Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence

# Distributed Multiplicative Weights Methods for DCOP

**Daisuke Hatano**
National Institute of Informatics
JST, ERATO, Kawarabayashi Large Graph Project
hatano@nii.ac.jp

**Yuichi Yoshida**
National Institute of Informatics, and
Preferred Infrastructure, Inc.
yyoshida@nii.ac.jp

### Abstract

We introduce a new framework for solving distributed constraint optimization problems that extend the domain of each variable into a simplex. We propose two methods for searching the extended domain for good assignments. The first one relaxes the problem using linear programming, finds the optimum LP solution, and rounds it to an assignment. The second one plays a cost-minimization game, finds a certain kind of equilibrium, and rounds it to an assignment. Both methods are realized by performing the multiplicative weights method in a distributed manner. We experimentally demonstrate that our methods have good scalability, and in particular, the second method outperforms existing algorithms in terms of solution quality and efficiency.

## Introduction

In the wake of the computational sustainability project, the importance of distributed cooperative problem solving to deal with enormous sizes such as a smart grid is rapidly increasing in AI communities. The *distributed constraint optimization problem* (DCOP for short) is arguably the most studied problem in this setting, where the goal is to find an assignment that minimizes the total sum of costs incurred by (local) cost functions. Since it takes a prohibitively long time to exactly solve DCOP, we need to resort to incomplete algorithms, and a plethora of incomplete algorithms have been proposed in the literature, such as local search based algorithms (Maheswaran, Pearce, and Tambe 2004; Zhang et al. 2005), inference based algorithms (Farinelli et al. 2008), graph based algorithms (Bowring et al. 2008; Kiekintveld et al. 2010), divide-and-coordinate based algorithms (Vinyals, Rodriguez-Aguilar, and Cerquides 2010; Hatano and Hirayama 2013), and sampling based algorithms (Ottens, Dimitrakakis, and Faltings 2012; Nguyen, Yeoh, and Lau 2013).

In this paper, we present a novel approach for DCOP, in which the finite domain of a variable is extended to the $d$-dimensional simplex, where $d$ is the size of the domain. We propose two methods that search the extended domain for good assignments, both based on the multiplicative weights method, which is a versatile algorithm that can be used in machine learning, optimization, and game theory (see (Arora, Hazan, and Kale 2012; Schapire 2003) for surveys).

The first method, called DMW-LP (distributed multiplicative weights method for solving linear programming) utilizes linear programming (LP for short) relaxations. In this method, the agents cooperatively solve the LP relaxation of the given DCOP instance, and round the obtained LP solution to integer values. Though LPs are convex optimization problems and they are known to be solvable using the multiplicative weights method, we need to modify the algorithm so that it runs in a distributed manner. We prove that our method converges to an optimal LP solution. As DMW-LP computes the LP value, we can use this value as a lower bound on the (integer) optimal value.

The second method, called DMW-Game (distributed multiplicative weights method for solving games) plays a *cost-minimization game* to solve DCOP. In this game, each player associated with a variable keeps providing probability distributions over its domain, and tries to minimize the *regret*, which is the average additional cost incurred by the probability distributions against the strategy of outputting a best single value all the time. We can make the regret of each agent arbitrarily small by utilizing the multiplicative weights method. Finally, we round the obtained probability distributions to integer values. We prove that our method converges to a certain kind of equilibrium, called a *coarse correlated equilibrium*.

We empirically compare our methods with previous state-of-the-art methods. We demonstrate that our methods are scalable, and that DMW-Game outperforms other methods in terms of solution quality and efficiency.

## Preliminaries

For a positive integer $t$, $[t]$ denotes the set $\{1, 2, \ldots, t\}$. We use bold symbols to denote vectors. Let $\triangle_k$ be the $k$-dimensional simplex, i.e., $\triangle_k = \{(x_1, \ldots, x_k) \mid x_1 + \cdots + x_k = 1\}$. Let $\blacktriangle_k$ denote the interior of $\triangle_k$. For two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ denotes their inner product. For a probability distribution $\mathcal{P}$ over a domain $D$, $a \sim \mathcal{P}$ means that we sample a value $a \in D$ from the distribution $\mathcal{P}$.

**Algorithm 1** The multiplicative weights method

**Input:** A decision list $D$ and a parameter $T$
**Output:** $\boldsymbol{p}^0, ..., \boldsymbol{p}^t$ satisfying Theorem 1.
Fix $\eta \leq 1/2$. For each decision $a$, set $w_a^1 := 1$.
**for** $t = 1, 2, ..., T$ **do**
Choose decision $a$ with probability $p_a^t := \frac{w_a^t}{\|\boldsymbol{w}^t\|_1}$.
Observe the costs of the decisions $c_1^t, \ldots, c_d^t$.
For every decision $a$, set $w_a^{t+1} \leftarrow w_a^t(1 - \eta c_a^t)$.

## Distributed Constraint Optimization Problems

In the *constraint optimization problem* (COP for short), an instance is defined as $\phi = (X, E, F, D)$, where a set $X = \{x_1, x_2, \ldots, x_n\}$ of *variables*, a set of edges $E$ over variables, and a set $F = \{f_{i,j}\}_{(i,j)\in E}$ of *binary cost functions*, where each variable $x_i \in X$ has a finite *domain* $D_i$ from which it takes its value, and each function $f_{i,j} : D_i \times D_j \to \mathbb{R}^+$ returns a non-negative cost. The goal of COP is to find an assignment $\boldsymbol{x} \in D := D_1 \times D_2 \times \cdots \times D_n$ that minimizes the *total cost* $f(\boldsymbol{x}) := \sum_{(i,j)\in E} f_{i,j}(x_i, x_j)$. Let $d_i = |D_i|$ for each $i \in [n]$. We define $d_{\max} = \max_{i\in[n]} d_i$ and $d = |D| = d_1 d_2 \cdots d_n$.

