On Binary Max-Sum and Tractable HOPs

Marc Pujol-Gonzalez, Jesus Cerquides, Gonzalo Escalada-Imaz, Pedro Meseguer, Juan A. Rodriguez-Aguilar

Artificial Intelligence Research Institute, IIIA
Spanish National Research Council, CSIC
08193 Bellaterra, Spain
{mpujol,cerquide,gonzalo,pedro,jar}@iiia.csic.es

Abstract. The Max-Sum message-passing algorithm has been used to approximately solve several unconstrained optimization problems, specially in the distributed context. In general, the complexity of computing messages is exponential. However, if the problem is modeled using the so called *Tractable HOPs* (THOPs), binary MaxSum's messages can be computed in polynomial time. In this paper we review existing THOPs, and present new ones, aiming at providing an updated view of efficient message computation.

1 Introduction

The Max-Sum algorithm has been successfully employed to solve several multi-agent coordination problems modeled using the Distributed Constraint Optimization paradigm [8, 5, 7]. In most cases, these coordination problems are highly dynamic (e.g. a disaster response scenario). In those cases, exact solving is unfeasible because of the uncertainty of the agents' knowledge and the real-time pressure to act quickly. Therefore, in these scenarios we have to rely on approximate solving.

Technically, the Max-Sum algorithm [1] solves unconstrained optimization problems, either exactly or approximately depending on the problem's topology (optimally when it is a tree, approximately when it is a more complex graph). It comes from the Max-Product algorithm used for belief propagation in probabilistic graphical models, which performs products of probabilities. Since such products may result in quite small numbers, which may be affected by rounding procedures and produce errors in the long term, it is computationally preferred to work on the logarithmic space, were the product becomes a sum.

Max-Sum is a message-passing algorithm. The complexity of computing Max-Sum messages is exponential in the number of variables involved in the same factor, and this is a major issue for its practical applicability when this number increases. Then, it is of utmost importance to develop Max-Sum versions able to compute messages in polynomial time (preferred with low exponent). In the last years, linear or linearithmic complexities have been achieved for computing special types of factors called Tractable HOPs (THOPs) [4, 9, 7]. As a consequence,

Max-Sum is even more suited to handle multiagent coordination problems where a fast response is desirable when those can be modeled using THOPs.

This paper aims at providing a comprehensive view of these approaches, currently scattered in different publications. With this aim, we present the Max-Sum algorithm and describe those factors (or potentials) for which an efficient Max-Sum message computation has been developed. In addition, we describe a new potential types, not made explicit in the specialized literature.

The structure of the paper is as follows. We summarize the Max-Sum algorithm (n-ary and binary versions) and the formalization of affinity propagation [3] in terms of binary Max-Sum and THOPs in Section 2. We present existing THOPs in Section 3. New THOPs are enumerated in Section 4, providing the complexity of message computation. Finally, the paper concludes in Section 5.

2 Background

In this section we present the Max-Sum algorithm, along with its Binarized version with Tractable Higher Order Potentials and where the efficiency results come from.

2.1 Max-Sum

Let us consider a probabilistic graphical model defined by,

- a set $X = \{x_1, ..., x_n\}$ of random variables; each variable x_i may take a finite number of values; variable x_i with a value assigned is written $\mathbf{x}_i$; the sequence of variables with value is written $\mathbf{x}$;
- a set of potentials (or factors) $\{f_1, ..., f_r\}$; a potential f_j is a function from the possible values of the subset of variables involved in the potential f_j into the set of real numbers R.

As mentioned above, Max-Sum is an algorithm coming from the Max-Product algorithm. This is an inference algorithm used to solve the MAP (Maximum A Posteriori) problem (finding the assignment with highest global probability to the problem variables) in probabilistic graphical models. Typically, this amounts at computing $arg\ max_{\mathbf{x}} \prod_j f_j(\mathbf{x})$. Working in the logarithmic space to avoid underflow problems the problem becomes $arg\ max_{\mathbf{x}} \sum_j log(f_j(\mathbf{x}))$. Since the log function is monotonically increasing, the argument that maximizes the addition of logarithms is the same that the one that maximizes the addition without the log function, so finally Max-Sum computes $arg\ max_{\mathbf{x}} \sum_j f_j(\mathbf{x})$.

We can see the potentials f_j as utilities among the variables involved in each potential. Under this view, the problem becomes a *constraint optimization problem* (COP) and the assignment $arg\ max_{\mathbf{x}} \sum_j f_j(\mathbf{x})$ is the assignment that reaches the maximum global utility, so it is an optimal solution for this problem. Under this view the sum of potentials $\sum_j f_j(\mathbf{x})$ can be seen as the objective function to be optimized. Any existing constraint on this optimization problem

has to be included in the objective function as a new factor (so technically the problem is formulated as *unconstraned optimization*).

