Balancing Asymmetry in Max-sum using Split Constraint Factor Graphs

Liel Cohen and Roie Zivan

Ben Gurion University of the Negev,
Beer Sheva, Israel
lielc@post.bgu.ac.il,zivanr@bgu.ac.il

Abstract. Max-sum is a version of Belief Propagation, used for solving DCOPs. On tree-structured problems, Max-sum converges to the optimal solution in linear time. When the constraint graph representing the problem includes multiple cycles, Max-sum might not converge and explore low quality solutions. Damping is a method that increases the chances that Max-sum will converge. Damped Max-sum (DMS) was recently found to produce high quality solutions for DCOP when combined with an anytime framework.
We propose a novel method for adjusting the level of asymmetry in the factor graph, in order to achieve a balance between exploitation and exploration, when using Max-sum for solving DCOPs. By converting a standard factor graph to an equivalent split constraint factor graph (SCFG), in which each function-node is split to two function-nodes, we can control the level of asymmetry for each constraint. Our empirical results demonstrate that by applying DMS to SCFGs with a minor level of asymmetry we can find high quality solutions in a small number of iterations, even without using an anytime framework. As part of our investigation of this success, we prove that for a factor-graph with a single constraint, if this constraint is split symmetrically, Max-sum applied to the resulting cycle is guaranteed to converge to the optimal solution and demonstrate that for an asymmetric split, convergence is not guaranteed.

1 Introduction

The Max-sum algorithm [5] is an incomplete inference (GDL-based) algorithm for solving Distributed Constraint Optimization Problems (DCOP), a general model for distributed problem solving that has a wide range of applications in multi-agent systems. Max-sum has drawn considerable attention in recent years, including being proposed for multi-agent applications such as sensor systems [24, 21] and task allocation for rescue teams in disaster areas [17]. Max-sum is actually a version of the well known Belief propagation algorithm [26], used for solving DCOPs. Agents in Max-sum propagate cost/utility information to all neighbors. This contrasts with other inference algorithms such as ADPOP [14], which only propagate costs up a pseudo-tree structure overlaid on the agents. As is typical of inference algorithms, Max-sum is purely exploitive both in the computation of its beliefs and in its selection of values based on those beliefs.

Belief propagation in general (and Max-sum specifically) is known to converge to the optimal solution for problems whose constraint graph is acyclic. On problems with

cycles, the agents' beliefs may fail to converge, and the resulting assignments that are considered optimal under those beliefs may be of low quality [26, 5, 31]. This occurs because cyclic information propagation leads to inaccurate and inconsistent information being computed by the agents. Unfortunately, many DCOPs that were investigated in previous studies are dense and indeed include multiple cycles (e.g., [11, 7]). However, in contrast to most DCOP algorithms, Max-sum was found to produce solutions with similar quality when applied to symmetric and asymmetric problems [32].

Damping is a method that decreases the effect of cyclic information propagation in Belief propagation by balancing the weight of the new calculation performed in each iteration and the weight of calculations performed in previous iterations. As a result it increases the chances for convergence [9, 20, 22, 15]. A recent investigation of the effect of damping on Max-sum when applied to DCOPs revealed that damping generates efficient exploration that, when combined with an anytime framework, produces high quality results [2]. However, without the anytime framework [30], a large number of iterations is required for damped Max-sum (DMS) to reach a high quality solution.

In this paper we contribute to the development of incomplete inference algorithms for solving DCOPs by proposing a novel degree of freedom for balancing between exploration and exploitation, in Max-sum. This degree of freedom, is the level of asymmetry in function-nodes representing constraints in the factor graph. The ability to control the level of asymmetry for each constraint is achieved by shifting a standard factor graph to an equivalent split constraint factor graph (SCFG), where each constraint is represented by two function-nodes instead of one.[1].

Our empirical evaluation reveals that the level of asymmetry in SCFGs can determine the level of exploration that the algorithm performs. When combining damping with low levels of asymmetry, Max-sum converges very fast to high quality solutions, without the need of an anytime framework.

As part of our investigation of this success, we investigate the effect of a split on a single constraint factor graph. We prove that when the cost table of the single function-node is split symmetrically, Max-sum is guaranteed to converge on the resulting cycle to the optimal solution, regardless of the damping factor used, and demonstrate that this is not the case when the constraint is split asymmetrically.

2 Related Work

The first paper to propose the use of Belief propagation for solving DCOPs and named it *Max-sum* was [5]. This work was followed by a number of studies that addressed the in-convergence of the algorithm on graphs that include multiple cycles, including [18], which proposed Bounded Max-sum and [31] which proposed Max-sum_ADVP.

Max-sum was applied to sensor nets both with mobile and static sensors [21, 6, 27], to supply chain management [1] and teams of rescue agents [17]. A number of papers made an attempt to overcome its most apparent drawback, the exponential computation of the content of messages sent by function-nodes [21, 10]. For specific applications with cardinality constraints, the *Tractable High Order Potentials* (THOP) method [23],

[1] A similar factor graph was used in [32] for representing asymmetric DCOPs.

which was adjusted to DCOPs, and implemented by [16] can be used. It reduces the computation runtime of function-nodes to $O(Klog(K))$, where K is the number of neighboring variable-nodes of the calculating function-node.

Max-sum was applied to asymmetric DCOPs in [32], by having each agent involved in a constraint hold a function-node representing its personal costs for that constraint. Thus, for each binary constraint there were two representing function-nodes. In contrast to other DCOP algorithms, Max-sum versions were found to maintain the quality of solutions they produce when applied to asymmetric problems. The main difference from our work is that, while they have used more than one function-node for a single constraint in order to represent the given natural structure of an asymmetric problem, we initiate a split of a standard function-node to two function-nodes representing the same constraint as an algorithmic method.

The possibility to encourage convergence of Max-sum by splitting nodes in the factor-graph was first suggested in [19]. This study investigated the theoretical conditions for convergence of the algorithm when using constant even splits. It further proposes a sequential version of the algorithm which, under some conditions, is guaranteed to converge to a local optimum and specifies the conditions in which this algorithm converges to the global optimum. They mention that by using damping with damping factor of $\frac{1}{n}$, this algorithm converges in a distributed synchronous execution as well. The main difference from our work is that we investigate the use of splitting nodes to control the level of symmetry in the factor graph and to balance exploitation and exploration, and therefore we investigate theoretical and empirical implications of constant even and uneven, random and limited random splits.

