithm and verify empirically that our approach provides significant time savings over the standard Max-Sum algorithm.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems, Coherence and coordination*

General Terms

Performance, Algorithms

Keywords

Distributed Problem Solving, DCOP, Max-Sum

1 Introduction

Distributed constraint optimization (DCOP) [13] is a popular framework for cooperative multi-agent systems where agents try to maximize the system utility given constraints. When the system contains many constraints each of which involves a subset of variables in the system, the agent decision making problem can be straightforwardly represented as a DCOP. It has been applied to many real application domains with such characteristics such as sensor networks [21, 5], meeting scheduling [10], and traffic control [6].

Despite its recent popularity, algorithms on DCOPs have been focused mainly on problems with binary constraints [15, 13, 11]. Although problems with n-ary constraints can be reformulated into the ones with binary constraints, an exponential number of binary constraints and many new variables need to be introduced as in [10]. Additionally, there is an overhead associated with this mapping. Despite the importance of n-ary constraints in real applications, existing research on DCOP with n-ary constraints [14, 21, 3, 19] has not directly tackled the computational difficulty in handling n-ary constraints in DCOPs. We tackle this problem in this paper in the context of the Max-Sum algorithm.

In the DCOP, each agent is responsible for setting its variable value. Further, agents have a local view of the graph, which includes only their immediate neighbors. Therefore, message-passing approaches such as Max-Sum are ideally suited for such multiagent coordination problems [3, 16]. Max-Sum performs repetitive maximization operations for each constraint to select the locally best configuration of the associated variables given the local function and a set of incoming messages. The complexity of this step grows exponentially as the number of associated variables (constraint arity) grows. There have been approaches that try to reduce the complexity in the context of Max-Sum. [16] reduces domain sizes of variables associated with constraint functions for task allocation domains where agents' action choices are strictly divided into working on a task or not. [18] performs a branch and bound search with constraint functions that the upper and lower bound can be evaluated with a subset of variable values. However, these techniques require domain characteristics that limit their applicability.

Our main contributions lie in addressing the computational bottleneck without imposing any restriction on constraint characteristics and also in providing formal guarantees regarding expected runtime improvement, which could be very significant, achieving up to an exponential improvement over the standard scheme. Reducing such computational overhead is particularly crucial in practical multiagent settings, where agents are often assumed to be resource constrained such as mobile robots [18, 5].

McAuley et al. [12] recently provided an efficient technique for finding the maximum sum of lists based on offline sorting. Their technique, called Fast Belief Propagation (FBP), finds the maximum sum of particular variable configurations in two sorted lists of length N with an expected complexity of $O(\sqrt{N})$. Assuming order statistics of variables on the lists

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013),* Ito, Jonker, Gini, and Shehory (eds.), May 6–10, 2013, Saint Paul, Minnesota, USA.

are independent, the technique achieves an expected time of $O(N\sqrt{N})$ to compute a single Max-Sum message for a binary constraint, which is smaller than $O(N^2)$ required by the standard approach.

Although the FBP technique offers substantial computational benefit on DCOPs with binary constraints, it cannot be directly used on graphs with n-ary constraints. Based on their theoretical analysis on n-ary constraint functions, the benefit decreases as the constraint arity increases. Additionally, order statistics of lists summed in the scheme should be positively correlated or independent for the theoretical analysis to hold. However, this property easily gets violated in domains where a variable's value can affect multiple constraint function values in opposite ways. When order statistics of these lists are negatively correlated, the technique may perform worse than a simple technique using dynamic programming. Lastly, the graph is required to be given offline for computational savings. Often, a DCOP is a one-shot problem in which an expensive preprocessing sorting step that dominates the actual problem complexity is not realistic.

To remedy these limitations, we have developed a variant of McAuley's technique, which we call Generalized Fast Belief Propagation (G-FBP). Our approach is fundamentally different from that of FBP in that it does not require the offline and complete sorting of different data structures as in FBP; rather it uses *partially sorted* lists. The key idea behind our approach is that often, only a small, representative sample of values from different message/value lists is needed to efficiently perform the maximization procedure. Further, our approach works for arbitrary arity graphs as opposed to pairwise graphs required by the FBP algorithm [12]. We also provide expected runtime complexity analysis for this general case and show that we can indeed achieve $O(N\sqrt{N})$ complexity for pairwise graphs with only partially sorted lists. For m-ary graphs, this translates into an expected complexity of $O(mN^{\frac{m+1}{2}})$ as opposed to the exhaustive approach's complexity of $O(mN^m)$, which is a significant reduction. We also note that an advanced version of FBP is presented in [2], which has a theoretical expected complexity of $O(mN^{\frac{(m-1)^2}{m}+1})$ for general m-ary graphs. Our approach has strictly better expected complexity.

Additionally, we have devised a correlation measure which decides whether the order statistics of items on the lists are negatively correlated. We then use this measure to conditionally apply the G-FBP scheme to a particular maximization operation. Given the definition of correlation on order statistics, we show that this measure correctly computes the correlation.

Finally, we add another feature to our approach, which leads to an extended version of G-FBP called GSC-FBP. This approach reuses computation from the previous iterations results. Its effectiveness lies in the fact that messages become less likely to change in later stages of the algorithm.

This paper is structured as follows. In Section 2, we give an overview of DCOPs and the Max-Sum algorithm and in Section 3, an overview of FBP approach. Next, in Section 4, we formulate the G-FBP approach and provides a condition for the expected complexity. In Section 5, we propose a correlation measure which determines the applicability of G-FBP technique on certain data sets. Finally, we summarize the key results and discuss future work in Section 6.

