# Graph Based Optimization For Multiagent Cooperation

Arambam James Singh
Singapore Management University
arambamjs.2016@smu.edu.sg

Akshat Kumar
Singapore Management University
akshatkumar@smu.edu.sg

## ABSTRACT

We address the problem of solving math programs defined over a graph where nodes represent agents and edges represent interaction among agents. The objective and constraint functions of this program model the task agent team must perform and the domain constraints. In this multiagent setting, no single agent observes the complete objective and all the constraints of the program. Thus, we develop a distributed message-passing approach to solve this optimization problem. We focus on the class of graph structured linear and quadratic programs (LPs/QPs) which can model important multiagent coordination frameworks such as distributed constraint optimization (DCOP). For DCOPs, our framework models functional constraints among agents (e.g. resource, network flow constraints) in a much more tractable fashion than previous approaches. Our iterative approach has several desirable properties—it is guaranteed to find the optimal solution for LPs, converges for general cyclic graphs, and is memory efficient making it suitable for resource limited agents, and has anytime property. Empirically, our approach provides solid empirical results on several standard benchmark problems when compared against previous approaches.

## KEYWORDS

Distributed constraint optimization;multiagent cooperation; mathematical optimization

**ACM Reference Format:**
Arambam James Singh and Akshat Kumar. 2019. Graph Based Optimization For Multiagent Cooperation. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 9 pages.

## 1 INTRODUCTION

Several frameworks have been proposed for cooperative multiagent decision making such as distributed constraint optimization (DCOP) for single-shot decisions [10, 44], and dynamic DCOPs [13] and decentralized Markov decision processes (Dec-MDPs) for sequential multiagent decision making [2]. Finding optimal decisions can often be formulated as solving a mathematical program that models the team's goal as the objective function to optimize and agent interactions using constraints. E.g., DCOPs can be modeled as a quadratic or a linear program [16, 34, 37], and Dec-MDP policies can be computed using quadratic/mixed-integer programs [15, 33]. Therefore, developing efficient approaches for solving such programs is crucial. We focus on solving a class of graph structured mathematical programs where nodes represent agents and edges encode interaction among agents. We show that such a graph structured program subsumes several DCOP variants, and crucially, allows to model succinctly several complex interdependencies among agents, such as those based on flow conservation, in a much more tractable manner than a standard DCOP formulation.

Similar to DCOPs, we assume that agents can only communicate with their immediate neighbors, where neighborhood is defined based on agents participating in common constraints and different sub-components of the objective function. Thus, a key goal of our work is to develop message-passing approaches that can solve such programs in a distributed fashion. Such distributed optimization is popular in the systems and control community [7, 8]. However, their focus is on solving *convex* programs, whereas the formulations and techniques we present can address non-convex programs also, and thus are highly general to model real world settings.

**Related work in DCOPs:** Distributed constraint optimization (DCOP) has emerged in recent years as an important framework for multiagent coordination [25, 28, 32, 44]. DCOPs have been used to model several multiagent coordination problems such as target tracking in sensor networks and meeting scheduling [24], managing smart grids and smart homes [12, 14, 36], multiagent logistics [21] and mobile sensing [42]. In DCOPs, agents control a set of variables with constraint or *utility functions* defined over subsets of variables. The task for agents is to assign values to variables to maximize the global utility using only local coordination among them.

Several complete and approximate algorithms have been proposed such as local search algorithms DSA and MGM [23, 45, 46], distributed search based ADOPT and its extensions [29, 43], dynamic programming based DPOP [32], and belief propagation based max-sum algorithm [9, 39]. In several settings, it is often desirable to optimize agents' decisions subject to *functional constraints* such as resource constraints [4, 11, 26]. Logical language based specifications combine answer set programming with DCOPs [20]. However, this approach also has complexity exponential in the tree-width of the agent interaction graph.

**Related work in distributed math program optimization:** In the graphical models community, several approaches exist to solve graph-structured linear and quadratic programs [18, 34, 35, 38], and they are also applicable to solve DCOPs which are closely related to the problem of maximum a posteriori (MAP) estimation in graphical models [16, 30]. However, the scope of LPs and QPs that can be solved over graphs is highly restricted in such previous work. A major limitation is that they highly restrict constraint types addressable within their respective QP/LP solvers. E.g., the distributed QP solver of [18, 35] can only handle probability normalization constraints. The LP solver of [38] can only handle probability normalization/marginalization constraints. This limitation is because their solvers works on the Lagrangian dual of the problem. Having different constraints change the dual, making their current solver inapplicable. Therefore, addressing different kind of functional constraints is not straightforward as constraints change the dual, making their current technique inapplicable. In [11], a

*Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada.*

distributed approach is developed to solve QPs for resource constrained DCOPs. However, this approach cannot solve linear programs or address general linear constraints other than the resource constraints.

**Our contributions:** Our main contribution is the development of a graph-based math optimization framework called GraphOpt, and an associated message-passing based solver to solve optimization problems specified in GraphOpt framework. Our iterative approach has several desirable properties. It is guaranteed to converge to the optimal solution for LPs and has anytime nature. It is memory efficient (each message is at most quadratic in the maximum domain size of any variable) , making it suitable for resource limited agents, such as embedded sensors in internet of things.

Our framework considerably expands the applicability and scalability of DCOPs by providing support for a wide range of functional dependencies among agents specified as linear constraints of the program such as network flow constraints that arise in smart grids [12] or *physical dependency* constraints that link sensors in a smart home to a physical value such as light level [36]. Functional constraints can be addressed in a tractable fashion without creating exponentially large utility/cost tables as in standard DCOPs. Unlike the ASP-DPOP approach, our message-passing approach does not has exponential complexity in the treewidth; maximum size of any message is quadratic in the maximum domain size of any variable.

Unlike previous distributed math optimization solvers, our approach can handle arbitrary linear constraints that fit within our GraphOpt framework, which brings significant expressive power to the class of problems solvable over graphs. To concretely illustrate this point, we present a new problem based on congestion-based multiagent routing that requires modeling network flow constraints and a mixed objective function having both quadratic and linear terms. Such programs are not readily solvable using previous LP/QP solvers.

## 2 DISTRIBUTED OPTIMIZATION

A DCOP is defined using the tuple $\langle \mathcal{X}, \mathcal{D}, \Theta \rangle$. The set $\mathcal{X} = \{x_1, .., x_n\}$ is a set of $n$ variables; $\mathcal{D} = \{D_1, \dots, D_n\}$ is the (finite) domain of possible values a variable can take. We assume that there is an agent associated with each variables $x_i$ which controls its assignment. The set $\Theta = \{\dots, \theta_{ij}, \dots\}$ is the set of pairwise utility or constraint functions. A constraint function between variables $x_i$ and $x_j$ is defined as $\theta_{ij} : D_i \times D_j \rightarrow \Re$. We also have unary constraints associated with each variable $x_i$ as $\theta_i : D_i \rightarrow \Re$. The DCOP framework can be represented using a *constraint network* $G = (V, E)$ as follows. There is a node $i$ for each variables $x_i$. For each constraint $\theta_{ij}$, we create an edge between the nodes $i$ and $j$ in the graph. The objective is to find the joint assignment to solve:

$$\max_{x_1,\dots,x_n} \Big[ \sum_{i \in V} \theta_i(x_i) + \sum_{(i,j) \in E} \theta_{ij}(x_i, x_j) \Big] \tag{1}$$

In DCOPs, an agent for a node $i$ is only aware of its shared constraints with neighbor agents. Thus, there is no centralized view of the whole problem, requiring local message-passing based coordination.