Recently, the effect of the use of damping within Max-sum was investigated [2]. It was found to immensely improve the quality of solutions traversed by Max-sum, and when combined with an anytime framework [30], produce high quality solutions. However, the use of Max-sum within an anytime framework, requires agents to exchange their value assignments in each iteration. This is not a requirement of the algorithm and therefore, such an exchange reduces the privacy of the algorithm. When an anytime framework is not used, a large number of iterations is required for Max-sum to find solutions with low costs.

3 Background

In this section we present background on DCOPs, the Max-sum algorithm and the damping method.

3.1 Distributed Constraint Optimization

Without loss of generality, in the rest of this paper we will assume that all problems are minimization problems (as in many DCOP papers, e.g., [11]). Thus, we assume that all constraints define costs and not utilities.[2]

[2] The inference algorithm for minimization problems is actually *Min-sum*. However, we will continue to refer to it as *Max-sum* since this name is widely accepted.

A $DCOP$ is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$. $\mathcal{A}$ is a finite set of agents $\{A_1, A_2, ..., A_n\}$. $\mathcal{X}$ is a finite set of variables $\{X_1,X_2,...,X_m\}$. Each variable is held by a single agent (an agent may hold more than one variable). $\mathcal{D}$ is a set of domains $\{D_1, D_2,...,D_m\}$. Each domain D_i contains the finite set of values that can be assigned to variable X_i. We denote an assignment of value $d \in D_i$ to X_i by an ordered pair $\langle X_i, d \rangle$. $\mathcal{R}$ is a set of relations (constraints). Each constraint $R_j \in \mathcal{R}$ defines a non-negative *cost* for every possible value combination of a set of variables, and is of the form $R_j : D_{j_1} \times D_{j_2} \times \ldots \times D_{j_k} \rightarrow \mathbb{R}^+ \cup \{0\}$. A *binary constraint* refers to exactly two variables and is of the form $R_{ij} : D_i \times D_j \rightarrow \mathbb{R}^+ \cup \{0\}$.[3] A *binary DCOP* is a DCOP in which all constraints are binary. A *partial assignment* (PA) is a set of value assignments to variables, in which each variable appears at most once. *vars(PA)* is the set of all variables that appear in PA, $vars(PA) = \{X_i \mid \exists d \in D_i \wedge \langle X_i, d \rangle \in PA\}$. A constraint $R_j \in \mathcal{R}$ of the form $R_j : D_{j_1} \times D_{j_2} \times \ldots \times D_{j_k} \rightarrow \mathbb{R}^+ \cup \{0\}$ is *applicable* to PA if each of the variables $X_{j_1}, X_{j_2}, \ldots, X_{j_k}$ is included in $vars(PA)$. The *cost of a partial assignment* PA is the sum of all applicable constraints to PA over the value assignments in PA. A *complete assignment* (or a *solution*) is a partial assignment that includes all the DCOP's variables ($vars(PA) = \mathcal{X}$). An *optimal solution* is a complete assignment with minimal cost.

For simplicity, we make the common assumption that each agent holds exactly one variable, i.e., $n = m$, and we concentrate on binary DCOPs, in which all constraints are binary. These assumptions are customary in DCOP literature (e.g., [13, 28]). In our description of algorithms and their properties, we will assume that there are no ties, i.e., that each entry in the constraint tables held by function-nodes has a unique numeric value. This can easily be achieved by using a method similar to the one proposed in [4], which we use in our empirical study.[4]

3.2 The Max-Sum algorithm

[5]Max-sum operates on a *factor-graph*, which is a bipartite graph in which the nodes represent variables and constraints [8]. Each variable-node representing a variable of the original DCOP is connected to all function-nodes that represent constraints, which it is involved in. Similarly, a function-node is connected to all variable-nodes that represent variables in the original DCOP that are involved in the constraint it represents. Variable-nodes and function-nodes are considered "agents" in Max-sum, i.e., they can send and receive messages, and perform computation.

A message sent to or from variable-node x (for simplicity, we use the same notation for a variable and the variable-node representing it) is a vector of size $|D_x|$ including a cost for each value in D_x. Before the first iteration, all nodes assume that all messages they previously received (in iteration 0) include vectors of zeros. A message sent from a variable-node x to a function-node f in iteration $i \geq 1$ is formalized as follows:

[3] We say that a variable is *involved* in a constraint if it is one of the variables the constraint refers to.

[4] For an example of the need to break ties in the factor-graph see [31].

[5] For lack of space we describe the algorithm briefly and refer the reader to more detailed descriptions in [5, 18, 31].

$$Q^i_{x\to f} = \sum_{f'\in F_x, f'\neq f} R^{i-1}_{f'\to x} - \alpha$$

where $Q^i_{x\to f}$ is the message variable-node x intends to send to function-node f in iteration i, F_x is the set of function-node neighbors of variable-node x and $R^{i-1}_{f'\to x}$ is the message sent to variable-node x by function-node f' in iteration $i-1$. α is a constant that is reduced from all costs included in the message (i.e., for each $d \in D_x$) in order to prevent the costs carried by messages throughout the algorithm run from growing arbitrarily.

A message sent from a function-node f to a variable-node x in iteration i includes for each value $d \in D_x$:

$$min_{PA_{-x}} cost(\langle x, d\rangle, PA_{-x})$$

where PA_{-x} is a possible combination of value assignments to variables involved in f not including x. The term $cost(\langle x, d\rangle, PA_{-x})$ represents the cost of a partial assignment $a = \{\langle x, d\rangle, PA_{-x}\}$, which is: $f(a) + \sum_{x'\in X_f, x'\neq x, \langle x', d'\rangle \in a} Q^{i-1}_{x'\to f}.d'$, where $f(a)$ is the original cost in the constraint represented by f for the partial assignment a, X_f is the set of variable-node neighbors of f, and $Q^{i-1}_{x'\to f}.d'$ is the cost that was received in the message sent from variable-node x' in iteration $i-1$, for the value d' that is assigned to x' in a. x selects its value assignment $\hat{d} \in D_x$ following iteration k as follows:

$$\hat{d} = \arg\min_{d\in D_x} \sum_{f\in F_x} R^k_{f\to x}.d$$

Introducing Damping into Max-sum In order to add damping to Max-sum a parameter $\lambda \in [0, 1)$ is used. Before sending a message in iteration k an agent performs calculations as in standard Max-sum. Denote by $\widehat{m^k_{i\to j}}$ the result of the calculation made by agent A_i of the content of a message intended to be sent from A_i to agent A_j in iteration k. Denote by $m^{k-1}_{i\to j}$ the message sent by A_i to A_j at iteration $k-1$. The message sent from A_i to A_j in iteration k is calculated as follows:

$$m^k_{i\to j} = \lambda m^{k-1}_{i\to j} + (1-\lambda)\widehat{m^k_{i\to j}}$$

Thus, λ expresses the weight given to previously performed calculations with respect to the most recent calculation performed. Moreover, when $\lambda = 0$ the resulting algorithm is standard Max-sum.