For each $i$, let us define $f_i : D \to \mathbb{R}^+$ as $f_i(\boldsymbol{x}) = \sum_{j\in[n]:(i,j)\in E} f_{i,j}(x_i, x_j)$, which is the sum of the cost functions involving the variable $x_i$. Note that $f(\boldsymbol{x}) = \frac{1}{2}\sum_i f_i(\boldsymbol{x})$ holds. In this paper, we assume that $f_i(\boldsymbol{x}) \in [0, 1]$ for every $i \in [n]$ and $\boldsymbol{x} \in D$. Otherwise, we can normalize cost functions by dividing the maximum number of cost functions involving a variable times the maximum cost incurred by a cost function.

The *distributed constraint optimization problem* (DCOP for short) is a COP such that each variable is controlled by the unique agent associated with it. An agent can communicate with other agents through an edge: for $i \in [n]$, let $N(i) = \{j \in [n] : (i, j) \in E\}$ denote the set of (indices of) variables adjacent to $x_i$. In one round, the agent $i$, who controls the variable $x_i$, can send information to or receive information from agents in $N(i)$. The goal of DCOP is to find an assignment that minimizes the total cost.

## The Multiplicative Weights Method

Consider the following setting. We have a set $D$ of $d$ decisions, and we are required to select one decision from the set in each round. More specifically, in round $t$, we select a vector $\boldsymbol{p}^t = (p_1^t, \ldots, p_d^t)$ with $\sum_{a\in D} p_a^t = 1$. Let $\mathcal{P}^t$ be the probability distribution corresponding to $\boldsymbol{p}^t$. That is, $\mathcal{P}^t(a) = \boldsymbol{p}_a^t$ for each $a \in D$. Then we sample a decision $a$ from $\mathcal{P}^t$. Each decision incurs a certain cost, determined by nature or an adversary. After making our decision, all the costs are revealed in the form of the vector $\boldsymbol{c}^t = (c_1^t, \ldots, c_d^t)$. The expected cost to the algorithm using the vector $\boldsymbol{p}^t$ is $\mathbf{E}_{a\sim\mathcal{P}^t}[\boldsymbol{c}_a^t] = \langle \boldsymbol{p}^t, \boldsymbol{c}^t\rangle$. Hence after $T$ rounds, the total expected cost is $\sum_{t=1}^T \langle \boldsymbol{p}^t, \boldsymbol{c}^t\rangle$.

We wish to have an algorithm that achieves a total expected cost not too much more than the cost of the best single decision in hindsight, that is, $\min_{a\in D}\sum_{t=1}^T c_a^t$. Algorithm 1, called the *multiplicative weights method* (the MW method for short), is known to have this property. More specifically, we have the following.

**Theorem 1 ((Arora, Hazan, and Kale 2012)).** *Assume that all costs $c_a^t \in [-1, 1]$. By choosing $T = O(\frac{\log |D|}{\epsilon^2})$ and $\eta = \sqrt{\frac{\log d}{T}}$, the MW method guarantees that, after $T$ rounds, for any decision $a$, we have*

$$\frac{1}{T}\left(\sum_{t=1}^{T}\langle \boldsymbol{c}^t, \boldsymbol{p}^t\rangle - \sum_{t=1}^{T} \boldsymbol{c}_a^t\right) \leq \epsilon.$$

The left hand side is called the *regret* of the method. If the limit of the regret as $T \to \infty$ is at most zero, then the method is called a *no-regret method*. The MW method is an example of a no-regret method.

The idea of our methods for DCOP is that each agent individually performs the multiplicative weights method to guess the best decision, that is, the best value in its domain. Although the cost $c_i^t$ for an agent should reflect the loss caused by choosing the value $i$, we have many choices for how to define $c_i^t$. In the following two sections, we propose two methods, an LP-based method and a game-based method.

# DMW-LP: An LP-Based Method

In this section, we explain our method based on LP relaxations, called DMW-LP.

## LP Formulation

We now show our LP relaxation for the given COP instance. For each variable $x_i \in X$, we introduce LP variables $p_{i,1}, \ldots, p_{i,d_i}$ with the constraint $\sum_{a\in D_i} p_{i,a} = 1$. Here, $p_{i,a}$ is supposed to indicate the probability that the variable $x_i$ takes the value $a$. Hence, we can regard the set of LP variables $\boldsymbol{p}_i := \{p_{i,a}\}_{a\in D_i}$ as a probability distribution $\mathcal{P}_i$ over the domain $D_i$. For each cost function $f_{i,j} \in F$ over variables $x_i$ and $x_j$, we introduce LP variables $\{\mu_{i,j,a,b}\}_{a\in D_i, b\in D_j}$ with the constraint $\sum_{a\in D_i} \mu_{i,j,a,b} = p_{j,b}$ for every $b \in D_j$, and $\sum_{b\in D_j} \mu_{i,j,a,b} = p_{i,a}$ for every $a \in D_i$. Here, $\mu_{i,j,a,b}$ is supposed to indicate the probability that $x_i$ and $x_j$ take the values $a$ and $b$, respectively. Hence, we can regard the set of LP variables $\boldsymbol{\mu}_{i,j} := \{\mu_{i,j,a,b}\}_{a\in D_i, b\in D_j}$ as a probability distribution $\mathcal{P}_{i,j}$ over the domain $D_i \times D_j$. The marginal distributions of $\mathcal{P}_{i,j}$ on $x_i$ and $x_j$ should be equal to the distributions $\mathcal{P}_i$ and $\mathcal{P}_j$, respectively. Then, we minimize the sum of cost functions $f_{i,j}(a, b)$, where $(a, b)$ is sampled from the distribution associated with $\boldsymbol{\mu}_{i,j}$.

Formally, our LP formulation is expressed as follows:

$$
\begin{aligned}
\min & \sum_{f_{i,j} \in F} \mu_{i,j,a,b} f_{i,j}(a,b), \\
\text{subject to} & \sum_{a \in D_i} p_{i,a} = 1 \quad \forall x_i \in X, \\
& \sum_{a \in D_i} \mu_{i,j,a,b} = p_{j,b} \quad \forall f_{i,j} \in F, b \in D_j, \\
& \sum_{b \in D_j} \mu_{i,j,a,b} = p_{i,a} \quad \forall f_{i,j} \in F, a \in D_i, \\
& p_{i,a} \geq 0 \quad \forall x_i \in X, a \in D_i, \\
& \mu_{i,j,a,b} \geq 0 \quad \forall f_{i,j} \in F, a \in D_i, b \in D_j.
\end{aligned} \tag{1}
$$

This LP is called the basic LP in the theory community (Kun et al. 2012; Thapper and Živný 2012).