**Quadratic program:**

$$\max_{\mu} \sum_{i \in V} \sum_{x_i} \mu_i(x_i)\theta_i(x_i) + \sum_{(i,j) \in E} \sum_{x_i, x_j} \mu_i(x_i)\mu_j(x_j)\theta_{ij}(x_i, x_j) \tag{2}$$

$$\text{s.t.} \sum_{x_i} \mu_i(x_i) = 1 \ \forall i \in V \tag{3}$$

$$0 \le \mu \le 1 \tag{4}$$

**Linear program:**

$$\max_{\mu} \sum_{i \in V} \sum_{x_i} \mu_i(x_i)\theta_i(x_i) + \sum_{(i,j) \in E} \sum_{x_i, x_j} \mu_{ij}(x_i, x_j)\theta_{ij}(x_i, x_j) \tag{5}$$

$$\text{s.t.} \sum_{x_i} \mu_i(x_i) = 1 \qquad \forall i \in V \tag{6}$$

$$\sum_{x_j} \mu_{ij}(x_i, x_j) = \mu_i(x_i) \ \forall i \in V, \forall x_i, \forall j \in \text{Nb}_i \tag{7}$$

$$0 \le \mu \le 1 \tag{8}$$

**Table 1: Quadratic and linear programs for DCOPs**

### 2.1 LPs and QPs for DCOPs

DCOPs are closely related to the problem of MAP inference in graphical models [16, 30]. Using this connection, several optimization approaches developed in the graphical models community are applicable to DCOPs. We focus on two popular approaches: quadratic [35] and linear programs [38] for DCOPs. Table 1 shows both these programs. In the QP, we have a probability distribution $\mu_i(x_i)$ associated with each node $i$ of the graph (this distribution is optimized by the program). The constraints (3) in QP are probability normalization constraints. The objective function is quadratic due to bilinear terms ($\mu_i \cdot \mu_j$).

The LP associates distributions $\mu_{ij}$ with each edge $(i, j)$ of the graph in addition to $\mu_i$ parameters. The LP has additional constraints (7) which are equivalent to probability marginalization constraints. Let OPT denote the optimal integral solution quality of the DCOP, and let QP$^\star$, LP$^\star$ denote the optimal QP and LP objectives respectively. We have the following known relationship among them [35, 38]:

Theorem 1. $QP^\star = OPT \le LP^\star$

That is, the QP formulation is exact (but is non convex) for DCOPs, and the LP formulation provides an upper bound on the optimal DCOP quality. Notice also that the solution of QP and LP can be fractional (some $\mu_i$ can assign nonzero probability to multiple domain values of a variable). However, several approaches exist to round off such fractional values to an integral solution for both QP [35] and LP [34].

Although there are existing approaches to solve such programs, their scope is highly restricted. For example, existing QP solvers [18, 35] can only handle bilinear terms $\mu_i \cdot \mu_j$ in the objective and not quadratic terms as $\mu_i^2$ or linear terms as $\mu_{ij}$. Similarly, existing QP solvers are restricted to handling only probability normalization constraints. Similarly, existing LP solvers [38] require all terms in the objective as linear, and are only applicable when the constraints include normalization and marginalization constraints as in (3),(7). Thus, the scope of problems that can be solved using existing solvers is restricted when additional functional constraints are necessary. To provide a concrete example, we next present an application based on congestion-based multiagent routing.

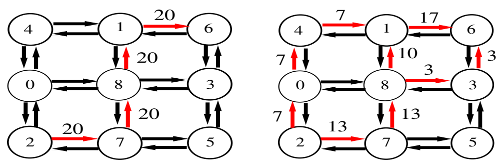

**Figure 1: 20 AGVs travel from node 2 to 6. Left figure (a) shows route ignoring congestion cost; figure (b) shows congestion aware route. Red edges with numbers denote the path taken by AGVs**

$$\min_{\boldsymbol{n}} \sum_{(i,j)} n_{ij}\cdot(\alpha_{ij}n_{ij}+\beta_{ij}) = \min_{\boldsymbol{n}} \sum_{(i,j)} \alpha_{ij}n_{ij}^2 + \sum_{(i,j)} \beta_{ij}n_{ij} \tag{9}$$

$$\sum_{j\in \mathrm{Nb}(i)} n_{ij} - \sum_{j\in \mathrm{Nb}(i)} n_{ji} = \begin{cases} N & \text{if } i \text{ is source } s \\ -N & \text{if } i \text{ is destination } d \\ 0 & \text{otherwise}\end{cases} \tag{10}$$

$$n_{ij} \text{ is integer}, n_{ij} \in [0, N] \tag{11}$$

**Table 2: Program for congestion-based routing**

$$\min_{\boldsymbol{f}} \sum_{(i,j)} \alpha_{ij}f_{ij}^2 + \sum_{(i,j)} \frac{\beta_{ij}}{N} f_{ij} \tag{12}$$

$$\sum_{j\in \mathrm{Nb}(i)} f_{ij} - \sum_{j\in \mathrm{Nb}(i)} f_{ji} = \begin{cases} 1 & \text{if } i \text{ is source } s \\ -1 & \text{if } i \text{ is destination } d \\ 0 & \text{otherwise}\end{cases} \tag{13}$$

$$0 \le f \le 1 \tag{14}$$

**Table 3: Reformulated QP for congestion-based routing**

## 2.2 Congestion-based routing

We introduce a new benchmark based on network congestion games [27]. Recently, there is an increasing interest in autonomous guided vehicles (AGVs). In settings such as logistics, multiple robots are deployed in warehouses [41]. Similarly, for automated container port, AGVs are being used for autonomously moving containers [31]. To coordinate multiple AGVs, intelligent traffic management is required. We model this problem using symmetric network congestion games where $N$ AGVs must move from a single source to a single destination in a given graph. In addition to the path cost, there is congestion cost depending on the number of AGVs crossing an edge in the graph at the same time. The goal is to find the minimum cost path for the AGV population. We treat this problem from a *system optimal* perspective for finding the *overall minimum cost* solution [27]. Meyers and Schulz [20] show that solving such congestion games (and its several variants) are NP-Hard for the general congestion cost function.

Fig. 1 shows an example of congestion based routing. We assume that each node in the graph is a *traffic agent*. Each traffic agent provides guidance regarding the next destination to all the incoming AGVs. E.g., node 8 in fig. 1(b) has total 13 incoming AGVs, 10 of them get routed to link (8, 1) and 3 to link (8, 3). We also assume that an agent can only communicate with its immediate neighbors to plan such 'congestion-aware' paths. We assume a linear congestion cost—the cost each AGV experiences include the standard edge cost $\beta_{ij}$ and the congestion cost $\alpha_{ij}n_{ij}$ if $n_{ij}$ AGVs cross the edge $(i, j)$ at the same time. Therefore, the total system cost for the complete route is $\sum_{(i,j)} \beta_{ij}n_{ij} + \sum_{ij}(\alpha_{ij}n_{ij})\cdot n_{ij}$. Table 2 shows the optimization formulation for this problem. The parameters to optimize are variables $n_{ij}$ for each directed edge $(i, j)$. These variables should also be integer, which makes the problem challenging. We can model this problem using DCOPs by creating a variable $n_{ij}$ for each edge $(i, j)$. The domain of $n_{ij}$ includes all integers from 0 to $N$, the AGV population size. We then create n-ary hard constraints between an agent $i$ and all its neighbors to model flow conservation (10). We also create a unary soft constraint for each variable $n_{ij}$ that encodes the objective term $\alpha_{ij}n_{ij}^2 + \beta_{ij}n_{ij}$. We empirically test this modeling approach with standard DCOP solvers such as max-sum and DSA. Our results confirm that this approach is highly inefficient and intractable because of several reasons. First, modeling flow constraints in dense graphs leads to exponentially large constraint tables in the degree of nodes in the graph. Second, because of flow constraints, which are hard constraints, approximate algorithms such as DSA fail to provide any reasonable solution as they violate this constraint for most test instances.

To avoid such tractability issues, we first relax the problem by assuming a continuous range $[0, N]$ for variables $n_{ij}$. We then divide the objective function in (9) by a constant $N^2$ or the AGV population size to get a QP in table 3. We used the substitution $f_{ij} = \frac{n_{ij}}{N}$ to denote the fraction of the AGV population traversing the edge $(i, j)$ at the same time.

The QP in table 3 is not solvable using existing QP/LP solvers [35] as they can not handle quadratic terms as $f_{ij}^2$. Similarly, existing solvers also cannot handle the flow constraint. Existing continuous variants of the max-sum approach [39] does not seem readily applicable as even though $f_{ij}$ variables are continuous, the max-sum approach requires piecewise linear utility functions whereas our objective function (12) is quadratic. Furthermore, flow constraints (13) are hard constraints for which support is not provided in the continuous max-sum. To solve such programs with both quadratic and linear terms, we next describe our main contribution—a graph based optimization framework and a message passing solution approach.

## 3 GRAPH-BASED OPTIMIZATION

We now describe our graph-based optimization framework. We are given a pairwise graph $G = (V, E)$. The semantics of this graph is same as for DCOPs. Each node $i \in V$ can be considered an agent with an associated *random variable* $x_i$. With each edge, there is a potential function $\theta_{ij}$ associated, and with each node a unary function $\theta_i$ is associated. The key difference in our approach lies in providing explicit support for linear constraints defined over nodes and edges of the graphs. Specifically, a set of linear constraints $M$ is also provided. Our goal is to solve the program with the structure in table 4.