For example, let us consider a simple problem with two variables x_1 and x_2, where the global utility to be maximized is the addition of individual utilities $u(x_1) + u(x_2)$, under the constraint that x_1 and x_2 must take different values. This constraint can be expressed as the following factor,

$$diff(x_1, x_2) = \begin{cases} 0 & \text{if } x_1 \neq x_2 \\ -\infty & \text{otherwise} \end{cases}$$

which enters in the objective function, that is,

$$u(x_1) + u(x_2) + diff(x_1, x_2)$$

Now this objective function is unconstrained, and Max-Sum can work on it, looking for the optimum. Obviously, the maximum of this function occurs when the last term is evaluated to 0 (that is, when $x_1 \neq x_2$ so the original constraint is satisfied).

Max-Sum operates over the factor graph of the problem, defined as follows:

- There are two types of nodes, variable nodes and factor nodes (graphically represented as circles and squares, respectively).
- Each problem variable is represented by a variable node, each potential is represented by a factor node.
- Each variable node is connected to each factor node in whose potential it is involved (the factor graph is bipartite: variable-factors).

Max-Sum is a message-passing algorithm on the factor graph. In the distributed context, Max-Sum can be seen as an iterative algorithm. An iteration ends when each node has sent one message to each neighbor. At iteration p, a variable node x computes a message for each neighbor factor. For factor f, this message is computed adding all messages that x received at the previous iteration $p-1$, except the message that arrived from f,

$$\mu_{x \to f} = \sum_{g \in N(x) | g \neq f} \mu_{g \to x}$$

where $N(x)$ is the set of neighbors of variable node x in the factor graph. Similarly, at iteration p a factor f computes a message for each neighbor variable. The message for variable x is computed in two steps:

1. Adding the potential (or factor) f plus all messages that f received at the previous iteration $p-1$, except the message that arrived from x.
2. Taking the maximum among all variables that are not x.

$$\mu_{f \to x} = max_{y \in N(f) | y \neq x}(f + \sum_{y \in N(f) | y \neq x} \mu_{y \to f})$$

It is well known that Max-Sum is exact if the factor graph has a tree structure. Otherwise, Max-Sum is an approximate algorithm, for which there is no guarantee of convergence. Nevertheless, after a number of iterations, a decoding process[1] is performed at the variable nodes, obtaining a value to be assigned to each variable. These assignments have been found useful for a number of applications.

In the case of distributed optimization (when problem parts are distributed among agents, DCOPs), it is relatively direct to implement Max-Sum into a distributed context (because of that, Max-Sum has been widely used for distributed applications [8]).

2.2 Binary Max-Sum

From standard Max-Sum we can develop binary Max-Sum as follows. A standard variable with d possible values becomes d boolean variables, all connected by a new factor that assures that only one of them will be active. [2] A factor involving k standard variables has (in the worst case) d^k entries. With boolean variables its factor increases up to $2^{d \times k}$ entries. [3] Therefore, the binary version of Max-Sum implies a multiplicative increment in the number of variables, a linear increment in new factors and an exponential increment in the size of existing factors. Then, moving into the binary version implies a significant increment in the size of the problem representation. However, working on the binary version could be beneficial if this implies substantial simplifications in the message computations, as we will see in the next sections.

Having factors involving a low number of variables is obviously beneficial, because the exponential blow-up is limited. However, in some cases the number of variables in a factor is high. Then, the use of *tractable high order potentials* (THOPs) it is of special interest. THOPs are factors involving several variables for which Max-Sum messages can be computed in polynomial time (in fact, between linear and quadratic). Observe that in the general case, computing the message from a factor to a variable implies an exponential number of steps.

A common simplification in the binary case is as follows: instead of sending two values in each message (the value for 1 and 0, the two possible states of a boolean variable), you send a single scalar that is the difference between them, that is,

$$\nu_{p \to q} = \mu_{p \to q}(1) - \mu_{p \to q}(0)$$

where p/q are either variable/factor or factor/variable. This is equivalent to consider that, upon reception, the value of $\mu_{p \to q}(0)$ is always 0, and the sent scalar is the value corresponding to $\mu_{p \to q}(1)$. Additionally, sending the difference performs a normalization role that prevents message values from growing indefinitely when the factor graphs contains loops.