We note that once we have determined the values of $\boldsymbol{p} := \{p_{i,a}\}_{i \in [n], a \in D_i}$, we can locally optimize the values of $\boldsymbol{\mu} := \{\mu_{i,j,a,b}\}_{i,j \in [n], a \in D_i, b \in D_j}$. Indeed, $\boldsymbol{\mu}_{i,j}$ can be optimized just by looking at $\boldsymbol{p}_i$, $\boldsymbol{p}_j$, and $f_{i,j}$. We also note that the domain of $\boldsymbol{x}$ is the convex set $\triangle := \triangle_{d_1} \times \triangle_{d_2} \times \cdots \times \triangle_{d_n}$, and hence its dimensions can be indexed by a pair $(i,a)$, where $i \in [n]$ and $a \in D_i$.

We extend the domain of $f$ from $D$ to $\triangle$ as follows. We define $f(\boldsymbol{p}) = \sum_{(i,j) \in E} \sum_{a \in D_i, b \in D_j} \mu_{i,j,a,b} f_{i,j}(a,b)$, where $\boldsymbol{\mu}$ is locally optimized by using $\boldsymbol{p}$ as above. When all values of $\boldsymbol{p}$ are restricted to be integral, $f(\boldsymbol{p})$ coincides with the original cost function. The function $f(\cdot)$ is convex and continuous because it is determined by optimizing an LP.

## Computing Subgradients

We want to use the MW method by setting $c^t_{i,a}$ to the $(i,a)$-th coordinate of a (sub)gradient of $f$ at the current LP solution $\boldsymbol{p}^t$ (note that domain $\triangle$ is indexed by a pair $(i,a)$). The first issue we need to overcome is how to compute the (sub)gradient locally. Before going into the detail, we need to introduce definitions related to subgradients.

Let $f : D \to \mathbb{R}$ be a convex function, where $D$ is a convex open set. For a vector $\boldsymbol{x} \in D$, a vector $\boldsymbol{v}$ is called a *subgradient* of $f$ at $\boldsymbol{x}$ if for any vector $\boldsymbol{y} \in D$, we have

$$f(\boldsymbol{y}) - f(\boldsymbol{x}) \geq \langle \boldsymbol{y} - \boldsymbol{x}, \boldsymbol{v} \rangle.$$

The set of all subgradients of $f$ at $\boldsymbol{x}$ is called the *subdifferential* at $\boldsymbol{x}$ and is denoted by $\partial f(\boldsymbol{x})$. It is known that, if $f$ is continuous, then the subdifferential at any vector in $D$ is non-empty. Furthermore, if $f$ is differentiable, then the subdifferential at $\boldsymbol{x}$ consists of a unique element, namely, the gradient of $f$ at $\boldsymbol{x}$. Subdifferentials admit *additivity*, that is, for two convex functions $f, g : D \to \mathbb{R}$ and $\boldsymbol{x} \in D$, we have

$$\partial(f+g)(\boldsymbol{x}) = \partial f(\boldsymbol{x}) + \partial g(\boldsymbol{x}),$$

where, on the right hand side, the addition of sets of vectors $X$ and $Y$ is defined as $X + Y = \{\boldsymbol{x} + \boldsymbol{y} \mid \boldsymbol{x} \in X, \boldsymbol{y} \in Y\}$.

Let $f : \Delta \to \mathbb{R}$ be the objective function given by LP (1). Since $f$ is convex and continuous, the subdifferential at any $\boldsymbol{x} \in \blacktriangle$ is non-empty, where $\blacktriangle$ is the interior of $\triangle$.

In our method, the agent $i$ is in charge of the set of LP variables $\boldsymbol{x}_i$, and it computes a subgradient $\boldsymbol{v}_i$ of $f_i$ at $\boldsymbol{x}$. This is possible since $f_i$ only depends on $\boldsymbol{x}_i$ and $\{\boldsymbol{x}_j\}_{j \in N(i)}$. More specifically, the agent $i$ does the following: since the function $f_i$ is determined by an LP, we can assume that there is a subgradient of $f_i$ in the direction $\left(\boldsymbol{e}_{i,a_+} - \boldsymbol{e}_{i,a_-}\right)$ for some $a_+, a_- \in D_i$, where $\boldsymbol{e}_{i,a}$ is the unit vector corresponding to the dimension $(i,a)$. Hence, we can compute a subgradient as shown in Algorithm 2, which calculates the slopes along all directions of the form $\boldsymbol{e}_{i,a} - \boldsymbol{e}_{i,a'}$ for $a, a' \in D_i$ and takes the maximum of them. When calculating the slope of $f_{i,j}$ in the direction $\boldsymbol{e}_{i,a_+} - \boldsymbol{e}_{i,a_-}$, for each $b \in D_j$, we consider the slope obtained by increasing $\mu_{i,j,a_+,b}$ and decreasing $\mu_{i,j,a_-,b}$, that is, $f_{i,j}(a_+,b) - f_{i,j}(a_-,b)$.

**Algorithm 2** Computing subgradients

**Input:** An agent $i$ and $\{\boldsymbol{\mu}_{i,j}\}_{j \in N(i), a \in D_i, b \in D_j}$.
**Output:** The subgradient of $f_i$.
  **for** $a_+ \in D_i$ **do**
    **for** $a_- \in D_i$ **do**
      $\text{slope}_{a_+,a_-} \leftarrow 0$
      **for** $j \in N(i)$ **do**
        **for** $b \in D_j$ **do**
          **if** $\mu_{i,j,a_+,b} = 1$ **then continue**
          **if** $\mu_{i,j,a_-,b} = 0$ **then continue**
          $\text{slope}_{j,b} \leftarrow f_{i,j}(a_+,b) - f_{i,j}(a_-,b)$.
        $\text{slope}_j \leftarrow \max_{b \in D_j} \text{slope}_{j,b}$.
        $\text{slope}_{a_+,a_-} \leftarrow \text{slope}_{a_+,a_-} + \text{slope}_j$.
  $(a_+^*, a_-^*) \leftarrow \arg\max_{a_+,a_- \in D_i} \text{slope}_{a_+,a_-}$.
  **return** $\frac{1}{\sqrt{2}}(\mathbf{e}_{i,a_+^*} - \mathbf{e}_{i,a_-^*}) \cdot \text{slope}_{a_+^*,a_-^*}$.