We next explain the structure of this program.

- Associated with each random variable $x_i$ is a probability distribution $\mu_i(x_i)$. Its contribution to the objective function is $\sum_{x_i\in D_i}\left[\mu_i(x_i)\theta_i(x_i) + \alpha_i(x_i)\mu_i(x_i)^2\right]$, where $\alpha_i(x_i) \geq 0$ is a given parameter. Existing QP/LP solvers do not encode quadratic terms $\alpha(x)\mu^2(x)$ in a straightforward manner.

$$\max_{\boldsymbol{\mu}} \sum_{i\in V}\sum_{x_i}\mu_i(x_i)\theta_i(x_i) + \sum_{(i,j)\in Q}\sum_{x_i x_j}\mu_i(x_i)\mu_j(x_j)\theta_{ij}(x_i,x_j)$$

$$+ \sum_{(i,j)\in L}\sum_{x_i x_j}\mu_{ij}(x_i,x_j)\theta_{ij}(x_i,x_j) + \sum_{i\in V}\sum_{x_i}\alpha_i(x_i)\mu_i(x_i)^2 \qquad (15)$$

$$\text{s.t.} \sum_{i\in \mathrm{V}_m}\sum_{x_i} c_i^m(x_i)\mu_i(x_i) +$$

$$\sum_{(i,j)\in \mathrm{E}_m}\sum_{x_i,x_j} c_{ij}^m(x_i,x_j)\mu_{ij}(x_i,x_j) = k_m \ \forall m \in M \qquad (16)$$

$$0 \le \boldsymbol{\mu} \le 1 \qquad (17)$$

**Table 4: GraphOpt: Graph-based optimization**

- We are given a partition of the edge set as $E = L \cup Q$. We associate probabilities $\{\mu_{ij}(x_i, x_j)\forall x_i, x_j\}$ with each edge $(i, j) \in L$. The contribution of an edge $(i, j) \in L$ to the objective is $\sum_{x_i,x_j} \mu_{ij}(x_i, x_j)\theta_{ij}(x_i, x_j)$. Intuitively, this forms the *linear* part of the objective in parameters $\mu_{ij}$. The edge set $L$ is known as the set of *LP edges*.
- The contribution of an edge $(i, j) \in Q$ to the objective function is $\sum_{x_i,x_j} \mu_i(x_i)\mu_j(x_j)\theta_{ij}(x_i, x_j)$. This forms the quadratic part of the objective function; and the edge set $Q$ is known as the set of *QP edges*.

The overall objective function defined over the edges $E$ is shown in program (15). Notice that, we allow edge partitions $L$ or $Q$ to be empty, implying a linear program at one extreme ($Q = \phi$ and all $\alpha_i = 0$) or a quadratic program at another extreme ($L = \phi$). The distributions $\boldsymbol{\mu} = \{\mu_i, \mu_{ij}\}$ are the parameters to optimize by solving the program (15).

**Constraints:** In several real world problems, constraints model important domain characteristics. E.g., for optimizing network flow, flow conservation at each node must be modeled. In our constraint template (16), we allow high flexibility for defining a rich class of constraints. Each linear constraint $m \in M$ entails:

- The constraint $m$ can involve a subset $\mathrm{V}_m \subseteq V$ of nodes. Each node $i \in \mathrm{V}_m$ contributes the term $\sum_{x_i} c_i^m(x_i)\mu_i(x_i)$ to constraint $m$. Each coefficient $c_i^m(x_i)$ can be positive or negative or zero. We further partition the node set as $\mathrm{V}_m = \mathrm{V}_m^+ \cup \mathrm{V}_m^-$, which intuitively denote the set of nodes with positive or negative coefficients $c_i$. A node $i \in \mathrm{V}_m$ belongs either to $\mathrm{V}_m^+$ or $\mathrm{V}_m^-$, but not both. For all nodes $i \in \mathrm{V}_m^+$, we have $c_i^m(x_i) \ge 0 \ \forall x_i$, and $c_i^m(x_i) \le 0 \ \forall x_i \forall i \in \mathrm{V}_m^-$.
- The constraint $m$ can involve a subset $\mathrm{E}_m \subseteq L$ of LP edges. Each such edge contributes $\sum_{x_i,x_j} c_{ij}^m(x_i, x_j)\mu_{ij}(x_i, x_j)$ to constraint $m$. Similar to the node set $\mathrm{V}_m$, we partition the edge set as $\mathrm{E}_m = \mathrm{E}_m^+ \cup \mathrm{E}_m^-$ depending on coefficients $c_{ij}^m(\cdot,\cdot)$ being positive or negative.
- The constant term is $k_m \in \Re$.

For simplicity, we have restricted exposition to equality constraints only. Our approach can be extended to inequality constraints also. The probability normalization constraints ($\sum_{x_i} \mu_i(x_i) = 1$), and marginalization constraints ($\sum_{x_j} \mu_{ij}(x_i, x_j) = \mu_i(x_i)$ for each LP edge) fit the template (16), and thus are not explicitly stated.

**Constraint types:** It is easy to see that our framework subsumes existing QP/LP formulations. The major benefit of our framework

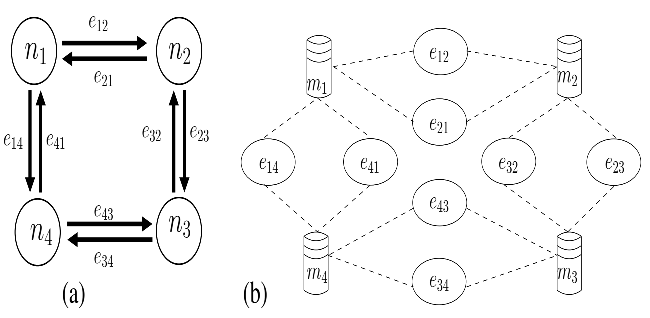

**Figure 2: Constraint augmented graph**

is that it can model several different types of common constraints in template (16). For example, our approach can handle standard normalization, marginalization constraints, resource constraints and network flow constraints. *Probability marginalization* constraint for the edge $(i, j)$ and a particular assignment $x_j$ is $\sum_{x_i} \mu_{ij}(x_i, x_j) - \mu_j(x_j) = 0$. We have $\mathrm{V}_m = \{j\}$, $\mathrm{E}_m = (i, j)$, $k_m = 0$. The term $-\mu_j(x_j)$ can be simulated by setting $c_j^m(x_j) = -1$ and $c_j^m(x_j') = 0$ for the rest of domain values $x_j'$. In a similar manner, we can get the term $\sum_{x_i} \mu_{ij}(x_i, x_j)$ by appropriately setting $c_{ij}^m$ values. Similarly, other linear constraints can be modeled.

**Constraint Augmented Graph:** Based on the given graph $G = (V, E)$ and the constraint set $M$, we define a *constraint-augmented* (CAG) undirected graph $G' = (V', E')$. The node set $V' = V \cup M$ includes one node for each agent $i$ and constraint $m$. Each $(i, j) \in E$ in the original edge set $E$ is included in $E'$. In addition, if an agent $i$ participates in the constraint $m$ ($i \in \mathrm{V}_m$), then we add an edge $(i, m)$ to $E'$. If an edge $(i, j) \in L$ participates in a constraint $m$, then we add edges $(i, m)$ and $(j, m)$ to $E'$. Our approach would be to solve the program (15) by performing distributed message-passing over this graph. Notice that in this graph, an agent only knows about its immediate neighbor agents $j$ and constraints $m$.

A small instance of the congestion-based routing problem is shown in Fig. 2(a). As mentioned in section 2.2, in the model of this problem each edge in the Fig. 2(a) graph becomes a variable, and for each flow constraint (one for each of the 4 nodes in Fig. 2(a)), we create new nodes ($m_1$ to $m_4$) in Fig. 2(b). Edges are created between different $m_i$ and $e_{ij}$ variables to account for flow constraints. Each variable $e_{ij}$ is binary with $\Pr(e_{ij} = 1) = f_{ij}$, $\Pr(e_{ij} = 0) = 1 - f_{ij}$, where each $f_{ij}$ are as defined in the program of table 3. Detailed mapping of the congestion based routing to the GraphOpt framework is provided in the extended version of the paper. Our solution approach performs local message-passing along the edges of CAG.