2 Distributed Constraint Optimization and Max-Sum Algorithm

A distributed constraint optimization algorithm(DCOP) is formally defined by the following parameters:

- A set of variables $\mathbf{X} = \{X_1, \ldots, X_r\}$, where each variable has a finite domain (maximum size N) of possible values that it can be assigned.
- A set of constraint functions $\mathbf{F} = (F_1, \ldots, F_k)$, where each constraint function, $F_j : \mathbf{X}_j \rightarrow \Re$, takes as input any setting of the variables $\mathbf{X}_j \subseteq \mathbf{X}$ and provides a real valued utility.

In DCOP, we assume that each variable x_i is owned by an agent and that an agent only knows about the constraint functions in which it is involved. The DCOP can be represented using a *constraint network*, where there is a node corresponding to each variable x_i. There is an edge (hyperedge) for each constraint F_j that connects all variables that are involved in the function F_j.

The objective in the DCOP is to find the complete variable configuration $\mathbf{x}$ that maximizes $\sum_{F_j \in \mathbf{F}} F_i(\mathbf{x}_i)$.

Max-Sum [3] is a message-passing DCOP algorithm belonging to the class known as Generalized Distributive Law (GDL) [1]. Max-Sum is a simple variation of the Max-Product algorithm where the global utility function is maximized. Max-Sum produces the optimal solution in an acyclic graph or a good approximate solution in a cyclic graph. [20]

The Max-Sum algorithm iteratively performs message-passing on the factor graph [8] corresponding to the DCOP. In this representation, there is a variable node for each variable and a factor node for each constraint function. A function node is connected to a variable node if the corresponding constraint function contains that variable in its domain. The messages exchanged in Max-Sum are of two types:

The message $q_{i\rightarrow j}$ **from Variable** i **to Function** j **:**

$$q_{i\rightarrow j}(x_i) = \alpha_{ij} + \sum_{k \in M_i \setminus j} r_{k\rightarrow i}(x_i) \qquad (1)$$

α_{ij} is a scalar set such that $\sum_{x_i} q_{i\rightarrow j}(x_i) = 0$, and M_i contains the indices of function nodes connected to variable node i.

The message $r_{j\rightarrow i}$ **from Function** j **to Variable** i**:**

$$r_{j\rightarrow i}(x_i) = \max_{\mathbf{x}_j \setminus i} \left[F_j(\mathbf{x}_j) + \sum_{k \in N_j \setminus i} q_{k\rightarrow j}(x_k) \right], \qquad (2)$$

where N_j contains the indices of variable nodes connected to the function node j in the factor graph.

The maximization operator in Eq. (2) is the most computationally expensive operation in the Max-Sum algorithm. For an m-ary DCOP, where the constraint function F_j has m variables in its domain, the complexity of computing the above message r is $O(N^m)$, where N is the maximal domain size of any variable. Therefore, optimizing this maximization operation is the focus of our work.

3 Fast Belief Propagation

The Fast Belief Propagation (FBP) [12] optimizes the maximization operator of Eq. (2) by using presorted constraint functions. Given a binary constraint, it uses two lists–a list of presorted constraint function values and a list of incoming message values that are sorted online. This operation

amounts to maximizing the sum of two lists $\mathbf{v}_a$ and $\mathbf{v}_b$:

$$\max_i \{\mathbf{v}_a[i] + \mathbf{v}_b[i]\} \quad (3)$$

The FBP algorithm performs the above operation with an expected $O(\sqrt{N})$ time complexity, instead of the simple $O(N)$ time algorithm, where N is the list size.

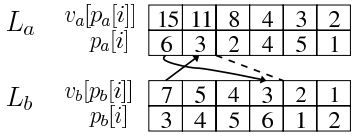


Figure 1: Example of FBP technique. The largest item 15 of $\mathbf{v}_a$ that has index 6 is summed with 3 in $\mathbf{v}_b$ with the same index (which maps to specific value combination of variables). Therefore, items with value smaller than 3 in $\mathbf{v}_b$ can be ignored as any value smaller than 3 cannot yield a value larger than $(15+3)$. We also limit the items smaller than 11 in $\mathbf{v}_a$ by applying the same idea. In this example only 2 computations are required to compute the maximum value using this technique.

Fig. 1 describes the main idea of the FBP algorithm. The list $\mathbf{p}_a$ and $\mathbf{p}_b$ denote the permutation that sort $\mathbf{v}_a$ and $\mathbf{v}_b$. For further details, please refer to [12].

As the expected computational complexity to find the maximum of two such lists is $O(\sqrt{N})$, the FBP algorithm achieves the total expected complexity of $O(N^{1.5})$ for the Max-Sum maximization operation, that is better than $O(N^2)$ time required by the standard implementation. The main drawback of the FBP approach is that the FBP approach requires the complete problem to be specified ahead of time as it requires presorting of constraint functions. Further, this approach is only applicable to pairwise graphs and the runtime guarantees do not extend to arbitrary arity graphs.

4 G-FBP

We now present a new maximization operation, G-FBP that uses *two partially* sorted lists, to find the maximum sum as in Eq. (2), instead of completely sorted lists used in FBP. This technique has the expected complexity of $O(\sqrt{N})$ for lists of size N given the condition on the size of the sorted parts of the lists. We begin with construction of partially sorted lists, present the G-FBP technique and then discuss the relation between the computational complexity and the length of the sorted parts of these lists.

4.1 Partial List Construction

In this section, we describe the two partially sorted lists used in our approach called the *value list* and the *message list*. We select and sort only the top $KN^{\frac{m-1}{2}}$ items of both lists where N is the domain size, m is the number of associated variables and K is a constant.