We note that the sum $\boldsymbol{v} = \sum_{i \in [n]} \boldsymbol{v}_i$ is actually a subgradient of the cost function $f$. This is because $f = \frac{1}{2}\sum_{i \in [n]} f_i$ and subdifferentials admit additivity.

## DMW-LP

DMW-LP basically follows the MW method. The difference is that it is performed in a distributed manner and the domain is $\triangle$ instead of a single simplex. In DMW-LP, each agent $i$ maintains a weight vector $\boldsymbol{w}_i = (\boldsymbol{w}_{i,1}, \ldots, \boldsymbol{w}_{i,d_i})$. In each round $t$, it provides a vector $\boldsymbol{p}^t_i = \boldsymbol{w}^t_i / \|\boldsymbol{w}^t_i\|_1$. The cost vector given here is the $i$-th part of the subgradient, which can be computed as in the previous section. The update rule is the same as the original MW method. Algorithm 3 summarizes our method. We note that we always have $\boldsymbol{p}^t \in \blacktriangle$ and hence a subgradient at $\boldsymbol{p}^t$ exists.

At the end of the algorithm, we round the vector $\boldsymbol{p}' = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{p}^t$ to an assignment. We consider the following two strategies.

- Majority strategy: we simply set $x_i = \arg\max_{a \in D_i} \boldsymbol{p}'_{i,a}$ for each $i \in [n]$. Note that this rounding method does not involve any communication.
- DSA strategy: In order to exploit joint distributions $\boldsymbol{\mu}_{i,j}$, we consider the following rounding strategy using the idea of DSA (Fitzpatrick and Meertens 2003). We repeat the following process for a constant number of steps (100 steps in our experiments). At each step, we fix each variable with a certain probability (30% in our experi-

**Algorithm 3** DMW-LP (with the majority strategy)

**Input:** A DCOP instance $\phi$ and a parameter $T$
**Output:** An assignment $x_1, ..., x_n$.

Set $\eta \leftarrow \sqrt{\frac{\log d}{nT}}$
**for** each agent $i$ **do**
  Set $\boldsymbol{w}_i^1 \leftarrow (1, 1, \dots, 1)$ and $\boldsymbol{p}_i^1 \leftarrow (\frac{1}{d_i}, \frac{1}{d_i}, \dots, \frac{1}{d_i})$.
  **send** $\boldsymbol{p}_i^1$ to each agent $j \in N(i)$.
**for** $t = 1$ to $T$ **do**
  **for** each agent $i$ **do**
    **receive** $\boldsymbol{p}_j^t$ from each agent $j \in N(i)$.
    Compute a subgradient $\boldsymbol{v}_i^t$ of $f_i$ at $\boldsymbol{p}^t$.
    $w_{i,a}^{t+1} \leftarrow w_{i,a}^t(1 - \eta v_{i,a}^t)$ for each $a \in D_i$.
    $p_{i,a}^{t+1} \leftarrow \frac{w_{i,a}^{t+1}}{\|\boldsymbol{w}_i^t\|_1}$ for each $a \in D_i$.
    **send** $\boldsymbol{p}_i^{t+1}$ to each agent $j \in N(i)$.
**for** each agent $i$ **do**
  Assign $x_i$ the value $\arg\max_{a \in D_i} \boldsymbol{p}'_{i,a}$, where $\boldsymbol{p}'_i = \frac{1}{T}\sum_{t=1}^T \boldsymbol{p}^t$.

ments). Suppose that we are going to fix a variable $x_i$. Let $\boldsymbol{\mu}^t$ be determined by $\boldsymbol{p}^t$ and $\boldsymbol{\mu}' = \frac{1}{T}\sum_{t=1}^T \boldsymbol{\mu}^t$. Let $\mathrm{FN}(i) = \{j \mid f_{i,j} \in F \text{ and } x_j \text{ is already fixed }\}$ be the set of neighbors of $i$ whose values have been fixed. For $j \in \mathrm{FN}(i)$, let $b_j$ be the value of $x_j$. Then, the probability that we set $x_i = a$ should be

$$\Pr[x_i = a \mid \bigwedge_{j \in \mathrm{FN}(i)} x_j = b_j] = \frac{\Pr[x_i = a \wedge \bigwedge_{j \in \mathrm{FN}(i)} x_j = b_j]}{\Pr[\bigwedge_{j \in \mathrm{FN}(i)} x_j = b_j]}$$

Though we cannot compute this probability from $\boldsymbol{\mu}'$ and $\boldsymbol{p}'$, in order to obtain a rounding method, we assume that $\{\boldsymbol{p}'_j\}_{j \in \mathrm{FN}(i)}$ are independent. Then, the probability above is equal to

$$\Pr[x_i = a] \prod_{j \in \mathrm{FN}(i)} \frac{\Pr[x_j = b_j \mid x_i = a]}{\Pr[x_j = b]} = \Pr[x_i = a] \prod_{j \in \mathrm{FN}(i)} \frac{\Pr[x_i = a \wedge x_j = b_j]}{\Pr[x_i = a]\Pr[x_j = b]} = p'_{i,a} \prod_{j \in \mathrm{FN}(i)} \frac{\mu'_{i,j,a,b}}{p'_{i,a} p'_{j,b}}.$$

With this probability we assign $a$ to $i$.

## A Proof of Convergence

Now we prove that Algorithm 3 converges to an optimal LP solution. Let $\boldsymbol{x}^*$ be the optimal LP solution. Then, we have the following.