---

[1] A popular decoding procedure consists of, at each variable node, adding the last received messages from factors and taking the value with highest probability.

[2] This is the *OneAndOnlyOne* factor defined in Section 3.1.

[3] Observe that $2^{d \times k} = 2^{d^k}$. Since $d << 2^d$, we conclude that $d^k << 2^{d \times k}$.

2.3 Affinity Propagation

Affinity Propagation [3] is an algorithm to find exemplars or representatives from a set of instances (or examples) such that the sum of squared errors is minimized between exemplars and instances. This problem is also known as *exemplar-based clustering*. It has been shown that Affinity Propagation can be seen as an application of the binary Max-Sum algorithm on a specific factor graph that represent the objective function to be optimized [4]. In the following, we present the main points of this view.

Given a number of instances $\{ins_1, ins_2, ..., ins_N\}$, the goal is to find a number of exemplars that act as representatives of the instances, such that the sum of squared errors is minimized. This problem has been attacked by k-medians clustering [2], which provided an approximated solution with some distance with the optimal solution computed by linear programming (with a substantial computation time).

In binary Max-Sum, we consider the binary variables c_{ij} with the meaning,

$$c_{ij} = \begin{cases} 1 & \text{if instance } i \text{ has as exemplar instance } j \\ 0 & \text{otherwise} \end{cases}$$

There is a similarity measure between instances i and j given by $s(i,j)$ (as default, you can take minus the sum of squared distances between i and j). For $s(k,k)$ you can take the preference for instance k to be exemplar.

The objective function is,

$$\sum_{ij}^{N} S_{ij}(c_{ij}) + \sum_{i=1}^{N} I_i(c_{i1}, ..., c_{iN}) + \sum_{j=1}^{N} E_j(c_{1j}, ..., c_{Nj})$$

where $S_{ij}(c_{ij}) = s(i,j)c_{ij}$ (so the first term is the sum of similarities between instances and chosen exemplars). Functions I_i and E_j are respectively,

$$I_i(c_{i1}..., c_{iN}) = \begin{cases} -\infty & \text{if } \sum_j c_{ij} \neq 1 \\ 0 & \text{otherwise} \end{cases}$$

$$E_j(c_{i1}..., c_{iN}) = \begin{cases} -\infty & \text{if } c_{jj} = 0 \text{ and there is some } i \neq j \text{ s.t. } c_{ij} = 1 \\ 0 & \text{otherwise} \end{cases}$$

The second term indicates that every instance must be assigned to one and only one exemplar, [4] while the third term establishes that if j is exemplar for instance i, it must also be for itself. These two terms are constraints that cost $-\infty$ if they are violated (that is, they are *hard* constraints). The objective function is maximized when these two terms are evaluated to 0 (that is, when the constraints they represent are satisfied).

The factor graph for this formalization is depicted in Figure 1 (using the standard approach of representing variables as circles and factors as squares). Messages exchanged between variables and factors are also depicted. After some simple calculations (see [4] for details), the expressions for each message are,

[4] In fact, I_i is the *OneAndOnlyOne* potential, as defined in Section 3.1.

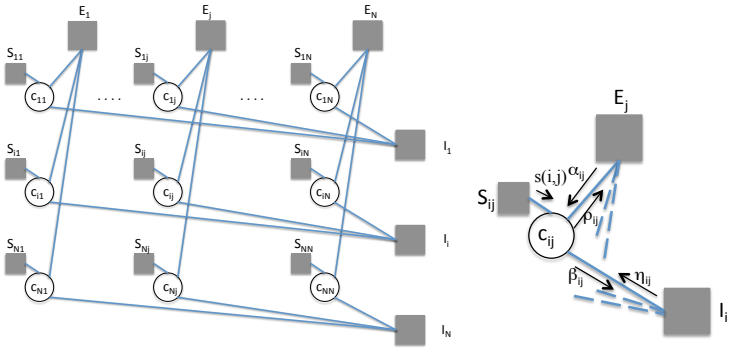

Fig. 1. Left: Factor graph of affinity propagation. Right: Messages exchanged

$$\beta_{ij} = s(i,j) + \alpha_{ij} \qquad \rho_{ij} = s(i,j) + \eta_{ij} \qquad \eta_{ij} = -max_{k \neq j} \beta_{ik}$$

$$\alpha_{ij} = \begin{cases} \sum_{k \neq j} max(\rho_{kj}, 0) & i = j \\ min\left(0, \rho_{jj} + \sum_{k \notin \{i,j\}} max(\rho_{kj}, 0)\right) & i \neq j \end{cases}$$

Computationally it is extremely easy to compute these messages: β_{ij} and ρ_{ij} are computed in constant time, while η_{ij} and α_{ij} require linear time.