## 4 INFERENCE FOR DISTRIBUTED OPTIMIZATION

We first provide an intuition behind our approach to solve the GraphOpt program in table 4. Instead of optimizing the objective (15) directly, we optimize the log of this expression. As log is a monotonic function, this would not change the optimal solution. Taking the log of an expression does not by itself makes the problem easier. E.g., we may have an expression $\log(x^2 + xy + z)$. However,

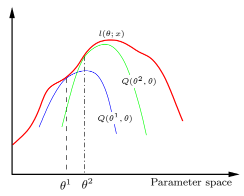

**Figure 3: The coordinate ascent strategy of the EM algorithm**

if we are able to apply log to each term as $\log x^2 + \log xy + \log z$, we get $2\log x + \log x + \log y + \log z$. This last expression is particularly advantageous w.r.t. maximization as log is a concave function making the whole expression concave, and thus the problem becomes convex optimization problem. We can then use the rich theory of convex optimization to efficiently solve this problem.

However, we cannot directly move log inside to each term. Instead, we use the well known Expectation-Maximization (EM) algorithm [5] which makes optimization with log terms easier. We first provide a high level overview of the EM algorithm followed by detailing its adaptation to GraphOpt. The EM algorithm is a general approach to the problem of maximum likelihood (ML) parameter estimation in models with latent variables. Let $X$ denote observed variables and $Z$ denote the hidden variables. Let $\theta$ denote the model parameters to be learned. The ML problem is to solve the following optimization problem given the observation $X=x$:

$$\max_\theta l(\theta;x) = \max_\theta \log \sum_z p(x,z;\theta) \tag{18}$$

It is hard to optimize the above problem as the summation is inside the log. Furthermore, maximizing the log-likelihood $l(\theta;x)$ is generally a non-convex optimization problem as shown in Figure 3. Therefore, the EM algorithm iteratively performs coordinate ascent in the parameter space. First, the EM algorithm computes a lower bound for the function $l(\theta;x)$ such that this lower bound touches $l(\theta;x)$, say at point $\theta^1$. This lower bound is denoted as $Q(\theta^1,\theta)$, defined as:

$$Q(\theta^1,\theta) = \sum_z p(z|x;\theta^1)\log p(x,z;\theta) - \sum_z p(z|x;\theta^1)\log p(z|x;\theta^1) \tag{19}$$

The last term in the above expression is the entropy of the variable $Z|x$. Figure 3 shows the lower bound $Q(\theta^1,\theta)$ by a blue curve. The function $Q(\theta^1,\theta)$ is also called *expected complete log-likelihood*. Importantly, the lower bound $Q$ is *concave* in parameters $\theta$ and thus, can be optimized globally to provide a better parameter estimate $\theta^2$. This process is also shown in Figure 3. Such coordinate ascent continues iteratively by defining a new lower bound $Q(\theta^2,\theta)$ at the point $\theta^2$ as also shown in Figure 3, and then optimizing it to yield the next better parameter estimate until convergence.

To connect such an iterative maximization strategy to solving GraphOpt, we first make a graphical model with some hidden random variables $Z$ and some observed variables $X$ such that maximizing the likelihood of observed data in this model is directly proportional to our objective (15). We show that this graphical model takes the form of a mixture of simple Bayes nets (BN) (see fig. 4) with parameters $\boldsymbol{\mu}$, same as in GraphOpt. We then perform the following steps of the EM algorithm:

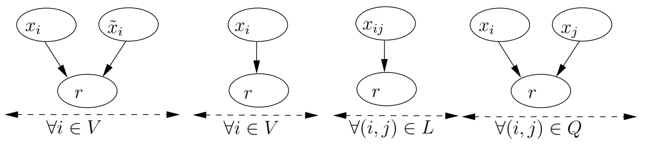

**Figure 4: Mixture of BNs for GraphOpt in table 4**

(1) (E-step) Formulate the expected complete log-likelihood $Q(\boldsymbol{\mu}^i,\boldsymbol{\mu})$ for the BN mixture model for iteration $i$
(2) (M-step) Maximize $Q(\boldsymbol{\mu}^i,\boldsymbol{\mu})$ w.r.t. $\boldsymbol{\mu}$ to yield better parameters $\boldsymbol{\mu}^{i+1}$
(3) Repeat steps 1 and 2 until convergence, i.e., until $\boldsymbol{\mu}^i \approx \boldsymbol{\mu}^{i+1}$

A key benefit of the EM algorithm is that while maximizing $Q(\boldsymbol{\mu}^i,\boldsymbol{\mu})$, we can also impose convex constraints over parameters $\boldsymbol{\mu}$. We can thus impose arbitrary linear constraint with template (16) while optimizing $\boldsymbol{\mu}$. We next detail EM approach for GraphOpt problem.

## 4.1 Mixture model for GraphOpt

The decision making-as-inference strategy has been used previously for DCOPs [11, 17], and for sequential multiagent problems [19, 40]. However, as highlighted in section 2.1, previous DCOP approaches solve a limited class of math programs. Our approach handles richer objective function and constraints than such previous approaches, and requires corresponding technical advances when deriving updates for the EM algorithm.

We create a mixture of simple Bayesian networks (BNs) corresponding to the GraphOpt problem in table 4 (refer to [6] for mixture models and EM). There are two mixture component for each node $i \in V$, one for each edge $(i,j) \in L$ and each edge $(i,j) \in Q$. Thus, there are total $2|V| + |E|$ mixture components. The mixture random variable is denoted using $W$ and it has a fixed uniform distribution $1/(2|V| + |E|)$.

Fig. 4 shows the structure of each of four types of BNs. Intuitively, the first type of BNs model the quadratic term $\alpha_i(x_i)\mu_i(x_i)^2$. The variable $x_i$ corresponds to the standard DCOP variable with the domain $D_i$. In addition, we create another variable $\tilde{x}_i$ with the same domain $D_i$. We further introduce a *binary* reward variable $r$ whose conditional probability is directly proportional to $\alpha_i(x_i)$ providing a link between the likelihood in this model and our objective in (15). The parameters of this model are set as:

$$P(x_i) = \mu_i(x_i)\ ,\ P(\tilde{x}_i) = \mu_i(x_i) \tag{20}$$

$$P(r=1|x_i,\tilde{x}_i) = \begin{cases} \frac{\alpha_i(x_i)+1}{K} & \text{if } x_i = \tilde{x}_i \\ \frac{1}{K} & \text{otherwise} \end{cases} \tag{21}$$

where $K$ is a large enough positive constant such that each $\frac{\alpha_i(x_i)+1}{K}$ is less than 1 to make it a probability. We also compute the probability $P(r=1)$, which will be used later for computing the likelihood, as below:

$$= \sum_{x_i,\tilde{x}_i:x_i=\tilde{x}_i} \mu_i(x_i)^2 P(r=1|x_i,\tilde{x}_i) + \sum_{x_i,\tilde{x}_i:x_i\neq\tilde{x}_i} \mu_i(x_i)\mu_i(\tilde{x}_i)P(r=1|x_i,\tilde{x}_i)$$

$$= \sum_{x_i} \frac{\alpha_i(x_i)+1}{K}\mu_i(x_i)^2 + \frac{1}{K}\Big(1 - \sum_{x_i}\mu_i(x_i)^2\Big) \tag{22}$$

$$= \sum_{x_i} \frac{\alpha_i(x_i)}{K}\mu_i(x_i)^2 + \frac{1}{K} \tag{23}$$

In the above derivation, we used the result $\sum_{x_i,\tilde{x}_i}\mu_i(x_i)\mu_i(\tilde{x}_i)=1$. Using this we have $\sum_{x_i,\tilde{x}_i:x_i\neq\tilde{x}_i}\mu_i(x_i)\mu_i(\tilde{x}_i)=1-\sum_{x_i}\mu_i(x_i)^2$.