The main intuition behind such a select-then-sort operation is that for the maximization operation, only top $KN^{\frac{m-1}{2}}$ items will be accessed most of the time; unsorted entries are not accessed in most cases. *Partial sorting* and using a single *message* list are keys to generalizing the approach to n-ary constraints while keeping the same complexity. Using a combined message list allows the algorithm to have the same expected complexity of $O(\sqrt{N})$ as in FBP technique on binary constraints where N is the total length of the list.

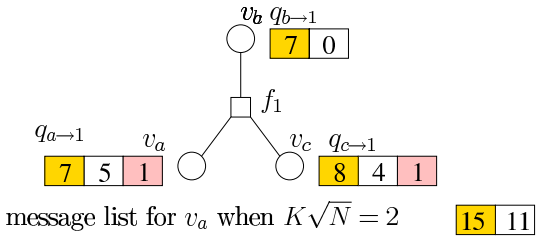


Figure 2: Example of Message List Generation. Each value in boxes denotes individually sorted message values from the variable nodes to the function nodes. The domain size is 2 for v_b and it is 3 for v_a and v_c thus the message size. In order to compute the message to v_a, messages from v_b and v_c ($q_{b\to1}$ and $q_{c\to1}$ respectively) are summed to generate the partial message list $[15, 11]$ instead of the complete list $[15, 11, 8, 8, 4, 1]$ with $|L_a| = 2$.

Partial sorting enables construction of this message list during runtime without any condition on N and m. We discuss the complexity of this operation later in this section.

Value list: Intuitively, the value list corresponds to a partially sorted version of the constraint function F_j given the specific value of a single variable in Eq. (2). There is one value list defined for every constraint function F_j and every value of variable x_i that is in the scope of F_j. It contains $\langle$index-value$\rangle$ pairs as:

$$L_b(j, x_i) = \{\langle \mathbf{x}^j, F_j(\mathbf{x}^j)\rangle | \mathbf{x}^j(i) = x_i\} \quad (4)$$

where $\mathbf{x}^j$ is a complete assignment to all the variables in the scope of function F_j; the condition $\mathbf{x}^j(i) = x_i$ implies that the variable x_i is fixed to a particular value in every $\mathbf{x}^j$. If the constraint function is m-ary (or involves m variables), then the length of each value list is N^{m-1}. Instead of completely sorting this list, which is expensive, we select the $KN^{\frac{m-1}{2}}$ largest values of $L_b(j, x_i)$, which are then sorted in decreasing order and inserted back to the front of this list.

Note that, the selection of top l items from a list of size N can be performed in $O(N)$ time using the standard selection algorithm, followed by a pivoting operation. Therefore, selecting and sorting $KN^{\frac{m-1}{2}}$ items never exceeds the complexity of iterating over all values and is much cheaper than sorting the complete list depending on the value of K. Further, such partially sorted lists can be constructed once per execution of the entire algorithm.

Message list: Intuitively, the message list represents a partially sorted list corresponding to the sum of incoming messages q to a function node, as shown in the second term of Eq. (2). There is one message list defined for every constraint function F_j and every variable x_i(Not every value of variable) that is in the scope of F_j. It contains $\langle$index-value$\rangle$ pairs as:

$$L_a(j, X_i) = \{\langle \mathbf{x}^j \backslash x_i, \sum_{k \in N_j \backslash i} q_{k\to j}(\mathbf{x}^j(k))\rangle\} \quad (5)$$

where $\mathbf{x}^j(k)$ denotes the assignment to the variable x_k under $\mathbf{x}^j$. The length of every complete list L_a is N^{m-1} and selecting the top most items requires iterating over all values. Fortunately, each message contains values on a single neighboring variable and are independent of each other in Max-Sum. Using the independency among messages, we

Algorithm 1 G-FBP(v_a, v_b) : Find $max(v_a[i] + v_b[i])$

Require: permutation array p_a and p_b that partially sort v_a and v_b in decreasing order(i.e $p_a[i]$ is the index of ith largest value of v_a.

1: {Initialization}
2: $S_b^{miss} \leftarrow \phi, S_a^{miss} \leftarrow \phi, val_{max} \leftarrow -\infty$
3: $end_a \leftarrow len(p_a), end_b \leftarrow len(p_b)$
4: $itr_a \leftarrow 0, boundfound \leftarrow false$
5: **if** $(p_a[1] \in p_b) \wedge (p_b[1] \in p_a)$ **then**
6: $index_{max} \leftarrow argmax_{i \in \{p_a[1], p_b[1]\}}\{v_a[i] + v_b[i]\}$
7: $val_{max} \leftarrow v_a[index_{max}] + v_b[index_{max}]$
8: $end_a \leftarrow p_a^{-1}[p_b[1]], end_b \leftarrow p_b^{-1}[p_a[1]]$
9: $boundfound \leftarrow true$
10: **end if**
11: **while** $itr_a < end_a$ {Until bounding items are found} **do**
12: $itr_a \leftarrow itr_a + 1$
13: **if** $p_a[itr_a] \notin p_b$ **then**
14: $S_b^{miss} \leftarrow S_b^{miss} \vee \{p_a[itr_a]\}$
15: **else**
16: $boundfound \leftarrow true$
17: **if** $v_a[p_a[itr_a]] + v_b[p_a[itr_a]] > val_{max}$ **then**
18: $index_{max} \leftarrow p_a[itr_a]$
19: $val_{max} \leftarrow v_a[index_{max}] + v_b[index_{max}]$
20: **end if**
21: **if** $p_b^{-1}[p_a[itr_a]] < end_b$ **then**
22: $end_b \leftarrow p_b^{-1}[p_a[itr_a]]$
23: **end if**
24: **end if**
25: **end while**
26: repeat 11-25 while interchanging a and b
27: {Process unmatched items by directly calculating from the constraint function and messages}
28: **for all** $i \in S_a^{miss} \vee S_b^{miss}$ **do**
29: compute the value $v_a[i] + v_b[i]$ by going through the value table and messages and update val_{max}.
30: **end for**
31: {failure case}
32: **if** $boundfound == false$ **then**
33: process unsorted part of the list and update val_{max}
34: **end if**
35: return val_{max}