Applying Max-sum to Asymmetric Problems When Max-sum is applied to an asymmetric problem, the representing factor graph has each (binary) constraint represented by two function-nodes, one for each part of the constraint held by one of the involved

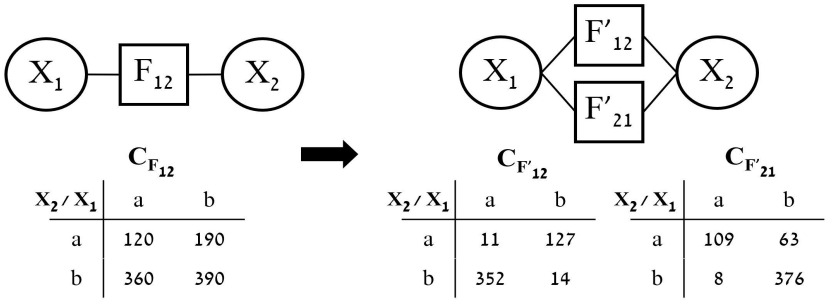

Fig. 1. An acyclic DCOP factor graph (on the left) and its equivalent SCFG (on the right).

agents. Each function-node is connected to both variable-nodes representing the variables involved in the constraint [32]. Figure 1 presents two equivalent factor graphs that include two variable-nodes, each with two values in its domain, and a single binary constraint. On the left, the factor graph represents a (symmetric) DCOP including a single constraint between variables X_1 and X_2, hence, it includes a single function node representing this constraint. On the right, the equivalent factor graph representing the equivalent asymmetric DCOP is depicted. It includes two function-nodes, representing the parts of the constraint held by the two agents involved in the asymmetric constraint. Thus, the cost table in each function-node includes the asymmetric costs that the agent holding this function-node incurs. The factor graphs are equivalent since the sum of the two cost tables held by the function-nodes representing the constraints in the factor graph on the right, is equal to the cost table of the single function-node representing this constraint in the factor graph on the left (see [32] for details).

Exploration and Exploitation in Max-sum In local search algorithms, an agent commonly selects an assignment that minimizes the cost according to the information available to it, i.e., it exploits the information. On the other hand, the agent can select an assignment that does not minimize (and even enlarges) its cost, hoping this will allow it to find assignments with lower cost in following iterations. Such actions, which do not result in immediate benefit are aka exploration. In distributed local search, even if each agent performs only exploitive actions, concurrent actions by a number of agents can generate exploration, e.g., in DSA when neighboring agents replace assignments concurrently.

In inference algorithms such as Max-sum agents do not propagate assignments. However, each variable-node can select an assignment at each iteration based on the costs it receives. Thus, as in the distributed stochastic algorithm (DSA) [29], a global view that examines the quality of the global assignment that can be inferred in each iteration, can reveal whether the agents are improving the global assignment or involuntarily, selecting assignments that are with higher costs and hence, exploring.

4 Split Constraint Factor Graphs

Damping can be used as a degree of freedom, to balance exploration and exploitation in Max-sum [2, 3]. However, when using damping, thousands of iterations are required for

the algorithm to find high quality solutions. By combining damped Max-sum (DMS) with an anytime framework [30], the number of iterations required in order to find high quality solutions is an order of magnitude smaller.

We aim at shortening the process of producing high quality solutions by Max-sum and eliminating the dependency on the anytime framework, which requires agents to share value assignments in contrast to the requirements of the algorithm. Thus, we propose an additional degree of freedom, the level of asymmetry of constraints in the factor graph. To this end, we propose the use of *Split Constraint Factor Graphs* (SCFGs) in which each constraint that was represented by a single function-node in the original factor graph, is represented by two function-nodes. The SCFG is equivalent to the original factor graph, if the sum of the cost tables of the two function-nodes representing each constraint in the SCFG is equal to the cost table of the single function-node representing the same constraint in the original factor graph. By tuning the similarity between the two function-nodes representing the same constraint we can determine the level of asymmetry in the SCFG.

Formally, an SCFG G' is equivalent to a factor graph G, if their sets of variable-nodes (and their domains) are equal, and if for each function-node F in G with cost table C_F, there exist function-nodes F' and F'' in G', which are connected to the same variable-nodes as F, and their constraint tables satisfy $C_F = C_{F'} + C_{F''}$. Thus, each SCFG has a single equivalent DCOP factor graph to which its function-nodes' constraints sum up to, however, a standard DCOP factor graph can have countless equivalent SCFGs.

Returning to the example portrayed in Figure 1, given a standard factor graph of a symmetric DCOP as presented on the left, the cost table of the function-node F_{12} in this factor graph is split using a different random ratio for each entry, thus generating a new equivalent SCFG (on the right) containing 2 function-nodes F'_{12} and F'_{21}.

We differentiate between constant SCFGs, in which constraint costs are split according to a predetermined constant ratio, and random SCFGs, in which each cost in each constraint table is split according to a randomly generated ratio.

It is important to notice the difference between the use we make of SCFGs and the use made of factor graphs with two function-nodes representing a constraint in [32]. There, they assume this type of factor graph is required to represent the asymmetric state of the world, while we assume the input problem is symmetric, and the generation of the SCFG is an algorithmic action (represented by the small black arrow between the factor graphs in Figure 1).

