Adaptive Price Update in Distributed Lagrangian Relaxation Protocol

Katsutoshi Hirayama
Kobe University
5-1-1 Fukaeminami-machi,
Higashinada-ku,
Kobe 658-0022, Japan
hirayama@maritime.kobe-u.ac.jp

Toshihiro Matsui
Nagoya Institute of Technology
Gokiso-cho, Showa-ku
Nagoya 466-8555, Japan
matsui.t@nitech.ac.jp

Makoto Yokoo
Kyushu University
744 Motooka, Nishi-ku
Fukuoka 819-0395, Japan
yokoo@is.kyushu-u.ac.jp

ABSTRACT

Distributed Lagrangian Relaxation Protocol (DisLRP) has been proposed to solve a distributed combinatorial maximization problem called the *Generalized Mutual Assignment Problem* (GMAP). In DisLRP, when updating *Lagrange multipliers* (*prices*) of goods, the agents basically control their *step length*, which determines the degree of update, by a static rule. A merit of this updating rule is that since it is static, it is easy to implement even without a central control. Furthermore, if we choose this static rule appropriately, we have observed empirically that DisLRP converges to a state providing a good upper bound. However, it must be difficult to devise such a good static rule for updating step length since it naturally depends on problem instances to be solved. On the other hand, in a centralized context, the Lagrangian relaxation approach has conventionally computed step length by exploiting the least upper bound obtained during the search and a lower bound obtained through preprocessing. In this paper, we achieve this approach in a distributed environment where no central control exists and name the resultant protocol *Adaptive DisLRP* (ADisLRP). The key ideas of this new protocol are to 1) compute global information with a spanning tree, 2) update step length simultaneously with a synchronization protocol, and 3) estimate lower bounds during the search. We also show the robustness of ADisLRP through experiments where we compared ADisLRP with the previous protocols on the critically hard benchmark instances.

Categories and Subject Descriptors

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

General Terms

Algorithms

Keywords

Distributed Combinatorial Optimization, Generalized Assignment, Lagrangian Relaxation, Subgradient Optimization, Spanning Tree, Synchronization

Cite as: Adaptive Price Update in Distributed Lagrangian Relaxation Protocol, Katsutoshi Hirayama, Toshihiro Matsui, Makoto Yokoo, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009),* Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 1033–1040

1. INTRODUCTION

Distributed combinatorial optimization deals with combinatorial optimization problems in which multiple agents are involved. A goal of this problem is to find a *global optimal solution* on the assumption that each individual agent works cooperatively. There has been high demand for distributed combinatorial optimization in various fields, such as resource scheduling [13], supply-chain management [19], and multi-robot coordination [4, 8]. Some researchers have also tried to develop protocols for general problem formulations, such as the non-linear programming problem [1] and the constraint optimization problem [15, 16, 17, 21, 22, 24, 25].

Recently, the *Generalized Mutual Assignment Problem* (GMAP) has been proposed as a new formalism for distributed task/goods assignment problems [11]. GMAP is a distributed combinatorial maximization problem where the agents, each having some goods, try to exchange their goods optimally while satisfying their individual resource constraints. Assuming that the recipient agents have the right to make decisions, GMAP may be otherwise stated as the *resource-constrained distributed set-partitioning problem*.

GMAP is an extension of the *Generalized Assignment Problem* known as GAP in the Operations Research community [3, 18, 20, 23]. The problem of finding an optimal solution to GAP/GMAP is NP-hard and furthermore, the problem of judging the existence of a feasible solution to it is also NP-complete.

A series of peer-to-peer communication protocols, which are generally named the Distributed Lagrangian Relaxation Protocol (DisLRP) [10, 11, 12], have been recently proposed to solve GMAP. DisLRP exploits a technique of *Lagrangian decomposition* [14] to translate GMAP into a set of 0-1 knapsack problems owned by the individual agents. Each agent solves its own 0-1 knapsack problem to find the most profitable combination of related goods that satisfies its resource constraint. We should point out here that, in the 0-1 knapsack problem of an agent, the profit of any good is computed by subtracting its current *price* (*Lagrange multiplier*) from its real profit to this agent.