To the best of our knowledge, [4] is the first paper explaining clearly the benefits of efficient message computation in binary Max-Sum. Although potentials are neither treated in isolation nor analyzed abstractly, this work represents a significant step forward in understanding the role of THOPs and how the simplest cases can be analyzed.

3 Tractable HOPs

The idea of THOPs, linked to Binary Max-Sum, has been presented in several papers before. In particular, we mention the work of Tarlow et al. [9], which clearly describe the proposed approach. In the following, we describe the different potentials that are known to be adequate for efficient binary Max-Sum operation.

3.1 OneAndOnlyOne

This potential requires that only one binary variable is active, out of a set of variables,

$$OneAndOnlyOne(\mathbf{x}) = \begin{cases} 0 & \text{if } \sum_{k=1}^{N} x_k = 1 \\ -\infty & \text{otherwise} \end{cases}$$

OneAndOnlyOne has been mentioned in [9]. It is easy to see that Max-Sum messages with this potential can be computed in linear time, as shown in [7], where this potential is called *selection factor*.

3.2 Cardinality Potentials

Cardinality Potentials are factors whose value depends solely on the number of variables that are active, disregarding which specific ones. Formally, given a set of (binary) variables X,

$$CardinalityPotential(X) = f\left(\sum_{x_i \in X} \mathbf{x}_i\right),$$

where f can be any function that takes a non-negative integer and returns a real-valued utility. Tarlow *et al.* [9] proved that max-sum's messages from this factors can be computed efficiently in $O(NlogN)$ time.

These potentials are useful to model a wide range of situations. For instance, in a task allocation context, a task could require at least k agents to be serviced. This requirement can be modeled as a *CardinalityPotential*, where $x_i \in X$ is the variable representing "allocate agent i to the task", and

$$f(y) = \begin{cases} 0 & \text{if } y \geq k \\ -\infty & \text{otherwise} \end{cases}.$$

Although, in the general case, computing the messages of cardinality potentials requires $O(NlogN)$ time, there are some particular cases that can be computed more efficiently. That is, for certain functions f, the messages can be computed in linear time instead. An example is the *OneAndOnlyOne* potential above, and we will present a few additional cases later on.

3.3 Weighting Potentials

Weighting potentials are potentials that specify an independent weight (incentive) for each of its constituent variables. That is, given a set of variables $X = \{x_1, \ldots, x_k\}$ and a set of real-valued weights $W = \{w_1, \ldots, w_k\}$, a *Weighting* potential is defined as

$$Weighting(X, W) = \sum_{i=0}^{k} \mathbf{x}_i w_i.$$

This potential is not very interesting in itself, because we could decompose it in k unary potentials and obtain the same results. However, Pujol-Gonzalez *et al.* [7] showed that a *Weighting* potential can be combined with any other THOP T, and the resulting potential's messages can be computed in $O(k + O(T))$ time.

Weighting potentials are very useful to express preferences between different possible choices. For instance, in a task allocation setting, *Weighting* potentials can be used to represent the fitness of an agent to perform different tasks.

3.4 Convex Set Potentials

Convex set potentials are ordered potentials that enforce all active variables to form a convex set. In other words, these factors define an order over the variables they contain. Thereafter, they enforce that only contiguous variables can be active. That is, given a set of variables X:

$$ConvexSet(X) = \begin{cases} 0 & \text{if } \mathbf{x}_{i-1} = 0 \vee \sum_{j<i} \mathbf{x}_j = 0 \quad \forall x_i \in X | \mathbf{x}_i = 1 \\ -\infty & \text{otherwise} \end{cases}$$

Based on the notion of maximum weight contiguous sequences, messages from this factor can be computed in linear $O(N)$ time as explained in [9].

These factors are useful to model temporal or spatial constraints. For instance, in a meeting scheduling problem, a *ConvexSet* potential could be used to enforce that meetings should be attended for an arbitrarily long contiguous period of time.

3.5 Before-After Potentials

Before-After potentials incentivize one (and only one) variable in an ordered set to be activated before one (and only one) variable in another set gets activated. Given two sets of variables X and Y, this amounts to

$$BeforeAfter(X,Y) = \begin{cases} -\infty & \text{if } \left(\sum_{\mathbf{x}_i \in X}\right) \neq 1 \vee \left(\sum_{\mathbf{y}_i \in Y}\right) \neq 1 \\ 0 & \text{if } i > j \quad \forall x_i \in X, y_j \in Y | \mathbf{x}_i = \mathbf{y}_i = 1 \\ -\alpha & \text{otherwise} \end{cases}$$

Messages of this factors can ben computed in $O(N)$ time using a similar procedure to the one used to compute *Cardinality Potentials*' one. The complexity is lower though because incoming messages do not need to be sorted in this case.