Algorithm 2 compute $r_{j \to i}$ in Eq. (2) with G-FBP

1: $messagechanged = false$
2: **if** cycle==0 **then**
3: **for all** $x_i \in X_i$ **do**
4: construct $L_b(j, x_i)$
5: **end for**
6: **end if**
7: **for all** $k \in N_j \setminus i$ **do**
8: **if** $q_{k \to j}$ has changed **then**
9: $messagechanged = true$
10: **end if**
11: **end for**
12: **if** $messagechanged == true$ **then**
13: construct $L_a(j, X_i)$
14: **end if**
15: **for all** $x_i \in X_i$ **do**
16: $r_{j \to i}(x_i) = G - FBP(L_b(j, x_i), L_a(j, X_i))$
17: **end for**
18: return $r_{j \to i}$

do not iterate the items in the list L_a completely in order to select the top elements. In our implementation, we incrementally construct each message list partially that only contains the top $KN^{\frac{m-1}{2}}$ items sorted in descending order, without ever generating the complete N^{m-1} sized lists. Although these message lists are constructed per iteration unlike value lists, there are only m such lists for each m-ary constraint in constrast to $(m \times N)$ value lists. The overhead of constructing message lists for a single message is $O(\log mK \times N^{\frac{m-1}{2}})$ and this does not dominate the expected complexity $O(N \times N^{\frac{m-1}{2}})$ of computing a message for an m-ary constraint for a reasonable K. Fig. 2 shows an example of a partial message list.

4.2 The G-FBP Algorithm With Partial Lists

We now describe the complete steps of the G-FBP maximization procedure that operates using such partial value and message lists where the ranks of items in unsorted part are not known. For ease of exposition, we describe the Alg. 1 in terms of the maximization problem in Eq. (3). The main difference between G-FBP and FBP is that G-FBP uses partially sorted lists $\mathbf{v}_a$ and $\mathbf{v}_b$ are partially sorted. Thus, we need to process items whose matching items are not found in the other list. In Alg. 1 lines 13–15 save the missing items for later processing in lines 27–30. Also, we need to detect cases when the maximization cannot be performed using the sorted components of lists (lines 32–34). In these cases, we compute sums for all variable value combinations to find the maximum. The example of applying this modified technique is shown in Fig. 3. Alg. 2 describes the steps of computing Eq. (2) in Max-Sum using G-FBP in Alg. 1.

4.3 Time Complexity and Selection of K

The main intuition behind G-FBP is the fact that the probability of finding the maximum value within the sorted section is very high with an appropriate K given the independence assumption of two lists. [12] uses enumerative combinatorics to construct the analysis on the probability of items with the same index not existing in M topmost items in the lists of size N. Under the assumption that the order statistics of two sorted lists are independent, this probability is computed as the probability of getting M red-colored balls where we randomly select M balls out of the box in which there are $(N - M)$ red-colored balls and M blue-colored balls.

Using the same notion, the probability of **not** finding the matching items within $K\sqrt{N}$ items in the lists of size N is

$$P(X > K\sqrt{N}; N) = \frac{(N - K\sqrt{N})!(N - K\sqrt{N})!}{(N - 2K\sqrt{N})!N!} \quad (6)$$

$$\leq \left(\frac{(N - K\sqrt{N})}{N}\right)^{K\sqrt{N}}, \quad (7)$$

where Eq. 7 can be derived by simply expanding Eq. 6 and using the relation $\frac{N-K\sqrt{N}-i}{N-i} < \frac{N-K\sqrt{N}}{N}$ to replace the intermediate terms. For the case of list size $N = 10000$, $K\sqrt{N} = 200$, the probability bound is as small as 0.0176. In other words, when there are two lists of length 10000 and 200 items are selected and sorted, the probability of finding matching items in 200 items on two lists is as large as 0.9824.

In Alg. 1, the algorithm iterates over all items if any set of items with the same index (that is, same variable configuration) is not found in the sorted part of the lists (see