The behavior of the agents in DisLRP is outlined as follows. Starting with some initial prices of the goods, the agents first solve their 0-1 knapsack problems concurrently using an exact solution algorithm. Note that the 0-1 knapsack problem itself is NP-hard, but fortunately, it is said to be an "easier hard" problem [5] and there exist practically efficient solvers in the literature. Then, the agents inform their respective *neighbors* of their solutions indicating which goods they tentatively selected. Next, after collecting solutions from all of their neighbors, the agents increase the prices of goods that are selected by more than one agent while they decrease the prices of goods that are not selected by any agent. On

the other hand, they keep the same prices of those that are selected by exactly one agent. With these new prices of the goods, the agents again solve their new 0-1 knapsack problems concurrently. These are one *round* of computation and communication performed by the agents. Generally, the agents repeat these rounds until they reach a stable state, where every good is selected by exactly one agent (equivalently, where the agents find a proper set-partition of the goods). If the agents follow a valid scheme to update prices, such a stable state coincides with a global optimal solution to GMAP.

Previously, when updating prices of goods, the agents in DisLRP basically control their *step length*, which determines the degree of update, by a static rule. A merit of this updating rule is that since it is static, it is easy to implement even without a central control. Furthermore, if we choose this static rule appropriately, we have observed empirically that DisLRP converges to a state providing a good upper bound. However, it must be difficult to devise such a good static rule for updating step length since it naturally depends on problem instances to be solved.

On the other hand, in a centralized context, the Lagrangian relaxation approach has conventionally computed step length by exploiting the least upper bound obtained during the search and a lower bound obtained through preprocessing [23]. In this paper, we achieve this approach in a distributed environment where no central control exists and name the resultant protocol *Adaptive DisLRP* (ADisLRP). The key ideas of this new protocol are as follows.

- **Compute global information with a spanning tree.** The agents in ADisLRP have to compute the least upper bound on the optimal value of GMAP while performing the usual process of DisLRP. However, the least upper bound is global information because an upper bound is, at any round, the total sum of all the optimal values of 0-1 knapsack problems over the agents. Furthermore, it is also dynamic information because different upper bounds are obtained over the rounds as the agents update the prices of goods. Therefore, to aggregate such dynamic and global information without any central control, the agents in ADisLRP use a *spanning tree* as with the protocol proposed in [10].

- **Update step length simultaneously with a synchronization protocol.** When finding an upper bound that is smaller than the current least upper bound, an agent will update its step length with this new least upper bound. We should emphasize here that all agents must renew their step length simultaneously. Otherwise, some agents might update their price of the same good with different step length and, as a result, their views on that price would become inconsistent. This inconsistency could be fatal to our scheme of making the agents find the least upper bound. Therefore, in ADisLRP, we incorporate a *synchronization protocol* over a spanning tree to make the agents update their step length simultaneously.

- **Estimate lower bounds during the search.** In computing step length, the centralized Lagrangian relaxation method also exploits a lower bound, which is usually obtained through preprocessing. Namely, it starts with an off-line heuristic method, such as a greedy algorithm or a local search algorithm, to find a feasible solution with a lower bound followed by the main procedure, in which this lower bound is consistently used to compute step length [23]. However, finding a feasible solution is as hard as finding an optimal solution in GAP/GMAP, and moreover, it is more desirable that a protocol for a distributed problem be comprised of one-stage procedures. Therefore, in ADisLRP, we combine an on-line protocol that can efficiently estimate a lower bound on the optimal value of GMAP.

The remainder of this paper is organized as follows. We first provide the formulation of GMAP in Section 2 and then describe DisLRP in Section 3. Next, we detail our key ideas about ADisLRP in Section 4 followed by our experimental results on the critically hard benchmark instances, showing the robustness of ADisLRP in Section 5. Finally, we conclude this paper and point out some future work in Section 6.