For the second type of BN components in fig. 4, we set the parameters and compute the probability $P(r=1)$ as follows:

$$P(x_i)=\mu_i(x_i)\forall x_i\in D_i,\quad P(r=1|x_i)=\frac{\theta_i(x_i)-\theta_{min}}{K}=\hat{\theta}_{x_i} \tag{24}$$

$$P(r=1)=\sum_{x_i}\mu_i(x_i)\hat{\theta}_{x_i} \tag{25}$$

For the third type of BN components in fig. 4, we set the parameters and compute the probability $P(r=1)$ as follows:

$$P(x_{ij}=(x_i,x_j))=\mu_{ij}(x_i,x_j)\ \forall x_i\in D_i,\ x_j\in D_j \tag{26}$$

$$P(r=1|x_{ij}=(x_i,x_j))=\frac{\theta_{ij}(x_i,x_j)-\theta_{min}}{K}=\hat{\theta}_{x_ix_j} \tag{27}$$

$$P(r=1)=\sum_{x_i,x_j}\mu_{ij}(x_i,x_j)\hat{\theta}_{x_ix_j} \tag{28}$$

For the fourth type of BN components in fig. 4, we set the parameters and compute the probability $P(r=1)$ as follows:

$$P(x_i)=\mu_i(x_i)\forall x_i\in D_i,\ P(x_j)=\mu_j(x_j)\ \forall x_j\in D_j \tag{29}$$

$$P(r=1|x_i,x_j)=\frac{\theta_{ij}(x_i,x_j)-\theta_{min}}{K}=\hat{\theta}_{x_ix_j} \tag{30}$$

$$P(r=1)=\sum_{x_i,x_j}\mu_i(x_i)\mu_j(x_j)\hat{\theta}_{x_ix_j} \tag{31}$$

Notice that we set the constant term $K$ such that all conditional probabilities in (21),(24),(27),(30) are less than 1. We next have the result connecting the likelihood of $r=1$ in this mixture model with that of GraphOpt objective function.

Theorem 2. *Maximizing the likelihood $P(r=1;\boldsymbol{\mu})$ of observing the variable $r=1$ in the mixture model of figure 4 subject to constraints $M$ in* (16) *is equivalent to solving the GraphOpt problem.*

Proof. The probability of observing $r=1$ in the mixture model is given as:

$$P(r=1;\boldsymbol{\mu})=\frac{1}{2|V|+|E|}\Big[\sum_{i\in V}P_1(r=1)+\sum_{i\in V}P_2(r=1)$$
$$\sum_{(i,j)\in L}P_3(r=1)+\sum_{(i,j)\in Q}P_4(r=1)\Big]$$

where $\frac{1}{2|V|+|E|}$ is the fixed distribution of the mixture variable $W$. We get the above summation by marginalizing out the mixture variable $W$. Probability $P_1(\cdot)$ refer to the probability $P(r=1)$ in the first mixture model component given in (23). It encodes the quadratic term $\alpha_i(x_i)\mu_i(x_i)^2$ of (15). Similarly, $P_2(\cdot)$ refers to the expression (25) encoding the unary value $\mu_i(x_i)\theta_i(x_i)$, $P_3$ in expression (28) encodes the linear term $\mu_{ij}(x_i,x_j)\theta_{ij}(x_i,x_j)$, and $P_4$ in (31) encodes the bilinear term $\mu_i(x_i)\mu_j(x_j)\theta_{ij}(x_i,x_j)$. Therefore, the likelihood is directly proportional to our objective (15), and parameters $\boldsymbol{\mu}$ subject to linear constraints (16). Thus, maximizing the likelihood solves the GraphOpt problem. □

**Optimality:** The EM algorithm converges to a stationary point of the log-likelihood [5]. If our program is a LP, then the log of objective is concave. Therefore, EM converges to a stationary point of this concave function subject to linear constraints, which is also the optimal solution [3].

## 4.2 Maximizing expected log-likelihood

As highlighted earlier, the EM algorithm maximizes the expected log-likelihood given in (19). In our mixture model, only the variable $r=1$ is observed; rest of the variables are hidden. We substitute different probabilities in (19) using our mixture model and get the following M-step optimization problem (proof in the extended version):

$$\min_{\boldsymbol{\mu}^\star}\sum_{i\in V,x_i}-\log\mu_i^\star(x_i)\Big[\hat{\theta}_{x_i}\mu_i(x_i)+\frac{2\alpha_i(x_i)\mu_i(x_i)^2+2\mu_i(x_i)}{K}+$$
$$\sum_{j\in \mathrm{Nb}_Q^i}\sum_{x_j}\hat{\theta}_{x_ix_j}\mu_i(x_i)\mu_j(x_j)\Big]-\sum_{(i,j)\in L,x_i,x_j}\hat{\theta}_{x_ix_j}\mu_{ij}(x_i,x_j)\log\mu_{ij}^\star(x_i,x_j) \tag{32}$$
$$\text{s.t.}\sum_{i\in \mathrm{V}_m}\sum_{x_i}c_i^m(x_i)\mu_i^\star(x_i)+\sum_{(i,j)\in \mathrm{E}_m,x_i,x_j}c_{ij}^m(x_i,x_j)\mu_{ij}^\star(x_i,x_j)=k_m\ \forall m$$

where $\mathrm{Nb}_Q^i$ denotes the neighbors $j$ of the node $i$ in the constraint graph $G$ such that $(i,j)$ is a *QP edge*. Notice that we negated the objective to make it a minimization problem. The M-step problem does not admit closed form solutions. Therefore, we develop an iterative approach based on convex optimization techniques to solve the M-step. Our high level approach is:

- We first write the Lagrangian dual of the problem (32). This dual has much simpler structure and constraints, making its optimization easier. As (32) is a convex optimization problem, optimal dual solution equals the optimal primal solution [3].
- We optimize the dual of M-step problem by using block coordinate ascent (BCA) method wherein we fix all the dual variables except one, and then optimize the dual over the one variable.

**M-step Dual:** The dual problem is given as $\max_{\boldsymbol{\lambda}} q(\boldsymbol{\lambda})$, where $\boldsymbol{\lambda}=\{\lambda_m\ \forall m\in M\}$ is the set of dual variables to be optimized for each constraint $m$. The dual problem is (derivation in extended version):

$$\max_{\boldsymbol{\lambda}}\sum_{i\in V,x_i}\delta_i(x_i)\log\Big(\sum_{m\in M(i)}\lambda_m c_i^m(x_i)\Big)+\sum_{(i,j)\in L,x_i,x_j}\delta_{ij}(x_i,x_j)\times$$
$$\log\Big(\sum_{m\in M(i,j)}\lambda_m c_{ij}^m(x_i,x_j)\Big)-\sum_{m\in M}\lambda_m k_m+\langle\text{ind terms of }\boldsymbol{\lambda}\rangle \tag{33}$$

The set $M(i)$ denotes all the constraints $m$ in which agent $i$ participates ($i\in \mathrm{V}_m$), analogously $M(i,j)$ denotes constraints $m$ such that $(i,j)\in \mathrm{E}_m$. We also used the below shorthand:

$$\delta_i(x_i;\boldsymbol{\mu})=\hat{\theta}_{x_i}\mu_i(x_i)+\frac{2\alpha_i(x_i)\mu_i(x_i)^2+2\mu_i(x_i)}{K}$$
$$+\sum_{j\in \mathrm{Nb}_Q^i}\sum_{x_j}\hat{\theta}_{x_ix_j}\mu_i(x_i)\mu_j(x_j) \tag{34}$$

$$\delta_{ij}(x_i,x_j;\boldsymbol{\mu})=\hat{\theta}_{x_ix_j}\mu_{ij}(x_i,x_j) \tag{35}$$

Using results from convex optimization [3], for any value of $\boldsymbol{\lambda}$, we have $q(\boldsymbol{\lambda})\leq Q^{\text{opt}}$, where $Q^{\text{opt}}$ is the optimal solution of (32). To optimize the dual, we follow the BCA strategy noted earlier. We start with an initial assignment to all $\boldsymbol{\lambda}$ variables. We then cycle through all the $\lambda_m$ variables (for each constraint $m$) optimizing the dual (33) w.r.t. the chosen $\lambda_m$ variables while keeping other variables fixed. Optimization w.r.t. a single $\lambda_m$ variable can be done by setting the partial derivative of the dual w.r.t. $\lambda_m$ to zero. This iterative process (called BCA) is guaranteed to converge to the optimal dual solution as our dual (33) is 1) continuously differentiable over its domain; 2)