BeforeAfter factors are also used to model temporal or spatial constraints. However, in this case they are used to enforce orderings between events. For instance, you could incentivize (or even enforce by setting $\alpha = \infty$) a certain meeting to take place before another one.

3.6 Composite Potentials

Composite potentials represent the conditioned composition of multiple THOPs. These are useful to represent the fact that, given the values of a set of conditioning variables C, the other variables in the factor (X) form another THOP. Formally,

$$CompositePotential(C, X) = THOP_{\mathbf{C}}(X),$$

where $THOP_{\mathbf{C}}$ is a possibly different THOP for each combination of values $\mathbf{C}$.

Notice that any HOP (tractable or not) defined over a set of variables T can be represented as a *CompositePotential* where $C = T$ and $Y = \emptyset$. Therefore, the

cost of computing these potentials is exponential on the number of variables in C. However, the interesting part of composite potentials is that their complexity is polynomial in X (because each $THOP_{\mathbf{C}}$ is polynomial). Globally, messages of composite potentials can be computed in $O(2^{|\mathbf{C}|} \times \sum_{\mathbf{C}} O(THOP_{\mathbf{C}}))$ time.

These potentials are useful when a large number of variables depend on the value of a few ones. For instance, the column potentials E_j in *Affinity Propagation* (see Section 2.3) can be understood as a composite potential with a single conditioning variable c_{jj} and a set of conditioned variables $X = \{c_{ij} | i \neq j\}$.

3.7 Equality Potential

Initially defined in [6] on a pair of variables, x_i and x_j, this potential requires that both variables take the same value,

$$Equality(x_i, x_j) = \begin{cases} 0 & \text{if } x_i = x_j \\ -\infty & \text{otherwise} \end{cases}$$

Messages leaving this binary factor can be computed in constant time. This potential can be easily generalized to involve any number of variables. In this case, messages for this potential can be computed in linear time.

4 New Tractable HOPs

Now that we have reviewed different THOPs that have been presented in the literature, we move on by introducing a few new useful HOPs and showing how their messages can be computed efficiently.

4.1 AllOne and AllZero

In some cases, a potential requires that all variables are active. This potential is *AllOne*, defined as,

$$AllOne(\mathbf{x}) = \begin{cases} 0 & \text{if } \sum_{k=1}^{N} x_k = N \\ -\infty & \text{otherwise} \end{cases}$$

The message from this potential f to a generic variable x_i is as follows. Considering $x_i = 0$, it is direct to see that all terms are $-\infty$ because x_i takes value 0. Therefore,

$$\mu_{f \to x_i}(0) = -\infty$$

Considering $x_i = 1$, the only term that is greater than $-\infty$ is the one with all x's taking value 1, for which f takes value 0. Then,

$$\mu_{f \to x_i}(1) = \sum_{j=1, j \neq i}^{N} \mu_{x_j \to f}(x_j = 1)$$

and the message to be sent to x_i is,

$$\mu_{f\to x_i}(1) - \mu_{f\to x_i}(0) = \sum_{j=1, j\neq i}^{N} \mu_{x_j\to f}(x_j = 1) - (-\infty) = \infty$$

A similar situation happens for the dual potential requiring that all variables are inactive, *AllZero*,

$$AllZero(\mathbf{x}) = \begin{cases} 0 & \text{if } \sum_{k=1}^{N} x_k = 0 \\ -\infty & \text{otherwise} \end{cases}$$

The message from this potential f to a generic variable x_i is as follows. For $x_i = 0$, the only term higher than $-\infty$ is the one with all variables taking value 0. Then,

$$\mu_{f\to x_i}(0) = \sum_{j=1, j\neq i}^{N} \mu_{x_j\to f}(x_j = 0) = 0$$

but for $x_i = 1$ all terms are $-\infty$. Therefore, the message to be sent to x_i is,

$$\mu_{f\to x_i}(1) - \mu_{f\to x_i}(0) = -\infty - 0 = -\infty$$

In both cases, messages from these potentials can be computed in constant time.

Taking in isolation, these two potentials are clearly redundant because variables can be replaced by their corresponding values (all 1's, or all 0's). However, these potentials can be useful as forming part of other, more complex, Composite Potentials.