2. GENERALIZED MUTUAL ASSIGNMENT PROBLEM

GMAP is a distributed combinatorial maximization problem in which the agents, each having some goods, try to exchange their goods optimally while satisfying their individual resource constraints. It can be viewed as a distributed version of GAP, which is formulated as the following Integer Programming (IP) problem.

$$\begin{aligned}
\mathcal{GAP} \quad & (\text{decide } x_{kj},\ \forall k \in A,\ \forall j \in J): \\
\max. \quad & \sum_{k\in A}\sum_{j\in J} p_{kj}x_{kj} && (1)\\
\text{s. t.} \quad & \sum_{k\in A} x_{kj} = 1,\ \ \forall j \in J, && (2)\\
& \sum_{j\in J} w_{kj}x_{kj} \le c_k,\ \ \forall k \in A, && (3)\\
& x_{kj} \in \{0,1\},\ \forall k \in A,\ \forall j \in J, && (4)
\end{aligned}$$

where $A = \{1, ..., m\}$ is a set of agents; $J = \{1, ..., n\}$ is a set of goods; p_{kj} and w_{kj} are the profit and the amount of required resource, respectively, when agent k obtains good j; c_k is the capacity (amount of available resource) of agent k. x_{kj} is a decision variable whose value is set to 1 when agent k obtains good j and 0 otherwise. The objective is to find the most profitable assignment of n goods to m agents such that (2) every good is assigned to exactly one agent, which will be referred to as the *assignment constraints*, and (3) the assignment satisfies all of the resource constraints imposed on individual agents, which will be referred to as the *knapsack constraints*. GAP is NP-hard; furthermore, the problem of judging the existence of a feasible solution to GAP is NP-complete.

The *Lagrangian relaxation problem*, denoted as $\mathcal{LGAP}(\mu)$, is obtained by dualizing the assignment constraints (2) of $\mathcal{GAP}$ as follows [5, 23].

$$\begin{aligned}
\mathcal{LGAP}(\mu) \quad & (\text{decide } x_{kj},\ \forall k \in A,\ \forall j \in J): \\
\max. \quad & \sum_{k\in A}\sum_{j\in J} p_{kj}x_{kj} + \sum_{j\in J}\mu_j\left(1 - \sum_{k\in A} x_{kj}\right) && (5)\\
\text{s. t.} \quad & \sum_{j\in J} w_{kj}x_{kj} \le c_k,\ \forall k \in A, && (6)\\
& x_{kj} \in \{0,1\},\ \forall k \in A,\ \forall j \in J. && (7)
\end{aligned}$$

where μ_j is a real-valued parameter called a *Lagrange multiplier* (*price*) for good j and the vector $\mu = (\mu_1\, \mu_2\, \ldots\, \mu_n)$ is called a *Lagrange multiplier vector* (*price vector*). It is known that for any value of μ, the optimal value of $\mathcal{LGAP}(\mu)$ provides an upper bound on the optimal value of $\mathcal{GAP}$.

Since the objective (5) is additive over the agents and the constraints (6) are separable over the agents, this maximization can be achieved by each agent k solving the following subproblem [14]:

$$\begin{aligned} \mathcal{LGMP}_k(\mu) \quad & (\text{decide } x_{kj}, \forall j \in R_k) : \\ \text{max.} \quad & \sum_{j \in R_k} p_{kj} x_{kj} + \sum_{j \in R_k} \mu_j \left(\frac{1}{|S_j|} - x_{kj} \right) \\ \text{s. t.} \quad & \sum_{j \in R_k} w_{kj} x_{kj} \leq c_k, \\ & x_{kj} \in \{0, 1\}, \ \forall j \in R_k, \end{aligned}$$

where R_k is a set of goods that may be assigned to agent k and S_j is a set of agents to whom good j may be assigned. Without loss of generality, we can assume $S_j \neq \emptyset$ (i.e., $|S_j|$ is not equal to zero).

To solve GMAP, without gathering all information in one place, distributed solution is possible by exploiting the following properties on the relation between the decomposed subproblems and the global problem [11].

PROPOSITION 1. *For any value of μ, the total sum of the optimal values of $\{\mathcal{LGMP}_k(\mu) | \, k \in A\}$ provides an upper bound on the optimal value of $\mathcal{GAP}$.*

PROPOSITION 2. *For some value of μ, if all of the optimal solutions to $\{\mathcal{LGMP}_k(\mu) | \, k \in A\}$ satisfy the assignment constraints (2) of $\mathcal{GAP}$, then these optimal solutions constitute an optimal solution to $\mathcal{GAP}$.*

3. DISTRIBUTED LAGRANGIAN RELAXATION PROTOCOL

We should notice that the subproblem $\mathcal{LGMP}_k(\mu)$ of agent k is equivalent to the 0-1 knapsack problem. Namely, this problem implies that each good j in R_k has $(p_{kj} - \mu_j)$ as its profit and w_{kj} as its amount of required resource and, among these goods, the agent k must select the most profitable subset that would fit into a knapsack of capacity c_k. We will hereafter use the term *knapsack profit* for $(p_{kj} - \mu_j)$ and *GMAP profit* for p_{kj}, if the meaning of profit is not clear from the context. The agents in DisLRP repeatedly solve their 0-1 knapsack problems while changing the value of μ [11]. The overall behavior of the agents can be summarized as follows.