5 Splitting a Single Constraint

In Section 6 we present empirical evidence of the success of applying DMS to symmetric SCFGs and SCFGs with minor asymmetry. As part of an attempt to explain this success we investigate the different effect of a symmetric split and an asymmetic split in a single constraint factor graph.[6]

[6] Although Lemmas 1 and 2 can be implied from Lemma 3, for simplicity of presentation we enclose all three, provide complete proof for Lemma 1, and intuitive explanations how to generalize the proof so it will apply to Lemmas 2 and 3.

Lemma 1. *On a factor graph with two variable-nodes X_1 and X_2 and two identical function-nodes, F_{12} and F_{21}, each connected to both variable-nodes, Max-sum is guaranteed to converge to the optimal solution after the first iteration.*

Proof: We prove by induction on the number of iterations. If the smallest cost c in the cost table held by both function-nodes[7] is in entry i, j representing the cost when the value assignments selected are the i'th value in the domain of X_1 and the j'th value in the domain of X_2, then in the first iteration of the algorithm, each function-node will send to X_1 a vector where the i'th cost in it is c, and the messages sent to X_2 will include c in the j'th cost of the vector. All other costs in these vector must be larger than c. Thus, following the first iteration values i and j are selected. The induction assumption is that in the k'th iteration[8], $k > 1$, the i'th cost in the messages sent to X_1 and the j'th cost in the messages sent to X_2 will be $\frac{k+1}{2}c$, while all other costs in these vectors will be larger. In the next iteration $(k + 1)$, X_1 and X_2 will send forward the message received from each function-node, to the other function-node. In iteration $k+2$ in the vector sent in each message to X_1 the i'th cost will include a sum of the smallest cost in the vector received, which according to the induction assumption is $\frac{k+1}{2}c$, and the smallest cost in the cost table, c. Thus, the resulting cost, which must still be smallest in the vector, is $\frac{k+1}{2}c + c = \frac{k+3}{2}c$. For similar reasons, the smallest cost in the vectors sent to X_2 in iteration $k + 2$ are the j'th costs and they are equal to $\frac{k+3}{2}c$. □

Lemma 2. *On a factor graph with two variable-nodes X_1 and X_2 and two function-nodes, F_{12} and F_{21}, each connected to both variable-nodes. If for any real number m, the cost tables held by the two function-nodes maintain the relation $C_{F_{12}} = mC_{F_{21}}$ then Max-sum is guaranteed to converge to the optimal solution after the first iteration.*

The main difference is that in the first iteration, X_1 is sent one message with the i'th cost equal to c and another where it is equal to mc and this is true for the j'th costs in the messages sent to X_2. In iteration k the i'th cost in a message sent to X_1 and the j'th cost in the messages sent to X_2 will include alternating summations of c and mc, while all other costs in the vectors will include corresponding summations of other (larger) numbers.

Lemma 3. *On a factor graph with two variable-nodes X_1 and X_2 and two function-nodes, F_{12} and F_{21}, each connected to both variable-nodes. If for any real number m, the cost tables held by the two function-nodes maintain the relation $C_{F_{12}} = mC_{F_{21}}$ then DMS is guaranteed to converge to the optimal solution after the first iteration, regardless of the damping factor being used.*

In each iteration the costs calculated are multiplied by $1 - \lambda$ and added to the previous message sent, multiplied by λ. Thus, in the first iteration the costs we mention in the proofs for the lemmas above will be multiplied by $1 - \lambda$, but the smallest entries will remain i for X_1 and j for X_2. The same will be true for the $k + 2$ iteration in the induction step. The smallest entry will not change as a result of multiplying all messages by the same factor.

[7] Recall that we assumed in Section 3.1 that there are no ties, so such a cost is unique.

[8] Without loss of generality, we assume that k is odd. If it was even, then the assumption was that the cost is $\frac{k}{2}c$.

Proposition 1. *If DMS is applied to a constant SCFG generated from a factor graph including two variable-nodes and a single function-node representing the constraint among them, it will converge after the first iteration, regardless of the damping factor used.*

Proof: Immediate from Lemma 3.

It is important to notice that previous works on the behavior of Belief propagation on single cycle graphs, only prove the optimality of the solutions obtained when the algorithm converges [25]. Moreover, when damping is used (as in DMS), the algorithm might converge to sub-optimal solutions on single cycle graphs. Thus, the fact that on constant split cycles both Max-sum and DMS are guaranteed to converge after a single cycle to the optimal solution is novel and significant.

In contrast, when Max-sum is applied to an SCFG generated by a random split of a single constraint, function-nodes might choose in their calculations minimal costs, which do not correspond with identical value assignments, and further calculations may be based on inconsistent assignment choices for the same variable, producing impossible belief costs for the variable-nodes. Hence, Max-sum does not necessarily converge to the optimal solution. We empirically observed this behavior in experiments on 20,000 randomly generated, single cycle factor graphs, in which Max-sum did not converge to the optimal solution in many problem instances. Interestingly, DMS always converged, but not necessarily to the optimal solution. For example, when applied to the *SCFG* portrayed in Figure 1, standard Max-sum alternates endlessly between solutions of costs 120, 190 and 360. The optimal solution incurs a cost of 120. DMS, however, converges after 18 iterations to the suboptimal solution of cost 190.

Nevertheless, as demonstrated in Section 6, in *SCFG*s based on DCOPs containing multiple cycles, this pathology can induce exploration, which can be adjusted by determining the level of asymmetry of split constraints, and exploited by using a high damping factor.[9]

6 Experimental Evaluation

In order to investigate the advantages of the use of SCFGs when applying Max-sum to DCOPs, we present a set of experiments comparing standard Max-sum and DMS, both when applied to different SCFGs and two versions that guarantee convergence: Bounded Max-sum and Max-sum_ADVP (for detailed descriptions of these algorithms see [18, 31]). We also include in our experiments the results of the well known DSA algorithm (we use type C with $p = 0.7$ [29]), in order to give an insight on the quality of the results, in comparison with local search DCOP algorithms.

We evaluated the algorithms on random uniform minimization DCOPs and on structured and realistic problems, i.e., graph coloring, meeting scheduling and scale-free nets (see details below). At each experiment we randomly generated 50 different problem instances and ran the algorithms for 2,000 iterations on each of them. The results presented in the graphs are an average of those 50 runs. For each iteration we present the

[9] further insights on the relation between the success of our empirical results and the properties presented in this section are detailed in Section 6.2.

sum of costs of the constraints involved in the assignment that would have been selected by each algorithm at that iteration. The statistical significance of the results was verified using paired t-tests with significance level of p=0.05. In order to maximize the benefit of the algorithms exploration property we implemented all algorithms within the anytime framework proposed in [30]. This allowed us to report for each of the algorithms the best result it traverses within 2,000 iterations. Also, in all versions of Max-Sum, we used personal value preferences, selected randomly for the purpose of tie breaking, as was suggested in [4].