**Algorithm 1: solveGraphOpt**

```
1  Initialize: $\mu^\star_i(x_i) \leftarrow \frac{1}{|D_i|}\ \forall x_i, \forall i \in V$
2  $\mu^\star_{ij}(x_i, x_j) \leftarrow \mu^\star_i(x_i)\mu^\star_j(x_j)\ \ \forall x_i, x_j, \forall (i,j) \in L$
3  repeat
4      $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu}^\star$
5      Send $\beta_{i\to j}(x_j) \leftarrow \sum_{x_i} \hat{\theta}_{x_i x_j}\mu_i(x_i) \forall j \in \mathrm{Nb}^i_Q,\ \forall i \in V$
6      Send $\gamma_{i\to m}(x_i) \leftarrow \hat{\theta}_{x_i}\mu_i(x_i) + \frac{2\alpha_i(x_i)\mu_i(x_i)^2+2\mu_i(x_i)}{K} +$
         $\mu_i(x_i)\sum_{j\in\mathrm{Nb}^i_Q}\beta_{j\to i}(x_i), \forall m \in M(i), \forall i \in V$
7      Send $\gamma_{ij\to m}(x_i,x_j) \leftarrow \hat{\theta}_{x_i x_j}\mu_{ij}(x_i,x_j),\ \forall m \in M(i,j), \forall (i,j) \in L$
8      $\{\lambda_m\ \forall m \in M\} \leftarrow$ solveBCA($\{\gamma_i\}, \{\gamma_{ij}\}$)
9      $\mu^\star_i(x_i) \leftarrow \frac{\gamma_i(x_i;\boldsymbol{\mu})}{\sum_{m\in M(i)}\lambda_m c^m_i(x_i)}\ \forall x_i, \forall i \in V$
10     $\mu^\star_{ij}(x_i,x_j) \leftarrow \frac{\gamma_{ij}(x_i,x_j;\boldsymbol{\mu})}{\sum_{m\in M(i,j)}\lambda_m c^m_{ij}(x_i,x_j)} \forall x_i, x_j, \forall (i,j) \in L$
11 until $\boldsymbol{\mu}^\star \neq \boldsymbol{\mu}$
12 return $\boldsymbol{\mu}^\star$
```

is strictly concave w.r.t. each dual variable $\lambda_m$ due to the presence of log terms in (33), resulting in a unique solution for BCA [3].

**Message passing implementation:** During BCA, since we optimize the dual w.r.t. a single $\lambda_m$ one-by-one, this process is particularly amenable to a message-passing implementation over the constraint augmented graph as shown in fig. 2. We omit the derivation details for BCA due to space, instead present a message-passing implementation of EM in algorithm 1 and 2. The communication in this approach takes place among the following entities:

- An agent $i$ in the constraint graph $G=(V,E)$ can communicate with its immediate neighbor agents $j$
- For exposition ease, we assume that there is a pseudo-agent $ij$ for each LP edge $(i,j) \in L$. In implementation, any computation or message exchanges performed by this agent $ij$ can be performed by either agent $i$ or $j$.
- Let $M(i)$ denote the set of constraints $m$ which involve agent $i$ or $i \in \mathrm{V}_m$. An agent $i$ can directly communicate with all constraints $m \in M(i)$.
- Let $M(i,j)$ denote the set of constraints $m$ which involve edge $(i,j)$ or $(i,j) \in \mathrm{E}_m$. An agent $ij$ can communicate with each $m \in M(i,j)$.
- Any constraint $m$ can communicate with all agents involved in $m$ or all $i \in \mathrm{V}_m$ and all $(i,j) \in \mathrm{E}_m$.

Algorithm 1 shows the outer loop of the EM. Parameters $\boldsymbol{\mu}$ are initialized uniformly. In each outer loop (from lines 4-10), every agent $i$ sends the message $\beta$ to all its QP neighbors $\mathrm{Nb}^i_Q$. After all agents have received respective $\beta$ messages from their neighbors, they send the message $\gamma_i$ to all the constraints $m \in M(i)$ (line 6), and $\gamma_{ij}$ to all $m \in M(i,j)$ (line 7). Once all constraints $m$ have received $\gamma$ messages, the message-passing corresponding to the BCA approach starts in algorithm 2. To start inner loop iterations in alg. 2, dual variables $\lambda_m$ should be initialized to some value (line 2) such that the dual function (33) is not negative infinity. Based on the exact value of coefficients $c_i, c_{ij}$, it is relatively straightforward to generate such an initial assignment (setting variables either zero or one generally works). These dual variables $\lambda_m$ are sent to the involved agents (lines 3,4). Based on the received $\lambda$ values, each agent $i$ computes a function $f_m$ for each constraint $m \in M(i)$ in line 6, and sends this function to each constraint $m \in M(i)$ in line

**Algorithm 2: solveBCA($\{\gamma_i\}, \{\gamma_{ij}\}$)**

```
1  Initialize:
2  $\{\lambda_m\ \forall m \in M\} \leftarrow$ FindFeasible($\{\gamma_i\}, \{\gamma_{ij}\}$)
3  Send $\lambda_{m\to i} \leftarrow \lambda_m,\ \forall i \in \mathrm{V}_m,\ \forall m \in M$
4  Send $\lambda_{m\to ij} \leftarrow \lambda_m,\ \forall (i,j) \in \mathrm{E}_m,\ \forall m \in M$
5  for each node $i \in V$ do
6      Compute $f_m(x_i) = \sum_{m'\in M(i)\setminus m}\lambda_{m'}c^{m'}_i(x_i), \forall m \in M(i)$
7      Send $f_{i\to m}(x_i) \leftarrow f_m(x_i), \forall m \in M(i)$
8  for each LP edge $(i,j) \in L$ do
9      Compute $g_m(x_i,x_j) = \sum_{m'\in M(i,j)\setminus m}\lambda_{m'}c^{m'}_{ij}(x_i,x_j), \forall m \in M(i,j)$
10     Send $g_{ij\to m}(x_i,x_j) \leftarrow g_m(x_i,x_j), \forall m \in M(i,j)$
11 repeat
12     for each constraint $m \in M$ do
13         Find unique root $\lambda^\star_m$ of $h(\lambda_m) = 0$ ; $\lambda_m \in (\lambda^{\min}_m, \lambda^{\max}_m)$
14         Send $\lambda_{m\to i} \leftarrow \lambda^\star_m,\ \forall i \in \mathrm{V}_m$
15         Send $\lambda_{m\to ij} \leftarrow \lambda^\star_m,\ \forall (i,j) \in \mathrm{E}_m$
16         for each node $i \in V_m$ do
17             $f_m(i,x_i) = \sum_{m'\in M(i)\setminus m}\lambda_{m'}c^{m'}_i(x_i), \forall m \in M(i)$
18             Send $f_{i\to m}(x_i) \leftarrow f_m(i,x_i), \forall m \in M(i)$
19         for each edge $(i,j) \in E_m$ do
20             $g_m(x_i,x_j) = \sum_{m'\in M(i,j)\setminus m}\lambda_{m'}c^{m'}_{ij}(x_i,x_j), \forall m \in M(i,j)$
21             Send $g_{ij\to m}(x_i,x_j) \leftarrow g_m(x_i,x_j), \forall m \in M(i,j)$
22 until convergence
23 return $\{\lambda_m\ \forall m \in M\}$
```

7. Same process repeats for agents $ij$ in lines 8-10. After that, each constraint $m \in M$ starts updating their dual variables $\lambda_m$ in lines 12-21. The function $h(\lambda_m)$ is:

$$h(\lambda_m) = \sum_{(i,j)\in\mathrm{E}_m, x_i, x_j} c^m_{ij}(x_i,x_j)\frac{\gamma_{ij\to m}(x_i,x_j)}{\lambda_m c^m_{ij}(x_i,x_j) + g_{ij\to m}(x_i,x_j)} + \sum_{i\in\mathrm{V}_m, x_i} c^m_i(x_i)\frac{\gamma_{i\to m}(x_i)}{c^m_i(x_i)\lambda_m + f_{i\to m}(x_i)} - k_m = 0 \tag{36}$$

This function can have multiple roots. However, by imposing the condition that for any dual value $\lambda_m$, the dual function must remain positive, we can narrow down to the unique root in the interval (derivation omitted):

$$\lambda^{min}_m = \max\left(\max_{i\in\mathrm{V}^+_m, x_i}\frac{-f_{i\to m}(x_i)}{c_i(x_i)}, \max_{(i,j)\in\mathrm{E}^+_m, x_i, x_j}\frac{-g_{ij\to m}(x_i,x_j)}{c^m_{ij}(x_i,x_j)}\right)$$

$$\lambda^{max}_m = \min\left(\min_{i\in\mathrm{V}^-_m, x_i}\frac{-f_{i\to m}(x_i)}{c^m_i(x_i)}, \min_{(i,j)\in\mathrm{E}^-_m, x_i, x_j}\frac{-g_{ij\to m}(x_i,x_j)}{c^m_{ij}(x_i,x_j)}\right)$$

Once the constraint $m$ computes the new estimate $\lambda^\star_m$, all agents $i \in \mathrm{V}_m$ and $(i,j) \in \mathrm{E}_m$ update their estimates of the $f_m$ and $g_m$ functions in lines 16-21. Convergence of inner loop is easily detected by measuring the constraint violations (of constraints (16)); if maximum violation falls below a given threshold, the inner loop terminates.