(Stage 1) The agents set a counter t to zero and initialize their price vector $\mu^{(0)}$ as $(0 \ldots 0)$.

(Stage 2) Under a current value $\mu^{(t)}$ for the price vector, every agent k solves its own 0-1 knapsack problem $\mathcal{LGMP}_k(\mu^{(t)})$ using an exact solution algorithm. Note that the 0-1 knapsack problem itself is NP-hard, but fortunately, it is said to be an "easier hard" problem [5] and there exist practically efficient solvers in the literature. Then, it sends the optimal solution to its own *neighbors*. Neighbors are a group of agents who share interests in the same good. Formally, neighbors of agent k are a union of agents (except for k) having decision variables that appear in the assignment constraint on good j in R_k. Namely, it is denoted by $\bigcup_{j \in R_k} S_j \setminus \{k\}$.

(Stage 3) After receiving these optimal solutions from all of its neighbors, every agent k checks whether they satisfy the assignment constraints on the related goods. If the assignment constraints on all of the goods are satisfied, the agents can stop because the current optimal solutions constitute an optimal solution of $\mathcal{GAP}$ according to Proposition 2. Note that each agent will come to know this fact later by using a spanning tree described in the next section. Otherwise, for each good j whose assignment constraint is not satisfied, the involved agents (agents in S_j) simultaneously update its price from $\mu_j^{(t)}$ to $\mu_j^{(t+1)}$ using the *subgradient optimization method* [23]. After that, the agents increase the counter t by one and go back to Stage 2.

The agents, after initializing their counter and price vector, repeat Stages 2 and 3 until each agent comes to know the fact that they have reached an optimal solution of $\mathcal{GAP}$. The counter t represents the number of times the agents perform Stages 2 and 3. We view this one series of execution over Stages 2 and 3 as a unit and call it a *round*.

At Stage 3, the subgradient optimization method is applied to update the price of a good. This is a typical method to update prices in the Lagrangian relaxation approach. More specifically, an agent k first computes a *subgradient* $g_j^{(t)}$ for each good j in R_k by

$$g_j^{(t)} = 1 - \sum_{i \in S_j} x_{ij}, \tag{8}$$

based on the optimal solutions received from its neighbors. The subgradient $g_j^{(t)}$ indicates the gap between the number of agents required for j (one in this case) and the number of agents that currently select it. Then, with this subgradient $g_j^{(t)}$ along with *step length* $l^{(t)} (> 0)$, which determines the degree of update, the agent k updates the price of good j such that

$$\mu_j^{(t+1)} \leftarrow \mu_j^{(t)} - l^{(t)} \cdot g_j^{(t)}. \tag{9}$$

Intuitively, by this rule, the price of good j will increase when more than one agent currently select good j, while it will decrease when no agent currently selects good j. This dynamics of Lagrange multiplier μ_j implies the price of good j and that is why we call it such.

Note that, in DisLRP, the price information on good j is not under central control. It is rather distributed among the involved agents (agents in S_j), who generally try to keep its copies identical all the time. Therefore, when introducing a sophisticated price-updating scheme into DisLRP, we must devise some mechanism to meet this identical constraint among distributed prices.

4. ADAPTIVE DISLRP

Previously, when updating prices of goods with (9), the agents in DisLRP basically control their step length $l^{(t)}$ by a static rule. In [11], the author suggested that starting with some initial value $l^{(0)}$, it decays at a constant rate r $(0 < r \leq 1)$, meaning $l^{(t+1)} \leftarrow r l^{(t)}$. A merit of this updating rule is that since it is static, it is easy to implement even without a central control. Furthermore, if we choose this static rule appropriately, we have observed empirically that DisLRP converges to a state providing a good upper bound. However, it must be difficult to devise such a good static rule for updating step length since it naturally depends on problem instances to be solved.