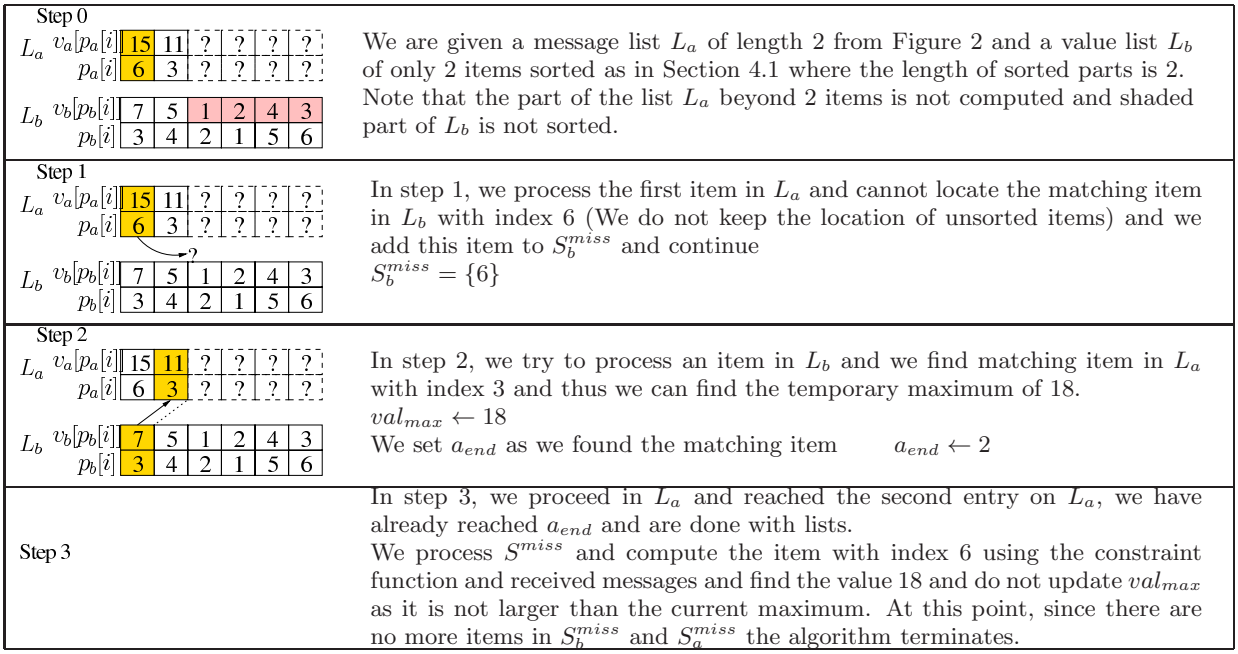


Figure 3: Example of G-FBP technique as in Algorithm 1

line 31–34). Therefore, finding such items is critical to the performance of the algorithm. In order to increase the probability of finding the matching items in sorted part of the lists, we increase the size of sorted parts. More specifically, with G-FBP that uses partially sorted lists, a specific condition is required to hold for the expected complexity for finding the maximum to be $O(\sqrt{N})$ given the lists of size N.

THEOREM 1. *The expected time complexity of* $O(\sqrt{N})$ *holds with partial lists when* $(1 - \frac{K}{\sqrt{N}})^{K\sqrt{N}} < \frac{1}{\sqrt{N}}$.

Proof: The expected running time is estimated based on the number of summed items evaluated in order to find the maximum. The expected number of summations E(Σ) is given as $\sum_{i=0}^{N-1} P(X > i; N)$. The probability $P(X > i; N)$ is the probability that the rank X of an item is not smaller than i. In our setting with partial lists, the probability of certain items to be in the unsorted part equals the probability of not finding the maximum within $K\sqrt{N}$. Thus,

$$P(X > i; N) = P(X > K\sqrt{N}) \text{ if } i > K\sqrt{N} \tag{8}$$

We re-write the expected number of summations as

$$E(\Sigma) = \sum_{i=0}^{K\sqrt{N}} P(X > i) + \sum_{i=K\sqrt{N}+1}^{N-1} P(X > K\sqrt{N}) \tag{9}$$

$$= \sum_{i=0}^{K\sqrt{N}} P(X > i) + \sum_{i=K\sqrt{N}+1}^{N-1} \frac{(N - K\sqrt{N})!(N - K\sqrt{N})!}{(N - 2K\sqrt{N})!N!} \tag{10}$$

$$\leq \sum_{i=0}^{N-1} P(X > i) + \sum_{i=K\sqrt{N}+1}^{N-1} \left(1 - \frac{K\sqrt{N}}{N}\right)^{K\sqrt{N}} \tag{11}$$

In Eq. 11,we already know from [2] that the first term is $O(\sqrt{N})$. The second summation equates to $(N - K\sqrt{N})/\sqrt{N}$ by the condition of the theorem and is dominated by $\sqrt{N}$. Therefore, the expected time complexity is $O(\sqrt{N})\square$

Note that the condition holds for $K = 2$ for list size 10000 and $K = 3$ for 1000000. Therefore, the condition holds for sufficiently small K for most cases. Also note that the analogous results hold for m-ary constraints since we use only two lists. The $\sqrt{N}$ is replaced by $N^{\frac{m-1}{2}}$.

5 Independence Assumption and Correlation Measure

As a consequence of having partial lists, the algorithm may compute all items on lists as in Alg. 1 line 31-34. The worst case complexity of using G-FBP is $O(mN^m)$ in this case. The Max-Sum with dynamic programming is $O(N^m)$(the standard Max-Sum $O(mN^m)$), so with G-FBP it may take more time than an efficient implementation of Max-Sum.

Also, the guarantee of the expected complexity of the FBP technique is constructed based on the assumption that the ranks of the items on the two lists are independent. However, the independence assumption of the FBP technique does not hold generally. The correlation of two lists are domain-dependent [12], and from our observations, it also varies for each constraint function and messages received per cycle.

If the two lists are negatively correlated, the expected complexity does not hold. It is likely that the G-FBP scheme fails to find the maximum item using partial lists, thereby increasing the time complexity of the algorithm. Consider the case that the two lists are completely negatively correlated such that $r_{x_i} = N - r_{y_i}$, where r is the rank of an item with index i on lists involving variables x and y respectively. The maximum value is not found until half of the lists are processed. Therefore, if we can detect negative correlation, then we should avoid applying the G-FBP approach.