## 5 EXPERIMENTS

We test on congestion-based routing problems described in section 2.2 and standard DCOP benchmarks based on random graphs and sensor networks. Our approach was implemented in Python and used C-based GNU Scientific library for root finding procedure. We compare our EM based solver with several standard approximate DCOP solvers such as Max-Sum (MS), DSA, and an efficient centralized solver Toulbar2 [1] which provides a strong baseline for solution quality comparisons. We used Frodo 2.0's [22] implementation of MS and DSA (default $p=0.5$). We set iterations for each approach (EM, MS, DSA) to 1000. Each data point is an average over 10 instances. We always show normalized solution quality. For the Toulbar2, we set 30 min. limit.

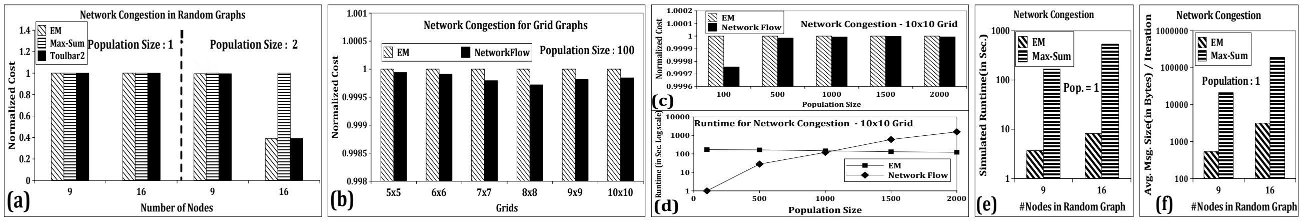

**Figure 5: Solution quality, Simulated Runtime and Message size comparisons for congestion-based routing instances**

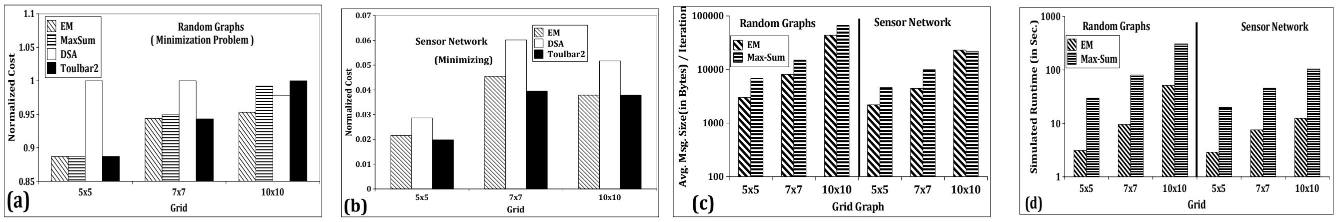

**Figure 6: Solution quality, Simulated Runtime and Message size comparisons on random and sensor network problem**

**Congestion aware routing:** We tested on random and grid shaped graphs with varying AGV population size. For DCOP solvers, we first convert the problem instance to a DCOP instance using $n_{ij}$ variables (as noted in sec. 2.2). We compare DCOP solvers against EM that solves the QP formulation in table 3. If the degree of a node in the graph is $d$, and AGV population is $N$, then it requires $2d$-ary constraints modeling flow conservation, and variable domain size is $N$. Thus, constraints tables are exponentially sized requiring $O(N^{2d})$ space. Due to this reason, DCOP solvers did not scale up to solve large networks. Fig. 5(a) shows comparison on small 9 and 16 node graphs with small population size. All the approaches, except DSA, provided good solution for $N = 1$. EM always achieved optimal solution, same as Toulbar2. DSA failed for these problems and did not find any feasible solution satisfying the flow constraints highlighting how the presence of functional hard constraints adversely affects the accuracy of previous approximate solvers.

On larger grid graphs and higher population size, none of the DCOP solvers scaled up. For larger grid graphs in fig. 5(b), we provide comparisons against an optimal network flow based solver [27] that can solve symmetric congestion games with linear congestion cost. The cost of EM's solution is only marginally worse than the network flow solver showing the accuracy of the QP approximation. For larger population $N$, results in fig. 5(c,d) show that the optimal solver scales poorly with the increasing $N$. The runtime of the network flow solver increases exponentially with the increasing $N$, whereas EM has nearly constant (centralized) runtime ($\approx$180 sec) with varying $N$. These set of results confirm that for problems with functional constraints, it is more tractable to solve a graph-based math program that handles functional constraints explicitly.

Fig. 5(e) show the simulated runtime for small 9 and 16 node instances solvable using DCOP solvers. This result shows that EM is orders of magnitude faster than MS. As EM's message-passing structure and MS's message-passing are not equivalent, for fairness sake, we show total *average* network load per iteration of MS and per outer loop of EM in fig. 5(f). That is, for EM, per iteration load counts messages exchanged in a single outer loop and *all* the inner loops. Fig. 5(f) clearly shows that EM has significantly lower network overhead leading to speedups provided by EM.

**DCOP instances:** We tested EM on standard DCOP benchmarks: random grid graphs and sensor network problems [24]. Fig. 6(a) shows results for random graphs. Each cost is uniformly sampled from [1, 100] and domain $|D_i| = 4$. We note that solution quality of EM is at par with the solution of Toulbar2 for 5x5 and 7x7 grids. However, for a larger 10x10 grid, EM achieves better quality than other solvers. Fig. 6(b) shows results on sensor network domain. We used 4 time slots for sensors, and target detection cost was selected uniformly from range [1, 200]. These results confirm that EM achieves better solution quality than DSA for all settings and similar quality as toulbar2. We omit results for MS as it found poor quality solutions violating hard constraints for several instances. Figures 6(c,d) show simulated runtime and network load (per iteration) comparisons between EM and MS (both in log-scale). These results show that EM had lower network load than MS for most instances, and was faster than MS. These results confirm that our approach is competitive with existing DCOP solvers, and significantly more scalable in the presence of functional constraints.

## 6 CONCLUSION

We presented a general graph-based optimization framework called GraphOpt for multiagent coordination. Our framework augments DCOPs with the ability to address general functional constraints defined using the language of mathematical programs. This approach provides significant modeling advantages and tractability by alleviating the need for creating high arity utility tables. Our framework is also more general than previous QP/LP solvers as it can model a richer class of objective function and constraints than previous work. We also developed a message-passing approach, based on the EM algorithm, to solve GraphOpt problems. Empirically, it scaled much better than the previous DCOP approaches on benchmarks based on congestion based routing and provided higher quality solutions.

## ACKNOWLEDGMENTS

This research was supported by the Singapore Ministry of Education Academic Research Fund (AcRF) Tier 2 grant MOE2018-T2-1-179. Second author is also support by the Lee Kong Chian Fellowship.

## REFERENCES

[1] D. Allouche, S. de Givry, and T. Schiex. 2010. *ToulBar2, an open source exact cost function network solver.* Technical Report. INRA.

[2] Daniel S. Bernstein, Rob Givan, Neil Immerman, and Shlomo Zilberstein. 2002. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research* 27 (2002), 819–840.

[3] Dimitri P. Bertsekas. 1999. *Nonlinear Programming* (2nd ed.). Athena Scientific.

[4] E. Bowring, M. Tambe, and M. Yokoo. 2006. Multiply-constrained distributed constraint optimization. In *International Conference on Autonomous Agents and Multiagent Systems.* 1413–1420.

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical society, Series B* 39, 1 (1977), 1–38.

[6] Ivo D Dinov. 2008. Expectation Maximization and Mixture Modeling Tutorial. https://escholarship.org/uc/item/1rb70972. (2008).

[7] John C. Duchi, Alekh Agarwal, and Martin J. Wainwright. 2010. Distributed Dual Averaging In Networks. In *Advances in Neural Information Processing Systems.* 550–558.

[8] John C. Duchi, Alekh Agarwal, and Martin J. Wainwright. 2012. Dual Averaging for Distributed Optimization: Convergence Analysis and Network Scaling. *IEEE Trans. Automat. Contr.* 57, 3 (2012), 592–606.