4.2 AllEqual

This potential requires that all variables are equal, either 1 or 0,

$$AllEqual(\mathbf{x}) = \begin{cases} 0 & \text{if } x_j = x_{j+1} \quad 1 \leq j < N \\ -\infty & \text{otherwise} \end{cases}$$

The message from this potential f to a generic variable x_i is as follows. Considering $x_i = 0$, it is direct to see that the only term greater than $-\infty$ is the one when all variables take value 0. Therefore,

$$\mu_{f\to x_i}(0) = \sum_{j=1, j\neq i}^{N} \mu_{x_j\to f}(x_j = 0)$$

Considering $x_i = 1$, the same happens with the term when all variables take value 1. Then,

$$\mu_{f\to x_i}(1) = \sum_{j=1, j\neq i}^{N} \mu_{x_j\to f}(x_j = 1)$$

Then, the message to be sent to variable x_i is,

$$\mu_{f\to x_i}(1) - \mu_{f\to x_i}(0) = \sum_{j=1,j\neq i}^{N} \mu_{x_j\to f}(x_j=1) - \sum_{j=1,j\neq i}^{N} \mu_{x_j\to f}(x_j=0) = \sum_{j=1,j\neq i}^{N} \nu_{x_j\to f}$$

which happens to be the sum of messages received by factor f from all variables but x_i. Hence, factor f can compute all its outgoing messages in linear time. That is, it first computes the full sum, and then it sends the message to each neighbor by subtracting that neighbor's message from the full sum.

4.3 AtMostOne

This potential requires that at most one binary variable is active, out of a set of variables,

$$AtMostOne(\mathbf{x}) = \begin{cases} 0 & \text{if } \sum_{k=1}^{N} x_k \leq 1 \\ -\infty & \text{otherwise} \end{cases}$$

The message from this potential f to a generic variable x_i is as follows (first we consider value 0, then value 1):

$$\mu_{f\to x_i}(0) = max_{X|x_i=0}[f(x_1,...,x_{i-1},x_i=0,x_{i+1},...,x_N) + \textstyle\sum_{j=1,j\neq i}^{N} \mu_{x_j\to f}(x_j)]$$

Since f with more than one active variable is $-\infty$, these terms can be removed from computing max, keeping the terms with at most one active variable,

$$\begin{aligned} \mu_{f\to x_i}(0) = max[\ & f(0,...,x_i=0,...,0) + \textstyle\sum_{j=1,j\neq i}^{N} \mu_{x_j\to f}(x_j=0), \\ & f(1,...,x_i=0,...,0) + \mu_{x_1\to f}(x_1=1) + \textstyle\sum_{j=2,j\neq i}^{N} \mu_{x_j\to f}(x_j=0), \\ & ... \\ & f(0,...,x_i=0,...,1) + \mu_{x_N\to f}(x_N=1) + \textstyle\sum_{j=1,j\neq i}^{N-1} \mu_{x_j\to f}(x_j=0)] \end{aligned}$$

We can now write this expression in terms of $\nu_j = \mu_{x_j\to f}(1) - \mu_{x_j\to f}(0)$ (abusing the notation a bit, we write ν_j instead of $\nu_{j\to f}$). Remember that sending the difference between $\mu_{x_j\to f}(1)$ and $\mu_{x_j\to f}(0)$ we assume at reception that $\mu_{x_j\to f}(x_j=0)=0$, we obtain

$$\mu_{f\to x_i}(0) = max[\ 0+0, 0+\nu_1+0, ..., 0+\nu_N+0] = max[0,\nu_1,...,\nu_{i-1},\nu_{i+1},...,\nu_N]$$

Analogously, for value 1,

$$\mu_{f\to x_i}(1) = max_{X|x_i=1}[f(x_1,...,x_{i-1},x_i=1,x_{i+1},...,x_N) + \textstyle\sum_{j=1,j\neq i}^{N} \mu_{x_j\to f}(x_j)]$$

but this expression has a single term that is different from $-\infty$, the term where all variables but x_i take value 0. Then,

$$\mu_{f\to x_i}(1) = f(0,...,0,x_i=1,0,...,0) + 0 = 0$$

so the message to be sent to x_i is,

$$\begin{aligned} \nu_{f\to x_i} = \mu_{f\to x_i}(1) - \mu_{f\to x_i}(0) &= 0 - max[0,\nu_1,...,\nu_{i-1},\nu_{i+1},...,\nu_N] \\ &= -max[0,\nu_1,...,\nu_{i-1},\nu_{i+1},...,\nu_N] \end{aligned}$$

which can be easily computed in linear time.