**Theorem 2.** *Algorithm 3 achieves the following guarantee for all $T \geq 1$,*

$$\frac{1}{T}\Big(\sum_{t=1}^T f(\boldsymbol{p}^t) - \sum_{t=1}^T f(\boldsymbol{p}^*)\Big) = O\Big(\sqrt{\frac{n \log d}{T}}\Big).$$

Due to space limitations, we omit the proof for Theorem 2. Note that the only difference from Theorem 1 is that the domain is now $\triangle$ instead of a single simplex, and the modification to the proof is complicated but not too hard.

**Corollary 3.** *The vector $\boldsymbol{p}'$ is a feasible LP solution and $f(\boldsymbol{p}') - f(\boldsymbol{p}^*) = O(\sqrt{\frac{n \log d}{T}})$.*

*Proof.* Note that $\boldsymbol{p}^1, \dots, \boldsymbol{p}^T$ are feasible LP solutions, and hence their convex combination $\boldsymbol{p}'$ is also a feasible LP solution. The second claim is immediate from Theorem 2 and the convexity of $f(\cdot)$. □

**Corollary 4.** *By choosing $T = O(\frac{\log d_{\max}}{\epsilon^2})$, we have $f(\boldsymbol{p}') - f(\boldsymbol{p}^*) = O(\epsilon n)$.*

*Proof.* Since $\log d \leq n \log d_{\max}$, we have the desired result from Corollary 3. □

# DMW-Game: A Game-Based Method

In this section, we explain our second method, called DMW-Game. In this method, the agents play a cost-minimization game, and find a coarse correlated equilibrium using the MW method. The details are given below.

## Cost-Minimization Games

We introduce several notions from game theory. A *cost-minimization game* has the following components:

- a finite number of players denoted by $n$;
- a finite decision set $D_i$ for each player $i$
- a cost function $f_i : D \to [0, 1]$ for each player $i$, where $D = D_1 \times \cdots \times D_n$.

We consider the following way of playing cost-minimization games, called *no-regret dynamics*. In each round $t = 1, 2, ..., T$, we do the following.

- Each player $i$ simultaneously and independently chooses a distribution $\mathcal{P}_i^t$ over $D_i$ using a no-regret method.
- Each player $i$ receives a cost vector $(\boldsymbol{c}_{i,1}^t, \dots, \boldsymbol{c}_{i,1}^t)$, where $\boldsymbol{c}_{i,a}^t$ is the expected cost of the decision $a$ when the other players play according to their chosen distributions. That is, $\boldsymbol{c}_{i,a}^t = \mathbf{E}_{x \sim \mathcal{P}^t}[f_i(x_1, \dots, x_{i-1}, a, x_{i+1}, \dots, x_n)]$, where $\mathcal{P}^t = \prod_{i \in [n]} \mathcal{P}_i^t$.

The no-regret dynamics converges to an equilibrium in the following sense.

**Theorem 5 (Folklore).** *Suppose after $T$ rounds of no-regret dynamics, every player of a cost-minimization game has a regret of at most $\epsilon$ for each of its decisions. Let $\mathcal{P}^t = \prod_{i=1}^n \mathcal{P}_i^t$ denote the distribution at time $t$ and $\mathcal{P} = \frac{1}{T}\sum \mathcal{P}^t$ denote the time-averaged history of these distributions. Then $\mathcal{P}$ is an $\epsilon$-approximate coarse correlated equilibrium, in the sense that*

$$\mathop{\mathbf{E}}_{x \sim \mathcal{P}}[f_i(x)] \leq \mathop{\mathbf{E}}_{x \sim \mathcal{P}}[f_i(x_1, \dots, x_{i-1}, a_i, x_{i+1}, \dots, x_n)] + \epsilon.$$

*for every player $i$ and unilateral deviation $a_i$.*

A coarse correlated equilibrium protects against unilateral deviations. In contrast, a Nash equilibrium even prevents any agent from using another distribution in place of the current distribution to make the expected cost smaller. In this sense, any Nash equilibrium is a coarse correlated equilibrium. Though Nash equilibriums always exist (John F. Nash 1950), it is open to debate whether we can obtain any of them even approximately in polynomial time in $n$.

We want to assign a single decision to each player in the setting of DCOP, and we just want to guarantee that the current probability distribution of decisions of each player is not worse than any single decision of the player. In this way, we could justify using coarse correlated equilibriums instead of Nash equilibriums.

## DMW-Game

DCOP can be seen as a cost-minimization game by observing that $D_i$ is the domain of the $i$-th variable and the cost function $f_i : D_i \to [0, 1]$ is the cost function involving the $i$-th variable $x_i$.

Using the MW method as the no-regret method in the no-regret dynamics, we obtain Algorithm 4. Combining Theorems 1 and 5, we obtain the following.

**Theorem 6.** *By choosing $T = O(\frac{\log d_{\max}}{\epsilon^2})$, the vector $\boldsymbol{p}'$ is an $\epsilon$-approximate coarse correlated equilibrium.*

At the end of the algorithm, we need to round the vector $\boldsymbol{p}'$ to an assignment. We consider the following two strategies.

- Majority strategy: This is exactly the same as the majority strategy for DMW-LP.
- Restart strategy: we empirically found that, in DMW-Game, almost all variables are quickly fixed, that is, $\max_a \boldsymbol{p}_{i,a}$ becomes close to one for almost all $i \in V$. To restart DMW-Game again with this configuration, for every certain number of steps (100 steps in our experiments), the restart strategy resets variables $i$ (set $\boldsymbol{p}_{i,a} = \frac{1}{d_i}$ for all $a \in D_i$) if $\max_a \boldsymbol{p}_{i,a}$ is smaller than a threshold (0.99 in our experiments), and does not touch (almost) fixed variables. At the end of the process, we use the majority strategy to obtain an assignment.