All problems were formulated as minimization problems. The uniform random problems were generated by adding in each problem a constraint for each pair of agents (variables) with probability p_1. For each constrained pair we set a cost for each combination of value assignments, selected uniformly between 100 and 200.[10] Each problem included 50 variables with 10 values in each domain.

Graph coloring problems include random constraint graph topologies and all constraints $R_{ij} \in \mathcal{R}$ are "not-equal" cost functions where an equal assignment of neighbors in the graph incurs a cost of 50 and non equal value assignments incur 0 cost. Following the literature, we used $p_1 = 0.05$ and three values (i.e., colors) in each domain to generate these problems, which included 50 agents [29, 5, 30].

Scale-free network problems were generated using the Barabási–Albert (BA) model with an initial set of 7 connected agents, and additional 43 agents, which were added sequentially and connected to 3 other agents with a probability proportional to the number of links that the existing agents already had. The rest of the problem parameters were identical to the random uniform problems.

Meeting scheduling problems included ninety agents, which scheduled 20 meetings into 20 time slots. Each agent was a participant in two randomly chosen meetings. For each pair of meetings, a travel time was chosen uniformly at random between 6 and 10. When the difference between the time slots of two meetings is less than the travel time between them, participants in both meetings are overbooked, and a cost equal to the number of overbooked agents is incurred.

Space limitations do not allow us to present all results obtained in our comprehensive experimental study, therefore we focus on the most significant results. We present results when applying Max-sum and DMS to two constant SCFGs, one splitting the costs evenly among the two function-nodes (0.5 version) and one in which the split had 95% of the cost in one function-node and 5% on the other (0.95 version). For random SCFGs we specify the range of costs from which the cost for the first function-node was selected, i.e., version $0.4 - 0.6$ includes SCFGs where for each entry including cost c in the original constraint cost table, the cost for the corresponding entry of the first function-node was selected randomly between $0.4c$ and $0.6c$. Following [2] we present results of DMS with $\lambda = 0.9$. Our experiments with other λ values (0.5 and 0.7) validated that this version indeed performs best.

Figure 2 presents the costs per iteration and the costs of the anytime solution per iteration, when applying Max-sum, DSA, Bounded Max-sum and Max-sum_ADVP to

[10] This range was selected so that the numbers do not become too small and due to precision, generate distorted SCFGs. Obviously, if the input costs are between 0 and 100, adding 100 to each cost can be the first step of the splitting method.

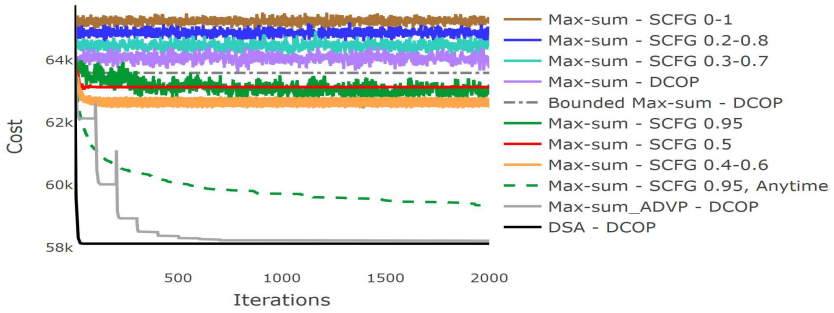

Fig. 2. Solution costs for Max-sum solving SCFGs of random uniform problems.

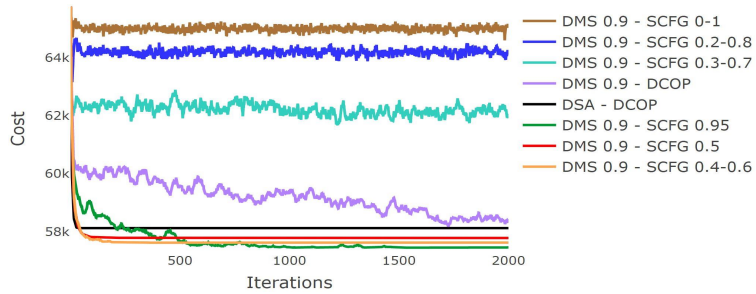

Fig. 3. Solutions costs for DMS solving SCFGs of random uniform problems.

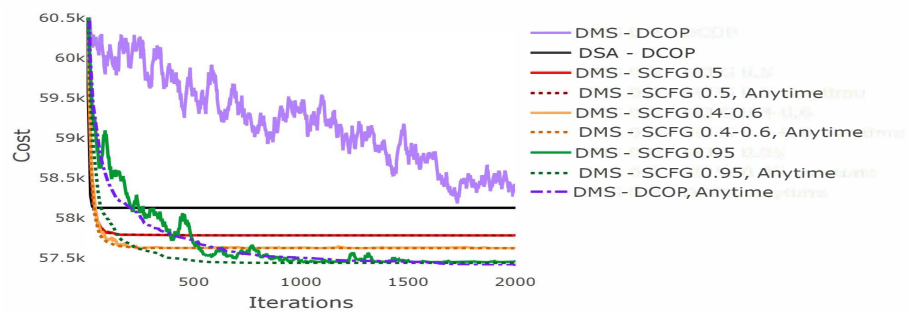

Fig. 4. Solution and anytime costs for DMS solving SCFGs of random uniform problems.

SCFGs. For constant SCFGs and random SCFGs where costs were selected from a small range, the solutions found have lower cost than standard Max-sum. However, the results are of much lower quality (higher costs) than DSA. In order to avoid density we only depict the best anytime results among all versions of the algorithm presented in this graph, which were obtained when applying Max-sum to constant SCFGs 0.95. They are significantly better than the costs per iteration when applying Max-sum to any of the SCFGs but are far worse than the results produced by DSA and Max-sum_ADVP. The results Max-sum produces on random SCFGs $0-1$, are consistent with the results in [32], where only random splits were considered and damping was not used.