Moreover, notice that we only need to know the maximum ν_j value ($j \neq i$) to compute $\nu_{f \to x_i}$. Hence, we can compute all outgoing messages in $O(N)$ time by precomputing the two largest incoming messages ν^* and ν^{**}. Thereafter, the single-valued message sent to each neighbor is simply

$$\nu_{f \to x_i} = \begin{cases} max(0, \nu^*) & \text{if } \nu_i \neq \nu^* \\ max(0, \nu^{**}) & \text{otherwise} \end{cases} .$$

4.4 AtLeastOne

This potential requires that at least one binary variable is active, out of a set of variables,

$$AtLeastOne(\mathbf{x}) = \begin{cases} 0 & \text{if } \sum_{k=1}^{N} x_k \geq 1 \\ -\infty & \text{otherwise} \end{cases}$$

The message from this potential f to a generic variable x_i is as follows (first we consider value 0, then value 1):

$$\mu_{f \to x_i}(0) = max_{X|x_i=0}[f(x_1, ..., x_{i-1}, x_i = 0, x_{i+1}, ..., x_N) + \sum_{j=1, j \neq i}^{N} \mu_{x_j \to f}(x_j)]$$

Since f when all variables are 0 is $-\infty$, this term can be removed from computing max, keeping the terms with at least one active variable,

$$\begin{aligned} \mu_{f \to x_i}(0) = max[\ & f(1, ..., x_i = 0, ..., 0) + \mu_{x_1 \to f}(x_1 = 1) + \sum_{j=2, j \neq i}^{N} \mu_{x_j \to f}(x_j = 0), \\ & f(0, ..., x_i = 0, ..., 1) + \mu_{x_N \to f}(x_N = 1) + \sum_{j=1, j \neq i}^{N-1} \mu_{x_j \to f}(x_j = 0), \\ & ... \\ & f(1, ..., x_i = 0, ..., 1) + \sum_{j=1, j \neq i}^{N} \mu_{x_j \to f}(x_j = 1)] \end{aligned}$$

Writing this expression in terms of $\nu_j = \mu_{x_j \to f}(1) - \mu_{x_j \to f}(0)$, and remembering that sending the difference between $\mu_{x_j \to f}(1)$ and $\mu_{x_j \to f}(0)$ we assume that upon reception $\mu_{x_j \to f}(x_j = 0) = 0$, we obtain

$$\begin{aligned} \mu_{f \to x_i}(0) &= max[0 + \nu_1 + 0, ..., 0 + \nu_N + 0, ..., 0 + \sum_{k=1, k \neq i}^{N} \nu_k] \\ &= max[\nu_1, ..., \nu_N, \nu_1 + \nu_2, ..., \nu_{N-1} + \nu_N, ..., \sum_{k=1, k \neq i}^{N} \nu_k] \end{aligned}$$

Now we consider two cases: when some $\nu_j > 0$ and when all $\nu_j \leq 0$. When some $\nu_j > 0$ the maximum is the term that is the addition of all positive ν. When all $\nu_j \leq 0$, the maximum is $max[\nu_1, ..., \nu_{i-1}, \nu_{i+1}, ..., \nu_N]$. Then,

$$\mu_{f \to x_i}(0) = \begin{cases} \sum_{\nu_j > 0} \nu_j & \text{if some } \nu_j > 0 \\ max[\nu_1, ..., \nu_{i-1}, \nu_{i+1}, ..., \nu_N] & \text{otherwise} \end{cases}$$

Analogously, for value 1,

$$\mu_{f \to x_i}(1) = max_{X|x_i=1}[f(x_1, ..., x_{i-1}, x_i = 1, x_{i+1}, ..., x_N) + \sum_{j=1, j \neq i}^{N} \mu_{x_j \to f}(x_j)]$$

All terms of this expression evaluate to finite values. Thus,

$$\begin{aligned}\mu_{f\to x_i}(1) = max[\ & f(0,...,x_i=1,...,0) + \textstyle\sum_{j=1,j\neq i}^{N} \mu_{x_j\to f}(x_j=0),\\ & f(1,...,x_i=1,...,0) + \mu_{x_1\to f}(x_1=1) + \textstyle\sum_{j=2,j\neq i}^{N} \mu_{x_j\to f}(x_j=0),\\ & ...\\ & f(0,...,x_i=1,...,1) + \mu_{x_N\to f}(x_N=1) + \textstyle\sum_{j=1,j\neq i}^{N-1} \mu_{x_j\to f}(x_j=0),\\ & ...\\ & f(1,...,x_i=1,...,1) + \textstyle\sum_{j=1}^{N} \mu_{x_j\to f}(x_j=1)]\end{aligned}$$