5.1 Correlation Measure

We modify the Spearman's rank correlation measure [17] to measure the correlation among two partially sorted lists. It has two limitations for a direct use in our approach. Firstly, we only know the ranks of items in the sorted part. Second, it is more important to be on the same side with respect to $K\sqrt{N}$th items (the smallest sorted item) than to the median of the ranks in order to determine the likelihood of finding the maximum item in the sorted part. Thus, we assume that the items in the unsorted part have the same rank. We also consider the items are positively correlated when they are on the same side with respect to $K\sqrt{N}$th item.

Therefore, we modify the measure in the following way. The ranks of the items in the unsorted parts of lists are considered to be the same and equal to the mean of them, i.e. $\frac{K\sqrt{N}+1+N}{2}$. Also, we consider $K\sqrt{N}+\frac{1}{2}$ as the rank of the imaginary median and consider the length of the lists to be $2K\sqrt{N}$. However, there are $(N-K\sqrt{N})$ items in the unsorted part of the lists. Therefore, we weigh the values related to these items with the ratio of number of items on the two different parts, i.e. $\frac{N-K\sqrt{N}}{K\sqrt{N}}$.

Let x and y be two lists of length N where r_{x_i} and r_{y_i} are the ranks of the respective items with index i. Let $r_m = K\sqrt{N}+\frac{1}{2}$ be the imaginary median rank. Our redefined correlation measure is :

$$\rho' = \frac{\sum_i (k_{x_i})(k_{y_i})}{\sqrt{\sum_i k_{x_i}^2 \sum_i k_{y_i}^2}} \tag{12}$$

where the rank (for each list) is calculated as:

$$k_i = \begin{cases} \frac{(N-K\sqrt{N})}{K\sqrt{N}}(r_i - r_m), & \text{if } r_i < r_m \\ \left(\frac{(K\sqrt{N}+1+2K\sqrt{N})}{2} - r_m\right), & \text{if } r_i > r_m. \end{cases}$$

Definition 1. *Given the item x_i of list x, and rank of two positions r^1 and r^2 on list y such that $|r^1 - r_{x_i}| < |r_2 - r_{x_i}|$, the ranks of two lists x and y of equal length are positively correlated when $P(r_{y_i} = r^1) > P(r_{y_i} = r^2)$. They are negatively correlated when $P(r_{y_i} = r^1) < P(r_{y_i} = r^2)$. They are independent when $P(r_{y_i} = r^1) = P(r_{y_i} = r^2)$.*

That is, if the lists are positively correlated, the items with same index are likely to appear at nearby locations in two lists. Using the above definition, we can state the following result about our modified correlation measure:

Theorem 2. *For any sample set s of the items with ranks in the range $0 \le r \le \frac{3}{4}r_m$, the following holds. When the ranks of the two lists are independent, then the expected value of the correlation measure for a set s is $E(\rho'_s) = 0$. When the two lists are positively correlated, then $E(\rho'_s) \ge 0$ and when the lists are negatively correlated, then $E(\rho'_s) \le 0$.*

Due to lack of space, we only provides a sketch of a proof here. With the Def. 1, we can construct a relation of the probabilities of an item being at specific ranks. We use this relation to compute the sign of the expected value of k_i of an item in set s and also the sign of $E(k_{x_i}k_{y_i})$ in the numerator in Eq. 12. $E(\sum X) = \sum E(X)$, so we can determine the sign of the expected value of the correlation measure of set s.

5.2 GSC-FBP: Improving the Correlation of the Message and the Value Lists

We now develop a technique that takes advantage of the fact that messages change little near convergence. When we detect such a situation, we do the following steps. We merge the message list into the value list, creating a new list L_{sum}. This step requires $O(N^m)$ time for m-ary graphs. However, this merging operation is done only once and the results are reused for future iterations. We also create a message list L_{change} which denotes the changes in the incoming, new messages. As the algorithm proceeds, the list L_{change}'s rank distribution becomes uniform, which makes it independent of L_{sum}. This improves the efficiency of our approach, whose performance degrades with negative correlation. Once, the list L_{sum} and L_{change} are computed, the GSC-FBP approach finds the maximum using Alg. 1.

6 Experiments

We evaluated the effectiveness of our approach against the Max-Sum algorithm on two sets of problems with n-ary constraints. For fairness of comparison, we used an implementation of Max-Sum that uses dynamic programming with the worst case complexity of $O(N^m)$ for a single constraint instead of the standard Max-Sum with $O(mN^m)$ where the arity is m and the domain size is N. The two sets of DCOP instances that are used in our experiments are:

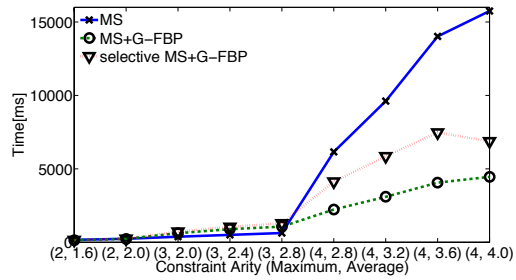
- 50 instances of random graphs with 25 variables with domain sizes from 10 to 30, and 15 constraints with the maximum arity of 2,3,4 or 5 with different average constraint arities.
- 25 instances of graphs in the radar coordination domain with 48 variables with domain size up to 15 and 96 constraints with the maximum arity 4.

We focus on the computational aspect of the algorithms because our approach does not affect the solution quality. Because the computational complexity of DCOPs is determined by the constraint arity among the parameters related to graph topology as well as by the domain size, we experiment on varying these two parameters .