Figure 3 presents results for DMS applied to the same SCFGs. While the trends are the same, and the best results are achieved using constant SCFGs and random SCFGs with a small range for selecting the first cost, here the best results significantly out-

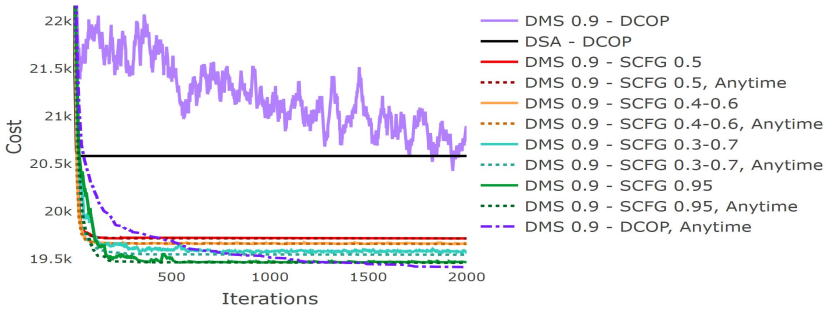

Fig. 5. Costs per iteration and anytime costs for DMS solving SCFGs of scale free nets.

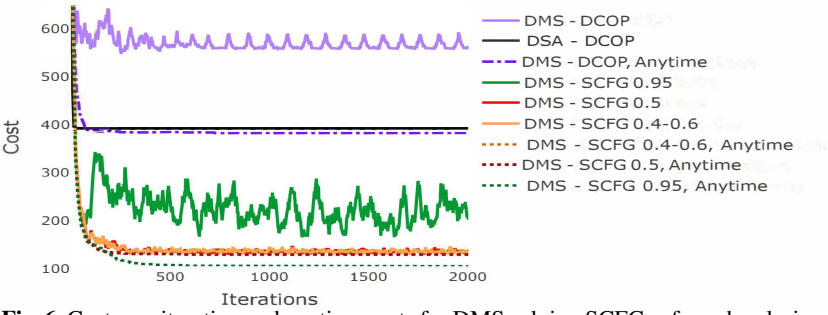

Fig. 6. Costs per iteration and anytime costs for DMS solving SCFGs of graph coloring problems.

perform DSA. When using SCFG 0.5 and SCFG $0.4-0.6$ the algorithm finds high quality solutions in a small number of iterations. When using SCFG 0.95, the algorithm performs more exploration, and after approximately 500 iterations it averagely finds solutions of higher quality than when the 0.5 and the $0.4-0.6$ SCFGs are used. Notice that all results presented in Figure 3 are cost per iteration and not the anytime costs. Figure 4 provides a closer look on the differences of the most successful versions of the algorithm and their anytime results. The anytime results of DMS on standard factor graphs are similar to the results per iteration of DMS on the SCFG 0.95 version. However, the anytime results of the 0.95 version converge must faster. When applied to SCFG 0.5 the algorithm converges very fast, yet the costs of the solutions are higher. When applied to random SCFGs $0.4-0.6$, solutions with lower costs are also reached very fast. The small level of additional exploration gives an advantage in this case.

The general trends were similar for denser uniform random problems with $p_1 = 0.7$ and for all other problem types, therefore, only selective graphs that give a closer view on the results of the most successful versions of DMS for these problems are presented. On scale free nets (Figure 5), both the cost per iteration and the anytime costs of solutions found by DMS when applied to constant SCFGs and random SCFGs $0.4-0.6$, converge much faster than the anytime result of DMS solving standard factor graphs. In addition, the costs per iteration and the anytime costs when applying DMS to random SCFGs $0.3-0.7$ of scale free nets were lower than for constant SCFGs 0.5 and random SCFGs $0.4-0.6$, thus, we depict them as well. On constant SCFGs 0.95, DMS found high quality solutions very fast.

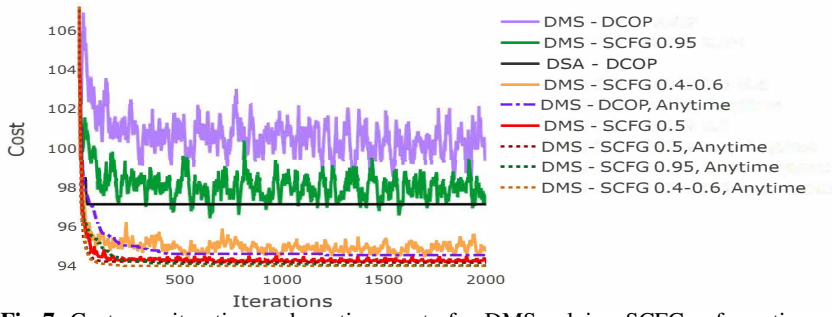

Fig. 7. Costs per iteration and anytime costs for DMS solving SCFGs of meeting scheduling problems.

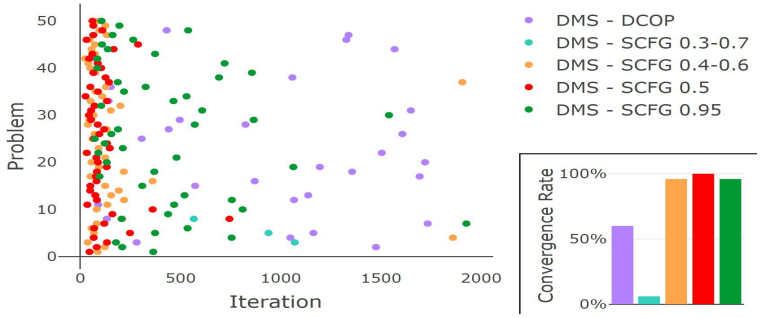

Fig. 8. Number of iterations for convergence and convergence rate on relatively sparse random uniform problems $p_1 = 0.2$.

For graph coloring problems (Figure 6) DMS on standard factor graphs produces solutions with relatively high costs, and its anytime results are similar to the results of DSA. On the other hand the solutions found by DMS when applied to constant SCFGs 0.5 and to random SCFGs $0.4-0.6$ are of significantly lower costs per iteration and anytime costs. The 0.95 version performs more exploration and the resulting anytime costs are lowest. The results for the meeting scheduling problems (Figure 7) show a similar trend. When DMS is applied to constant SCFGs 0.95 it performs more exploration than when applied to SCFGs 0.5 and $0.4 - 0.6$.