Writing this expression in terms of ν_j, we obtain

$$\begin{aligned}\mu_{f\to x_i}(1) &= max[0+0, \nu_1+0, ..., 0+\nu_N+0, ..., 0+\textstyle\sum_{k=1,k\neq i}^{N}\nu_k]\\ &= max[0, \nu_1, ..., \nu_N, \nu_1+\nu_2, ..., \nu_{N-1}+\nu_N, ..., \textstyle\sum_{k=1,k\neq i}^{N}\nu_k]\end{aligned}$$

Again we consider two cases: when some $\nu_j > 0$ and when all $\nu_j \leq 0$. When some $\nu_j > 0$ the maximum is the term that is the addition of all positive ν. When all $\nu_j \leq 0$, the maximum is 0. Then,

$$\mu_{f\to x_i}(1) = \begin{cases}\sum_{\nu_j>0}\nu_j & \text{if some } \nu_j > 0\\ 0 & \text{otherwise}\end{cases}$$

Therefore, the message $\nu_{f\to x_i}$ is,

$$\mu_{f\to x_i}(1) - \mu_{f\to x_i}(0) = \begin{cases}0 & \text{if some } \nu_j > 0\\ -max[\nu_1, ..., \nu_{i-1}, \nu_{i+1}, ..., \nu_N] & \text{otherwise}\end{cases}$$

which can be easily computed in linear time.

Furthermore, we can compute all messages in a single linear pass by doing the same preprocessing we did for the *AtLeastOne* potential. Basically, given the two largest incoming values ν^* and ν^{**}, this potential's outgoing messages can be assessed as

$$\nu_{f\to x_i} = \begin{cases}0 & \text{if } \nu_i \neq \nu^* \text{ and } \nu^* > 0\\ 0 & \text{if } \nu_i = \nu^* \text{ and } \nu^{**} > 0\\ -\nu^* & \text{if } \nu_i \neq \nu^* \text{ and } \nu^* \leq 0\\ -\nu^{**} & \text{otherwise}\end{cases}.$$

5 Conclusions

In this paper we presented the max-sum message passing algorithm, but focusing on its usage as an approximate Constraint Optimization Problem solver instead of as a graphical model solver. Although max-sum has been extensively used to solve COPs, the graphical modeling communities have recently developed results that are not widely known within the COP community. Hence, we examined these results, that show how the algorithm's complexity can be reduced from

exponential to (low-order) polynomial when solving specific types of factors. Thereafter, we reviewed a number of such factors that have been presented elsewhere, and provide examples of how they could be used to model typical situations that arise in COP problems. Finally, we presented a few additional factors, showing how they can also be processed efficiently. To summarize, this work brings the COP community closer to these very promising efficiency results, and provides a few additional tools to extend the range of problems efficiently solvable using max-sum.

References

1. C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.
2. M. Charikar, S. Guha, E. Tardos, and D. Shmoys. A constant-factor approximation algorithm or the k-median problem. *Journal of Computers and System Sciences*, 65:129–149, 2002.
3. B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315:972–976, 2007.
4. I. E. Givoni and B. J. Frey. A binary variable model for affinity propagation. *Neural Computation*, 21(6):1589–1600, 2009.
5. Y. Kim, M. Krainin, and V. Lesser. Application of Max-Sum algorithm to radar coordination and scheduling. In *Workshop on Distributed Constraint Reasoning*, 2010.
6. T. Penya-Alba, M. Vinyals, J. Cerquides, and J. A. Rodriguez-Aguilar. A scalable message-passing algorithm for supply chain formation. In *Proc. AAAI-12*, pages 1436–1442, 2012.
7. M. Pujol-Gonzalez, J. Cerquides, P. Meseguer, J. A. Rodriguez-Aguilar, and M. Tambe. Engineering the decentralized coordination of UAVs with limited communication range. In *Proc. CAEPIA-13, LNAI 8109,*, pages 199–208, 2013.
8. A. Rogers, A. Farinelli, R. Stranders, and N. R. Jennings. Bounded approximate decentralized coordination via the Max-Sum algorithm. *Artificial Intelligence*, 175:730–759, 2011.
9. D. Tarlow, I.E. Givoni, and R. S. Zemel. HOP-MAP: Efficient message passing with high order potentials. In *Proc. AISTATS-2010*, pages 812–819, 2010.