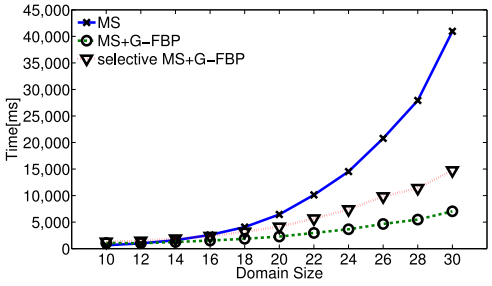
6.1 Random Graphs

Our initial tests perform a comparison over randomly generated DCOPs with n-ary constraints. We characterize each scenario by the maximum constraint arity (m_{max}), an average constraint arity (m_{avg}) and the variable domain size (N). We have explored scenarios with m_{max} from 2 to 5, m_{avg} from 1.6 to 4.4 with an increment of 0.4 and N from 10 to 30. From our knowledge, this problem set is one of the most computationally expensive problems for a DCOP. For the first problem set, we fixed the value K to 2 in regard to the length of the sorted parts of lists. This value is chosen based on the probability analysis from Section 4.3.

Fig. 4 shows the results for various arity settings and domain sizes. We observe that Max-Sum with G-FBP technique (MS+G-FBP) clearly outperforms Max-Sum (MS) for higher arities and larger domain sizes. Concretely, with G-FBP technique, the performance improved by 89% for an arity setting of (5,3.6) and the domain size of 10 and the performance improved by 82% for the domain size of 30 and arity setting of (3, 2.8). As the constraint arity and domain size increases, the number of entries that MS+G-FBP examines does not increase as much as the number of total entries. This increases the gain of MS+G-FBP. For lower arities and smaller domain size, MS performs better than MS+G-FBP because the overhead of sorting partial lists dominates the gain from finding the maximum value for shorter lists.



(a) Computation time as the constraint arity increases



(b) Computation time as the domain size increases

Figure 4: The computation time of Max-Sum(MS), Max-Sum with G-FBP(MS+G-FBP) and Max-Sum with G-FBP with correlation measure(selective MS+G-FBP) . For Fig.(a), the domain size of 10 was used. Datapoint (5, 3.6) is omitted to see the general trend. The performance of algorithms was (507178.5, 55070.6, 195056.9) for MS, MS+G-FBP and selective MS+G-FBP respectively. For Fig.(b) the arity setting of (3, 2.8) was used.

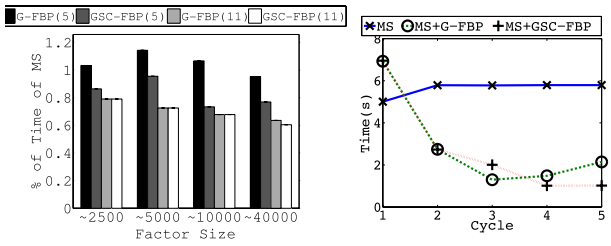
However, the use of the correlation measure in selective MS+G-FBP is not beneficial in this problem sets. Because the randomly generated constraint values leads the independence explained in Section 5 to hold, and thus there is no benefit in selectively applying G-FBP here and causes an additional overhead of computing the correlation measure as shown in Fig. 4.

6.2 Multiagent Radar Coordination Domain

Our next problem set is created from the abstracted radar coordination and scheduling application based on the real-time adaptive NetRad system [9]. Radars collect real-time data on the location and importance of phenomena and the system schedules the radars to focus their sensing on scheduled weather phenomena. This scheduling step can be thought of as a DCOP. See [7], for more details on the formulation.

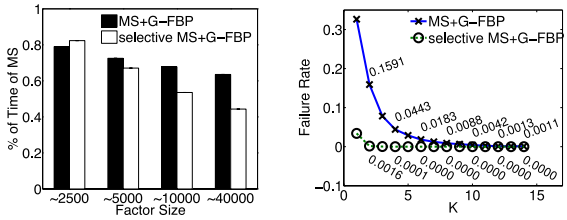
We developed a simulator in the Farm simulator framework [4]. Although it is a simulation environment, the utility functions are constructed based on the real scenario and the same utility function is used in the deployed system [9]. Our scenario involves 48 radars with a scenario of 96 phenomena with random locations, size, and type. The radars are placed in a grid with overlapping regions with other radars. This scenario creates problem instances with 48 variables, 96 constraints with the maximum arity of 4. In this data set, we do not directly control the constraint arity nor the domain size. These numbers vary in the experiments, so we categorized the computational difficulty of each problem instance by the maximum factor size. The maximum *factor size* is computed as the number of recorded entries in the constraint functions totaling $O(mN^m)$ for an m-ary constraint, where N is the maximal domain size of associated variables. We report the average runtime of 25 runs.

As shown in Fig. 5(a), both MS+G-FBP and MS+GSC-FBP outperform MS with an appropriate K as discussed in Sec. 4.3. However, there is not a significant difference between them when a reasonable K is chosen for at least this domain. As shown in Fig. 5(b), the time savings in later iterations of MS+G-FBP dominate the sorting overhead in the initial iteration, and leads to superior performance to MS. MS+G-FBP takes 36% less computation time than MS for the factor sizes in the range (10000, 40000] and K=11. The performance improvement by MS+G-FBP is not so significant as on the first dataset, because most constraints have lower arity and some variables have smaller domain size than the one related to the maximum factor size and also because of the data dependencies. Unlike randomly generated data sets, here variables have more structured dependencies through constraint functions and the independence assumption in Sec. 5 does not hold in this domain and MS+G-FBP performs poorly on instances of strongly negatively correlated lists. Therefore, we examine the use of correlation measure on this domain further.



(a) Time (b) Time Per Cycle

Figure 5: (a) The computation time ratio of MS+G-FBP and MS+GSC-FBP to MS. K value is in brackets. (b) Time taken at each cycle. $K = 11$.