Figure 8 presents the time for convergence for each of the 50 runs of each of the algorithms and the percentage of problems on which the algorithm converged among the 50 runs on each problem type. The random SCFGs with larger ranges than $0.3-0.7$ do not appear because they never converge. It is clear that constant SCFGs and the $0.4 - 0.6$ random version have higher convergence rate than when DMS is applied to standard factor graphs. Moreover, it is also apparent that the constant 0.5 version converges very fast. Both the constant SCFG 0.95 and the random SCFG $0.4 - 0.6$ versions converge slower and with slightly lower rates than the SCFG 0.5 version. Thus, they have more opportunities to perform exploration, which can explain the advantage they have in solution quality, as can be seen in Figure 4.

Figure 9 presents the time for convergence and the convergence rate for graph coloring problems. Here, it is clear that fast convergence prevents the algorithm from finding solutions with low cost, when applied to standard factor graphs. On the other hand,

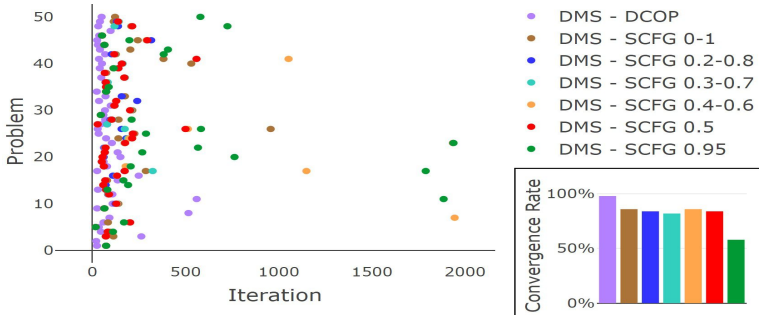

Fig. 9. Number of iterations for convergence and convergence rate on graph coloring problems.

when applied to SCFGs, DMS performs balanced exploration, which results in solutions with low costs. The constant SCFG 0.95 version triggers more exploration that results in lower anytime costs.[11]

6.1 Runtime overhead

If we consider each node in the factor graph as a separate agent, the overhead in runtime caused by splitting function-nodes is only for the variable-nodes, since they need to produce and send a double amount of messages. On the other hand, splitting function-nodes does not create a delay in the computation of function-nodes, since they are performed concurrently. However, commonly it is assumed that the role of the nodes in the factor-graphs are performed by the original (”real”) DCOP agents. Thus, when adding function-nodes to the graph we increment the runtime of each iteration. We note that a factor 2 increase in runtime can be easily achieved by having each agent that performed the role of a function-node in the original factor graph to perform the role of the two function-nodes resulting from its split. Our results indicate that for Symmetric SCFGs and SCFGs with limited randomness, the convergence of DMS is orders of magnitude faster than when using standard factor graphs and that is still true when each iteration takes a double amount of time.

6.2 Discussion

Our results indicate success in combining damping and asymmetry for balancing exploration and exploitation. Damping alone is enough to trigger Max-sum to explore solutions with low cost, however, it does not converge to high quality solutions within two thousand iterations. However, when using constant SCFGs split evenly (0.5), convergence is achieved within a few tens of iterations. The use of an uneven split of a constant SCFG (0.95 version) or random SCFGs with small ranges $(0.4 - 0.6)$, allows limited exploration that results in solutions with both per iteration and anytime lower costs.

[11] For lack of space we do not present convergence graphs for the other problems. As expected the meeting scheduling convergence results were similar to graph coloring while the results for the other problem types were similar to the convergence results of the sparse uniform random problems.

In order to explain this success one needs to look back at the results presented in Section 5. In problems with two variables and a single constraint, when splits are symmetric both Max-sum and DMS converge after a single iteration. Thus, in the general case, the difference between a single function-node representing a constraint and its symmetric split to two function-nodes is the ratio between the costs sent from the variable-nodes to the function-nodes and the costs in the function-nodes' tables. Consider a variable-node v and its neighboring function-node f in factor graph G, which is symmetrically split to f' and f'' in factor graph G' (for simplicity assume that other function-nodes are not split). Let vc be the vector that holds the sum of costs v has received from function-node neighbors, which are not f, in iteration i. In G, in iteration $i+1$, vc will be sent to f. In G', vc will be sent to both function-nodes f' and f''. However, each cost in the table of f is cut by half in the tables of f' and f''. Thus, f' and f'' will make different calculations than f giving more consideration to the differences between costs in vc.

This phenomenon allows DMS to converge faster. Damping allows Max-sum to explore high quality solutions because it reduces the effect of multiple counting of information as observed by Pearl [12, 3]. On the other hand, it slows the aggregation of costs in the propagated cost vectors, and thus the differences between costs in the vectors are less pronounced in the beginning of the run. In contrast, the function-node cost tables are fixed throughout the run. Thus, damping delays assignment replacements, which are required for convergence. Symmetric SCFGs reduce the differences in the cost tables by half but do not reduce the costs in the propagated vectors and thus, allow the required changes to take place faster.

On the other hands, as demonstrated in Section 5, random splits might generate oscillations. When such oscillations occur in multiple cycles in the graph, the beliefs propagated are inconsistent and prevent convergence.

7 Conclusion

We introduced a novel degree of freedom for balancing exploration and exploitation when using Max-sum for solving DCOPs, the level of asymmetry in the factor graph. To this end, we proposed to shift standard factor graphs representing a DCOP to equivalent split constraint factor graphs (SCFGs), in which each constraint is represented by two function-nodes. The level of asymmetry in SCFGs is determined by the similarity between table costs of function-nodes representing the same constraint.

We proved that Max-sum is guaranteed to converge to the optimal solution on cycles generated as a result of a constant split of a single constraint factor graph, regardless of the constant fraction and the damping factor used. This is in contrast to the general case where Max-sum is not guaranteed to converge on single cycle factor-graphs, and DMS might converge to a sub-optimal solution. Empirical results indicate that by tuning the two degrees of freedom, the damping factor and the level of asymmetry, Max-sum can produce solutions of high quality within a small number of iterations, even when an anytime framework cannot be used. When the level of exploration is too high, e.g., without damping, the algorithm fails to find solutions of high quality. On the other hand, limited exploration results in solutions of higher quality than immediate convergence.