**Algorithm 4** DMW-Game (with the majority strategy)

**Input:** A DCOP instance $\phi$ and a parameter $T$
**Output:** An assignment $x_1, ..., x_n$.

Set $\eta = \sqrt{\frac{\log d_{\max}}{T}}$.
**for** each agent $i$ **do**
  Set $\boldsymbol{w}_i^1 \leftarrow (1, 1, \ldots, 1)$ and $\boldsymbol{p}_i^1 \leftarrow (\frac{1}{d_i}, \frac{1}{d_i}, \ldots, \frac{1}{d_i})$.
  **send** $\boldsymbol{p}_i^1$ to each agent $j \in N(i)$.
**for** $t = 1$ to $T$ **do**
  **for** each agent $i$ **do**
    **receive** $\boldsymbol{p}_j^t$ from each agent $j \in N(i)$.
    $w_{i,a}^{t+1} \leftarrow w_{i,a}^t(1 - \eta f_i(a))$ for each $a \in D_i$.
    $p_{i,a}^{t+1} \leftarrow \frac{w_{i,a}^{t+1}}{\|\boldsymbol{w}_i^{t+1}\|_1}$ for each $a \in D_i$.
    **send** $\boldsymbol{p}_i^{t+1}$ to each agent $j \in N(i)$.
**for** each agent $i$ **do**
  Assign $x_i$ the value $\arg\max_{a \in D_i} p'_{i,a}$, where $\boldsymbol{p}'_i = \frac{1}{T}\sum_{i=1}^{T} \boldsymbol{p}^t$.

# Experiments

In this section, we experimentally confirm the solution quality and scalability of DMW, and the lower bound quality of DMW-LP. In this section, LP+Maj and LP+DSA mean DMW-LP with the majority strategy and the DSA strategy, respectively, and Game+Maj and Game+Res mean DMW-Game with the majority strategy and the restart strategy, respectively. We compare DMW with previous incomplete DCOP algorithms, MaxSum (Farinelli et al. 2008), DeQED (Hatano and Hirayama 2013), DSA (Zhang et al. 2005), and MGM (Maheswaran, Pearce, and Tambe 2004).

We conducted experiments on an Ubuntu server with Intel Core-i7 3770@3.4GHz and 4GB of memory. DMW and DeQED were written in Java. For DMW-LP and DMW-Game, we set $\eta$ to be 0.04 and 0.5, respectively. For MaxSum, DSA and MGM, we used the code in FRODO version 2.11 (Léauté, Ottens, and Szymanek 2009) with the default setting.

## DCOP Instances

We made three kinds of DCOP instances, random binary constraint networks, scale-free binary constraint networks, and meeting scheduling problems as real-world problems. For the first two kinds of instances, we created the underlying networks as follows.

**Random** We created an $n$-vertex network whose density is $\delta$, resulting in $\lfloor \delta \binom{n}{2} \rfloor$ edges.

**Scale-free** We created an $n$-vertex network using the Barabasi-Albert (BA) model (Barabási and Albert 1999), where each newly added vertex is connected to the two existing vertices, resulting in $2(n-2)+1$ edges.

We made sure of the connectivity of each network. Then, we created 20 COP instances for each topology in such a way that the domain size of each variable (nodes) is three, and costs of each cost function (edges) are randomly selected from $\{1, 2, ..., 10^5\}$.

For the third kind of instance, we made 20 instances of the meeting scheduling problem, which are created by the instance generator of FRODO version 2.11 (Léauté, Ottens, and Szymanek 2009) under the parameter of "-PEAV -infinity $10^5$ -maxCost $10^2$ 40 25 4 4".

## Solution Quality and Running Time

We first show that our methods efficiently output high-quality solutions compared to existing methods. To see this, we consider the following three measures: the *solution quality*, which is the cost of the output divided by the best lower bound given by DeQED$_a$ and DeQED$_m$, the *number of cycles*, and the *simulated runtime*, which is the sum of simulated runtime of all cycles, where the *simulated runtime of a cycle* is the longest running time of an agent in the cycle.

The results are shown in Table 1. On binary constraint networks, DMW-Game (especially with the fixing strategy)

Table 1: Average solution quality and simulated runtime (in msec) for (a) 100-node random networks with density 0.1, (b) 100-node scale-free networks, and (c) 100-node meeting scheduling problems.

| Prob. | Cyc. | LP+Maj sol. | LP+Maj time | LP+DSA sol. | LP+DSA time | Game+Maj sol. | Game+Maj time | Game+Res sol. | Game+Res time | MaxSum sol. | MaxSum time | DeQED$_a$ sol. | DeQED$_a$ time | DeQED$_m$ sol. | DeQED$_m$ time | DSA sol. | DSA time | MGM sol. | MGM time |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (a) | 100 | 1.339 | 12.4 | 1.304 | 13.5 | 1.199 | 0.5 | **1.184** | **0.3** | 1.331 | 208.1 | 1.233 | 4.6 | 1.409 | 0.5 | 1.196 | 22.5 | 1.208 | 26.2 |
| | 200 | 1.330 | 24.5 | 1.301 | 27.4 | 1.188 | 1.2 | **1.182** | **0.5** | 1.331 | 272.3 | 1.217 | 9.3 | 1.363 | 1.0 | 1.198 | 29.9 | 1.197 | 33.7 |
| | 300 | 1.326 | 37.9 | 1.305 | 41.8 | 1.185 | 1.6 | **1.182** | **0.7** | 1.327 | 368.8 | 1.214 | 14.0 | 1.340 | 1.4 | 1.207 | 43.0 | 1.207 | 43.0 |
| | 400 | 1.328 | 49.8 | 1.305 | 56.2 | 1.183 | 1.9 | **1.181** | **0.9** | 1.317 | 454.8 | 1.214 | 18.7 | 1.327 | 1.9 | 1.203 | 42.7 | 1.201 | 51.5 |
| | 500 | 1.325 | 62.3 | 1.311 | 70.4 | 1.183 | 2.3 | **1.181** | **1.1** | 1.329 | 588.6 | 1.214 | 23.4 | 1.322 | 2.4 | 1.198 | 46.3 | 1.202 | 63.6 |
| (b) | 100 | 1.319 | 12.1 | 1.209 | 16.8 | 1.113 | 0.4 | **1.094** | **0.3** | 1.266 | 120.9 | 1.135 | 8.7 | 1.189 | 0.6 | 1.158 | 26.6 | 1.153 | 38.7 |
| | 200 | 1.286 | 24.4 | 1.185 | 32.4 | 1.099 | 0.8 | **1.090** | **0.5** | 1.296 | 213.4 | 1.128 | 17.7 | 1.156 | 1.2 | 1.156 | 33.7 | 1.153 | 51.3 |
| | 300 | 1.257 | 38.2 | 1.168 | 48.5 | 1.097 | 1.2 | **1.089** | **0.8** | 1.248 | 268.3 | 1.126 | 26.4 | 1.133 | 1.8 | 1.168 | 42.0 | 1.161 | 62.2 |
| | 400 | 1.251 | 49.8 | 1.167 | 64.6 | 1.095 | 1.6 | **1.089** | **1.1** | 1.217 | 309.7 | 1.126 | 35.3 | 1.138 | 2.4 | 1.157 | 48.7 | 1.153 | 76.3 |
| | 500 | 1.234 | 61.4 | 1.149 | 80.3 | 1.092 | 2.0 | **1.089** | **1.3** | 1.220 | 396.5 | 1.126 | 44.1 | 1.131 | 3.0 | 1.149 | 58.1 | 1.142 | 86.6 |
| (c) | 100 | 2001 | 12.7 | 1983 | 12.9 | 970 | 0.4 | **914** | **0.2** | 3081 | 138.1 | 2402 | 2.2 | 5278 | 0.7 | 1011 | 43.1 | 1135 | 54.7 |
| | 200 | 1706 | 23.4 | 1838 | 24.5 | 925 | 0.8 | **914** | **0.4** | 2829 | 288.5 | 1591 | 5.1 | 1970 | 1.4 | 1143 | 58.8 | 1065 | 81.0 |
| | 300 | 1589 | 33.3 | 1793 | 36.0 | 925 | 1.3 | **914** | **0.6** | 2520 | 364.3 | 1499 | 7.7 | 1652 | 2.1 | 1107 | 74.7 | 1099 | 105.3 |
| | 400 | 1614 | 43.8 | 1702 | 48.2 | 925 | 1.7 | **914** | **0.8** | 2648 | 450.7 | 1469 | 10.3 | 1583 | 2.8 | 996 | 89.8 | 1096 | 128.0 |
| | 500 | 1678 | 54.2 | 1810 | 60.4 | 940 | 2.1 | **914** | **0.9** | 2516 | 551.2 | 1469 | 13.4 | 1548 | 3.6 | 1094 | 106.0 | 1144 | 152.5 |