(a) Time (b) Failing Probability

Figure 6: Performance improvement using correlation measure. $K = 11$.

On Experiments in Fig.6, we selectively applied G-FBP scheme (selective MS+G-FBP) when the correlation measure with the sample set of $\frac{K\sqrt{N}}{2}$ largest items in a value list computed as in Eq. 12 is positive. Selective MS+G-FBP takes 55% less computation time than MS in contrast to 36% for MS+G-FBP. As in Fig. 6(b), selective MS+G-FBP

almost always finds the maximum item in sorted parts of lists and the failure rate becomes zero when $K > 8$. Note that MS+G-FBP sometimes fails even with larger K values.

7 Conclusion

We presented a new approach, called generalized fast belief propagation (G-FBP), which optimizes the key computational bottleneck of the maximization operator in the popular Max-Sum algorithm. Our approach is applicable to a general setting in the context of arbitrary arity graphs as opposed to some previous approaches which operate only on pairwise graphs. We provide a significant reduction in the time complexity of computing a single message in the Max-Sum algorithm from $O(N^m)$ to $O(mN^{\frac{m+1}{2}})$ for general m-ary graphs. The key idea of our approach that distinguishes it from previous approaches is that only a small number of values are accessed from partially sorted lists to efficiently perform the maximization operation in Max-Sum, rather than performing the complete sorting. We also provide theoretical results regarding the number of samples required and a proof of expected complexity.

There are many interesting future research directions. Firstly, both G-FBP and GSC-FBP take a significant time penalty when they are unable to find the maximum item within the sorted lists; this potentially can be handled more gracefully if we semi-sort the items as we select top $K\sqrt{N}$ items. Such iterative processing will reduce the very high cost to compute the entire lists. Secondly, this technique can be applied in dynamic constraint optimization problems, when only a limited number of constraints change, to save the computation time by only partially sorting the lists. Our approach can therefore significantly increase the applicability of the Max-Sum algorithm to the multiagent domain by substantially reducing its computational overhead.

Acknowledgment

The authors would like to thank Dr. Akshat Kumar for valuable comments and discussions.

8 References

[1] S. Aji and R. McEliece. The generalized distributive law. *Information Theory, IEEE Transactions on*, 46(2):325–343, Mar 2000.

[2] T. S. Caetano and J. J. McAuley. Faster algorithms for max-product message-passing. *Journal of Machine Learning Research*, 12(4):1349–1388, 2011.

[3] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *AAMAS'08*, pages 639–646, 2008.

[4] B. Horling, R. Mailler, and V. Lesser. Farm: A Scalable Environment for Multi-Agent Development and Evaluation. In *Advances in Software Engineering for Multi-Agent Systems*, pages 220–237. 2004.

[5] M. Jain, M. Taylor, M. Tambe, and M. Yokoo. DCOPs meet the realworld: exploring unknown reward matrices with applications to mobile sensor networks. In *IJCAI'09*, pages 181–186, 2009.

[6] R. Junges and A. L. C. Bazzan. Evaluating the performance of DCOP algorithms in a real world, dynamic problem. In *AAMAS'08*, pages 599–606, 2008.

[7] Y. Kim, M. Krainin, and V. Lesser. Effective Variants of the Max-Sum Algorithm for Radar Coordination and Scheduling. In *IAT'07*, pages 357–364, October 2011.

[8] F. Kschischang, B. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *Information Theory, IEEE Transactions on*, 47(2):498 –519, feb 2001.

[9] J. F. Kurose, E. Lyons, D. McLaughlin, D. Pepyne, B. Philips, D. Westbrook, and M. Zink. An End-User-Responsive Sensor Network Architecture for Hazardous Weather Detection, Prediction and Response. In *AINTEC*, pages 1–15, 2006.

[10] R. T. Maheswaran, M. Tambe, E. Bowring, J. P. Pearce, and P. Varakantham. Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In *AAMAS'04*, pages 310–317. IEEE Computer Society, 2004.

[11] R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS'04*, pages 438–445. IEEE Computer Society, 2004.

[12] J. J. McAuley and T. S. Caetano. Exploiting data-independence for fast belief-propagation. In *ICML*, pages 767–774, 2010.

[13] P. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1-2):149–180, 2005.

[14] F. Pecora, P. J. Modi, and P. Scerri. Reasoning about and dynamically posting n-ary constraints in ADOPT. In *In Proceedings of 7th Int. Workshop on Distributed Constraint Reasoning (DCR-06), at AAMAS'06*, 2006.

[15] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *IJCAI'05*, pages 266–271, 2005.

[16] S. D. Ramchurn, A. Farinelli, K. S. Macarthur, and N. R. Jennings. Decentralized coordination in robocup rescue. *Comput. J.*, 53(9):1447–1461, 2010.

[17] C. Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 100(3/4):441–471, 1987.

[18] R. Stranders, A. Farinelli, A. Rogers, and N. R. Jennings. Decentralised coordination of mobile sensors using the max-sum algorithm. In *IJCAI'09*, pages 299–304, 2009.

[19] M. Vinyals, J. A. Rodriguez-Aguilar, and J. Cerquides. Constructing a unifying theory of dynamic programming DCOP algorithms via the generalized distributive law. *Autonomous Agents and Multi-Agent Systems*, 22(3):439–464, May 2011.

[20] Y. Weiss and W. T. Freeman. On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, 47(2):736–744, 2001.

[21] W. Zhang, G. Wang, Z. Xing, and L. Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artif. Intell.*, 161(1-2):55–87, Jan. 2005.