[9] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. 2008. Decentralised Coordination of Low-power Embedded Devices Using the Max-sum Algorithm. In *International Joint Conference on Autonomous Agents and Multiagent Systems.* 639–646.

[10] Ferdinando Fioretto, Enrico Pontelli, and William Yeoh. 2018. Distributed Constraint Optimization Problems and Applications: A Survey. *JAIR* 61 (2018), 623–698.

[11] Supriyo Ghosh, Akshat Kumar, and Pradeep Varakantham. 2015. Probabilistic Inference Based Message-Passing for Resource Constrained DCOPs. In *In International Joint Conference on Artificial Intelligence.* 411–417.

[12] Saurabh Gupta, Palak Jain, William Yeoh, Satish Ranade, and Enrico Pontelli. 2013. Solving Customer-Driven Microgrid Optimization Problems as DCOPs. In *Proceedings of the International Workshop on Distributed Constraint Reasoning (DCR).* 45–59.

[13] Khoi D. Hoang, Ping Hou, Ferdinando Fioretto, William Yeoh, Roie Zivan, and Makoto Yokoo. 2017. Infinite-Horizon Proactive Dynamic DCOPs. In *International Conference on Autonomous Agents and Multi Agent Systems.* 212–220.

[14] Akshat Kumar, Boi Faltings, and Adrian Petcu. 2009. Distributed Constraint Optimization with Structured Resource Constraints. In *International Joint Conference on Autonomous Agents and Multiagent Systems.* 923–930.

[15] Akshat Kumar, Hala Mostafa, and Shlomo Zilberstein. 2016. Dual Formulations for Optimizing Dec-POMDP Controllers. In *In International Conference on Automated Planning and Scheduling.* 202–210.

[16] Akshat Kumar, William Yeoh, and Shlomo Zilberstein. 2011. On Message-Passing, MAP Estimation in Graphical Models and DCOPs. In *International Workshop on Distributed Constraint Reasoning (DCR).* 57–70.

[17] Akshat Kumar and Shlomo Zilberstein. 2010. MAP Estimation for Graphical Models by Likelihood Maximization. In *Advances in Neural Information Processing Systems.* 1180–1188.

[18] Akshat Kumar and Shlomo Zilberstein. 2011. Message-Passing Algorithms for Quadratic Programming Formulations of MAP Estimation. In *In International Conference on Uncertainty in Artificial Intelligence.* 428–435.

[19] Akshat Kumar, Shlomo Zilberstein, and Marc Toussaint. 2015. Probabilistic Inference Techniques for Scalable Multiagent Decision Making. *Journal of Artificial Intelligence Research* 53 (2015), 223–270.

[20] Tiep Le, Tran Cao Son, Enrico Pontelli, and William Yeoh. 2017. Solving distributed constraint optimization problems using logic programming. *TPLP* 17, 4 (2017), 634–683.

[21] Thomas Léauté and Boi Faltings. 2011. Coordinating Logistics Operations with Privacy Guarantees. In *International Joint Conference on Artificial Intelligence.* 2482–2487.

[22] Thomas Léauté, Brammert Ottens, and Radoslaw Szymanek. 2009. FRODO 2.0: An open-source framework for distributed constraint optimization. In *IJCAI-09 Distributed Constraint Reasoning Workshop (DCR" 09).* 160–164.

[23] R Maheswaran, J Pearce, and M Tambe. 2004. Distributed algorithms for DCOP: A graphical game-based approach. In *International Conference on Parallel and Distributed Computing Systems.* 432–439.

[24] Rajiv Maheswaran, Milind Tambe, Emma Bowring, Jonathan Pearce, and Pradeep Varakantham. 2004. Taking DCOP to the Real World: Efficient Complete Solutions for Distributed Event Scheduling. In *International Joint Conference on Autonomous Agents and Multiagent Systems.* 310–317.

[25] Roger Mailler and Victor Lesser. 2004. Solving Distributed Constraint Optimization Problems Using Cooperative Mediation. In *International Joint Conference on Autonomous Agents and Multiagent Systems.* 438–445.

[26] Toshihiro Matsui, Hiroshi Matsuo, Marius Silaghi, Katsutoshi Hirayama, and Makoto Yokoo. 2008. Resource Constrained Distributed Constraint Optimization with Virtual Variables. In *In AAAI Conference on Artificial Intelligence.* 120–125.

[27] C.A. Meyers and A.S. Schulz. 2012. The complexity of welfare maximization in congestion games. *Networks* 59 (2012), 252–260.

[28] Pragnesh Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. 2005. ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees. *Artificial Intelligence* 161, 1-2 (2005), 149–180.

[29] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. 2006. ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees. *Artificial Intelligence Journal* 161 (2006), 149–180.

[30] Duc Thien Nguyen, William Yeoh, and Hoong Chuin Lau. 2013. Distributed Gibbs: a memory-bounded sampling-based DCOP algorithm. In *In International conference on Autonomous Agents and Multi-Agent Systems.* 167–174.

[31] Selwyn Parker. 2016. Advent of the automated container port. *http://www.marinemec.com/news/view,advent-of-the-automated-container-port_44131.htm* (2016).

[32] A. Petcu and B. Faltings. 2005. DPOP: A Scalable Method for Multiagent Constraint Optimization. In *International Joint Conference on Artificial Intelligence.* 266–271.

[33] Marek Petrik and Shlomo Zilberstein. 2009. A Bilinear Programming Approach for Multiagent Planning. *J. Artif. Intell. Res. (JAIR)* 35 (2009), 235–274.

[34] Pradeep Ravikumar, Alekh Agarwal, and Martin J. Wainwright. 2010. Message-passing for Graph-structured Linear Programs: Proximal Methods and Rounding Schemes. *Journal of Machine Learning Research* (2010), 1043–1080.

[35] Pradeep Ravikumar and John Lafferty. 2006. Quadratic programming relaxations for metric labeling and Markov random field MAP estimation. In *International Conference on Machine Learning.* 737–744.

[36] Pierre Rust, Gauthier Picard, and Fano Ramparany. 2016. Using Message-Passing DCOP Algorithms to Solve Energy-Efficient Smart Environment Configuration Problems. In *International Joint Conference on Artificial Intelligence.* 468–474.

[37] David Sontag. 2010. *Approximate Inference in Graphical Models using LP Relaxations.* Ph.D. Dissertation. Massachusetts Institute of Technology.

[38] David Sontag, Talya Meltzer, Amir Globerson, Tommi Jaakkola, and Yair Weiss. 2008. Tightening LP Relaxations for MAP using Message Passing. In *International Conference on Uncertainty in Artificial Intelligence.* 503–510.

[39] Ruben Stranders, Alessandro Farinelli, Alex Rogers, and Nicholas R. Jennings. 2009. Decentralised coordination of continuously valued control parameters using the Max-Sum algorithm. In *International Conference on Autonomous Agents and Multiagent Systems.* 610–608.

[40] Marc Toussaint and Amos J. Storkey. 2006. Probabilistic inference for solving discrete and continuous state Markov Decision Processes. In *International Conference on Machine Learning.* 945–952.

[41] Peter R. Wurman, Raffaello D'Andrea, and Mick Mountz. 2007. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. In *Innovative Applications of Artificial Intelligence (IAAI).* 1752–1759.

[42] Harel Yedidsion, Roie Zivan, and Alessandro Farinelli. 2018. Applying max-sum to teams of mobile sensing agents. *Eng. Appl. of AI* 71 (2018), 87–99.

[43] William Yeoh, Ariel Felner, and Sven Koenig. 2010. BnB-ADOPT: An Asynchronous Branch-and-Bound DCOP Algorithm. *Journal of Artificial Intelligence Research* 38 (2010), 85–133.

[44] Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. 1998. The Distributed Constraint Satisfaction Problem: Formalization and Algorithms. *IEEE Transactions on Knowledge and Data Engineering* 10 (1998), 673–685.

[45] Weixiong Zhang, Zhao Xing, Guandong Wang, and Lars Wittenburg. 2003. An analysis and application of distributed constraint satisfaction and optimization algorithms in sensor networks. In *International Conference on Autonomous Agents and Multiagent Systems.* 185–192.

[46] Roie Zivan, Steven Okamoto, and Hilla Peled. 2014. Explorative anytime local search for distributed constraint optimization. *Artificial Intelligence* 212, 0 (2014), 1 – 26.