Table 2: Average solution quality and simulated runtime (in msec) when changing the domain size $d$, the density $\delta$, and the number of variables $n$. MLE means that the memory limit is exceeded.

| | LP+Maj | | LP+DSA | | Game+Maj | | Game+Res | | MaxSum | | DeQED$_a$ | | DeQED$_m$ | | DSA | | MGM | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $d$ | sol. | time | sol. | time | sol. | time | sol. | time | sol. | time | sol. | time | sol. | time | sol. | time | sol. | time |
| 5 | 1.900 | 187.2 | 1.900 | 197.1 | 1.497 | 2.9 | **1.488** | **2.0** | 1.901 | 815.7 | 1.646 | 25.9 | 1.958 | 3.5 | 1.541 | 59.5 | 1.554 | 75.6 |
| 10 | 3.742 | 1085.7 | 3.739 | 1063.0 | 2.418 | 6.6 | **2.381** | **4.6** | 3.752 | 995.0 | 3.162 | 28.6 | 3.747 | 5.9 | 2.464 | 72.9 | 2.500 | 92.6 |
| 15 | 5.884 | 3089.2 | 5.880 | 2936.3 | 3.415 | 10.2 | **3.338** | **8.8** | 5.683 | 1343.7 | 5.103 | 30.9 | 5.842 | 9.3 | 3.480 | 99.4 | 3.500 | 115.8 |
| 20 | 8.451 | 6310.7 | 8.440 | 6432.8 | 4.600 | 15.6 | **4.463** | 14.7 | 8.040 | 3857.1 | 7.651 | 33.8 | 8.366 | **12.6** | 4.642 | 124.7 | 4.753 | 133.6 |
| $\delta$ | sol. | time | sol. | time | sol. | time | sol. | time | sol. | time | sol. | time | sol. | time | sol. | time | sol. | time |
| 0.2 | 1.477 | 127.2 | 1.477 | 116.3 | 1.282 | 2.8 | **1.278** | **1.8** | 1.411 | 1267.9 | 1.325 | 38.9 | 1.589 | 3.6 | 1.289 | 69.0 | 1.295 | 219.4 |
| 0.4 | 1.549 | 256.0 | 1.549 | 208.4 | 1.369 | 4.4 | **1.365** | **3.0** | MLE | | 1.432 | 62.0 | 1.602 | 5.6 | 1.372 | 115.3 | 1.378 | 360.4 |
| 0.6 | 1.574 | 343.7 | 1.576 | 312.7 | 1.419 | 6.2 | **1.412** | **4.4** | MLE | | 1.487 | 89.7 | 1.605 | 7.8 | 1.415 | 141.2 | 1.422 | 337.4 |
| 0.8 | 1.582 | 498.0 | 1.583 | 388.7 | 1.440 | 8.1 | **1.435** | **6.0** | MLE | | 1.514 | 113.4 | 1.608 | 9.4 | 1.439 | 189.1 | 1.440 | 474.3 |
| 1.0 | 1.588 | 481.6 | 1.589 | 478.1 | 1.456 | 10.0 | **1.450** | **7.1** | MLE | | 1.541 | 140.2 | 1.604 | 11.0 | 1.455 | 465.1 | 1.455 | 655.1 |
| $n$ | sol. | time | sol. | time | sol. | time | sol. | time | sol. | time | sol. | time | sol. | time | sol. | time | sol. | time |
| 1000 | 1.327 | 260.6 | 1.307 | 303.9 | 1.182 | 2.7 | **1.180** | **2.8** | MLE | | 1.242 | 209.9 | 1.338 | 5.6 | 1.203 | 297.7 | 1.209 | 152.1 |
| 2000 | 1.319 | 566.3 | 1.298 | 596.7 | 1.182 | 5.5 | **1.181** | **5.6** | MLE | | 1.238 | 359243.3 | 1.332 | 7.8 | 1.201 | 518.8 | 1.206 | 360.0 |
| 5000 | 1.323 | 2074.4 | 1.303 | 1933.7 | 1.183 | 13.4 | **1.181** | 12.8 | MLE | | MLE | | 1.335 | **10.2** | 1.203 | 1087.4 | MLE | |
| 10000 | 1.324 | 5516.7 | 1.303 | 5440.9 | 1.182 | 18.3 | **1.181** | 18.7 | MLE | | MLE | | 1.335 | **12.0** | 1.202 | 2372.2 | MLE | |

outperforms other algorithms in terms of solution quality. On meeting scheduling problems, however, DMW families are only competitive against DeQED$_m$. The reason is that the cost functions in these problems act as hard constraints, and DMW may fail to satisfy them when rounding.