References

1. Chli, M., Winsper, M.: Using the max-sum algorithm for supply chain emergence in dynamic multiunit environments. IEEE trans. on Systems, Man, and Cybernetics **45(3)**, 422435 (2015)
2. Cohen, L., Zivan, R.: Max-sum revisited: The real power of damping. In: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017. pp. 1505–1507 (2017)
3. Cohen, L., Zivan, R.: Max-sum revisited: The real power of damping. In: Workshop on Multi Agent Optimization (OptMAS) at AAMAS 2017, São Paulo, Brazil, May, 2017. pp. 1505–1507 (2017)
4. Farinelli, A., Rogers, A., Petcu, A., Jennings, N.: Decentralised coordination of low-power embedded devices using the max-sum algorithm. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2. pp. 639–646. International Foundation for Autonomous Agents and Multiagent Systems (2008)
5. Farinelli, A., Rogers, A., Petcu, A., Jennings, N.R.: Decentralized coordination of low-power embedded devices using the max-sum algorithm. In: AAMAS. pp. 639–646 (2008)
6. Farinelli, A., Rogers, A., Jennings, N.R.: Agent-based decentralised coordination for sensor networks using the max-sum algorithm. Autonomous Agents and Multi-Agent Systems **28**(3), 337–380 (2014)
7. Gershman, A., Meisels, A., Zivan, R.: Asynchronous forward bounding. J. of Artificial Intelligence Research **34**, 25–46 (2009)
8. Kschischang, F.R., Frey, B.J., Loeliger, H.A.: Factor graphs and the sum-product algorithm. IEEE Transactions on Information Theory **47:2**, 181–208 (Febuary 2001)
9. Lazic, N., Frey, B., Aarabi, P.: Solving the uncapacitated facility location problem using message passing algorithms. In: International Conference on Artificial Intelligence and Statistics. pp. 429–436 (2010)
10. Macarthur, K.S., Stranders, R., Ramchurn, S.D., Jennings, N.R.: A distributed anytime algorithm for dynamic task allocation in multi-agent systems. In: AAAI (2011)
11. Modi, P.J., Shen, W., Tambe, M., Yokoo, M.: Adopt: asynchronous distributed constraints optimizationwith quality guarantees. Artificial Intelligence **161:1-2**, 149–180 (January 2005)
12. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Francisco, California (1988)
13. Petcu, A., Faltings, B.: A scalable method for multiagent constraint optimization. In: IJCAI. pp. 266–271 (2005)
14. Petcu, A., Faltings, B.: Approximations in distributed optimization. In: van Beek, P. (ed.) CP 2005, LNCS 3709. pp. 802–806 (2005)
15. Pretti, M.: A message-passing algorithm with damping. Journal of Statistical Mechanics: Theory and Experiment **11**, P11008 (2005)
16. Pujol-Gonzalez, M., Cerquides, J., Meseguer, P., Rodríguez-Aguilar, J.A., Tambe, M.: Engineering the decentralized coordination of uavs with limited communication range. Advances in Artificial Intelligence pp. 199–208 (2013)
17. Ramchurn, S.D., Farinelli, A., Macarthur, K.S., Jennings, N.R.: Decentralized coordination in robocup rescue. Computer J. **53**(9), 1447–1461 (2010)
18. Rogers, A., Farinelli, A., Stranders, R., Jennings, N.R.: Bounded approximate decentralized coordination via the max-sum algorithm. Artificial Intelligence **175**(2), 730–759 (2011)
19. Ruozzi, N., Tatikonda, S.: Message-passing algorithms: Reparameterizations and splittings. IEEE Trans. Information Theory **59**(9), 5860–5881 (2013)
20. Som, P., Chockalingam, A.: Damped belief propagation based near-optimal equalization of severely delay-spread uwb mimo-isi channels. In: Communications (ICC), 2010 IEEE International Conference on. pp. 1–5. IEEE (2010)

21. Stranders, R., Farinelli, A., Rogers, A., Jennings, N.R.: Decentralised coordination of mobile sensors using the max-sum algorithm. In: IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009. pp. 299–304 (2009)
22. Tarlow, D., Givoni, I., Zemel, R., Frey, B.: Graph cuts is a max-product algorithm. In: Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (2011)
23. Tarlow, D., Givoni, I., Zemel, R.: Efficient message passing with high order potentials. AISTATS **9**, 812–819 (2010)
24. Teacy, W.T.L., Farinelli, A., Grabham, N.J., Padhy, P., Rogers, A., Jennings, N.R.: Max-sum decentralized coordination for sensor systems. In: AAMAS. pp. 1697–1698 (2008)
25. Weiss, Y.: Correctness of local probability propagation in graphical models with loops. Neural Computation **12**(1), 1–41 (2000)
26. Yanover, C., Meltzer, T., Weiss, Y.: Linear programming relaxations and belief propagation - an empirical study. Journal of Machine Learning Research **7**, 1887–1907 (2006)
27. Yedidsion, H., Zivan, R., Farinelli, A.: Explorative max-sum for teams of mobile sensing agents. In: International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '14, Paris, France, May 5-9, 2014. pp. 549–556 (2014)
28. Yeoh, W., Felner, A., Koenig, S.: Bnb-adopt: An asynchronous branch-and-bound dcop algorithm. Artificial Intelligence Research (JAIR) **38**, 85–133 (2010)
29. Zhang, W., Xing, Z., Wang, G., Wittenburg, L.: Distributed stochastic search and distributed breakout: properties, comparishon and applications to constraints optimization problems in sensor networks. Artificial Intelligence **161:1-2**, 55–88 (January 2005)
30. Zivan, R., Okamoto, S., Peled, H.: Explorative anytime local search for distributed constraint optimization. Artificial Intelligence **211** (2014)
31. Zivan, R., Parash, T., Cohen, L., Peled, H., Okamoto, S.: Balancing exploration and exploitation in incomplete min/max-sum inference for distributed constraint optimization. Autonomous Agents and Multi-Agent Systems **31**(5), 1165–1207 (2017)
32. Zivan, R., Parash, T., Naveh, Y.: Applying max-sum to asymmetric distributed constraint optimization. In: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015. pp. 432–439 (2015)