We can observe that DMW-Game obtained a better solution quality than DMW-LP for every kind of DCOP instance, which empirically shows that the convergent solution of DMW-Game is almost an integer solution. DMW-Game is more efficient than DMW-LP since the agents in DMW-Game only need a simple calculation whereas those in DMW-LP need to solve LPs. With the restart strategy, DMW-Game outputs a better solution and runs even faster.

## Scalability

Next, we evaluate the scalability of DMW by changing the domain size, the density, and the number of variables of the random binary constraint networks. When changing the number of variables, we set the density so that the average number of cost functions per agent is preserved.

The results are shown in Table 2, where the average solution quality and simulated runtime are measured in the 500th cycle. As we can observe, Game+Res achieves the best solution quality for all the settings, and is the fastest method in most cases. We were unable to run MaxSum, DeQED$_a$, and MGM on large-scale instances since the computational cost and the memory used by an agent increases along with the number of variables.

## Lower bounds

Table 3 summarizes the average lower bound qualities of DMW-LP and other methods that can compute lower bounds using the datasets as in Table 1. The lower bound is measured on the 50,000th cycle, and is enough to be converged. DMW-LP computed better lower bounds than other methods for binary constraint networks, and can only output a trivial lower bound of zero for the meeting scheduling problems.

Table 3: Average lower bound quality

| Prob. | LP+Maj | DeQED$_a$ | DeQED$_m$ |
|---|---|---|---|
| (a) | 0.835 | 0.779 | 0.780 |
| (b) | 0.901 | 0.842 | 0.849 |
| (c) | 0.000 | 0.002 | 0.002 |

## Conclusions

We proposed new incomplete methods for DCOP based on the multiplicative weights (MW) method. The first method, DMW-LP, solves an LP relaxation in a distributed manner. We proved that it outputs a solution arbitrarily close to the optimal LP solution. The second method, DMW-Game, plays a cost-minimization game, and outputs a solution arbitrarily close to a coarse correlated equilibrium.

We experimentally demonstrated the scalability of both methods using DCOP instances of several kinds of topologies and cost functions, and confirmed that DMW-Game in particular outperforms existing methods in terms of solution quality and efficiency.

## Acknowledgments

Yuichi Yoshida is supported by JSPS Grant-in-Aid for Young Scientists (B) (No. 26730009), MEXT Grant-in-Aid for Scientific Research on Innovative Areas (No. 24106003), and JST, ERATO, Kawarabayashi Large Graph Project.

## References

Arora, S.; Hazan, E.; and Kale, S. 2012. The multiplicative weights update method: A meta-algorithm and applications. *Theory of Computing* 8(1):121–164.

Barabási, A.-L., and Albert, R. 1999. Emergence of scaling in random networks. *Science* 286(5439):509–512.

Bowring, E.; Pearce, J. P.; Portway, C.; Jain, M.; and Tambe, M. 2008. On *k*-optimal distributed constraint optimization algorithms: new bounds and algorithms. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 607–614.

Farinelli, A.; Rogers, A.; Petcu, A.; and Jennings, N. R. 2008. 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 (AAMAS)*, 639–646.

Fitzpatrick, S., and Meertens, L. 2003. Distributed coordination through anarchic optimization. In *Multiagent Systems, Artificial Societies, and Simulated Organizations*. Springer. 257–295.

Hatano, D., and Hirayama, K. 2013. DeQED: An efficient divide-and-coordinate algorithm for DCOP. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, 566–572.

John F. Nash, J. 1950. Equilibrium points in n-person games. *Proceedings of the National Academy of Science* 36(1):48.

Kiekintveld, C.; Yin, Z.; Kumar, A.; and Tambe, M. 2010. Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 133–140.

Kun, G.; O'Donnell, R.; Tamaki, S.; Yoshida, Y.; and Zhou, Y. 2012. Linear programming, width-1 CSPs, and robust satisfaction. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS)*, 484–495.

Léauté, T.; Ottens, B.; and Szymanek, R. 2009. FRODO 2.0: An open-source framework for distributed constraint optimization. In *Proceedings of the IJCAI-09 Distributed Constraint Reasoning Workshop*, 160–164. http://liawww.epfl.ch/frodo/.

Maheswaran, R. T.; Pearce, J. P.; and Tambe, M. 2004. Distributed algorithms for DCOP: A graphical-game-based approach. In *Proceedings of the ISCA 17th International Conference on Parallel and Distributed Computing Systems (PDCS)*, 432–439.

Nguyen, D. T.; Yeoh, W.; and Lau, H. C. 2013. Distributed Gibbs: A memory-bounded sampling-based DCOP algorithm. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 167–174.

Ottens, B.; Dimitrakakis, C.; and Faltings, B. 2012. DUCT: An upper confidence bound approach to distributed constraint optimization problems. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI)*.

Schapire, R. E. 2003. The boosting approach to machine learning: An overview. In Denison, D. D.; Hansen, M. H.; Holmes, C.; Mallick, B.; and Yu, B., eds., *Nonlinear Estimation and Classification*. Springer.

Thapper, J., and Živný, S. 2012. The power of linear programming for valued CSPs. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 669–678.

Vinyals, M.; Rodriguez-Aguilar, J. A.; and Cerquides, J. 2010. Divide-and-coordinate by egalitarian utilities: Turning DCOPs into egalitarian worlds. In *Proceedings of the 3rd International Workshop on Optimisation in Multi-Agent Systems (OPTMAS)*.

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