On the other hand, in a centralized context, the Lagrangian relaxation approach has conventionally computed step length at round t, $l^{(t)}$, by using the following formula [23]:

$$l^{(t)} = \pi^{(t)} \cdot \frac{\min_{s \in \{0, \ldots, t\}} ub^{(s)} - lb}{\sum_{j \in J} \{g_j^{(t)}\}^2}, \tag{10}$$

where $ub^{(s)}$ is an upper bound found at round s and $\min_{s \in \{0, \ldots, t\}} ub^{(s)}$ is the least upper bound found by round t; lb is a lower bound

usually obtained through preprocessing; $g_j^{(t)}$ is a subgradient for good j computed by (8). On the other hand, $\pi^{(t)}$ is a positive scalar parameter taking 2 as its initial value and being reduced by half when the least upper bound has not been updated during some specified interval of rounds. Thus, in the centralized approach, step length $l^{(t)}$ is computed at each round t by exploiting information on a problem instance, some of which can be computed off-line (lb) or by a static rule ($\pi^{(t)}$), but the others must be computed on-line ($ub^{(s)}$ and $g_j^{(t)}$).

In this paper, we achieve this approach in a distributed environment where no central control exists and name the resultant protocol *Adaptive DisLRP* (ADisLRP). This section describes the key ideas of this new protocol.

4.1 Computing Global Information

To compute step length with (10), the agents need to know an upper bound $ub^{(t)}$ of any round t. In GMAP, this bound is global information since it must be, as claimed by Proposition 1, computed by taking the total sum of the optimal values of $\{\mathcal{LGMP}_k(\mu^{(t)})|\, k \in A\}$. Moreover, the square sum of $g_j^{(t)}$ over the goods, a denominator in (10), is also global information. In a distributed environment without central control, this global information of round t must be computed by using some explicit protocol among the agents. ADisLRP incorporates the CollectTree protocol, which was also used in DisLRP_U [10], to compute this global information.

The CollectTree protocol is a typical distributed data collection algorithm. It assumes a *spanning tree* has been constructed over the graph whose nodes are agents and edges are communication links. A spanning tree of a graph is a sub-graph consisting of the same set of nodes, but only a subset of edges such that any pair of nodes is connected through exactly one path. Since a spanning tree is the basis for many distributed systems, a lot of efficient distributed algorithms have been proposed to construct various types of spanning trees [2, 6, 7]. The CollectTree protocol can employ any spanning tree among those constructed by existing distributed algorithms.

Here we show the outline of CollectTree used in ADisLRP. At some round t, agent k creates a list $GI_k^{(t)}$ in which it keeps its local information of this round, denoted by $I_k^{(t)}$ (which represents a piece of $ub^{(t)}$ and a piece of the square sum of $g_j^{(t)}$), and sends this local information (as well as an optimal solution found at Stage 2 in Section 3) only to its *spanning-tree neighbors*, who are neighboring agents connected through spanning-tree links. A spanning-tree neighbor, k', who received this message will first append the received piece of local information to its list $GI_{k'}^{(t)}$ collecting those pieces of information of round t, and then will relay this message at the next round $t+1$ to its spanning-tree neighbors except k. In this manner, the agents in ADisLRP diffuse their local information of round t over a spanning tree while they proceed with their usual process of the subsequent rounds. On the other hand, when an agent has no information of round t to be sent/relayed, it will send a special character $\#^{(t)}$, meaning transmission completed, to that spanning-tree neighbor. By doing this, each agent k will receive $\#^{(t)}$ from all of its spanning-tree neighbors at a certain round in the future. At this point, the agent k assures itself that all local information at round t has been gathered into its list $GI_k^{(t)}$, and therefore it can compute global information with this list.

In order to get the upper bound $ub^{(t)}$ at round t, each agent k must diffuse the optimal value of $\mathcal{LGMP}_k(\mu^{(t)})$ as its local information $I_k^{(t)}$ and, after receiving $\#^{(t)}$ from all of its spanning-tree neighbors, compute the total sum over the elements of $GI_k^{(t)}$. On the other hand, in order to get the square sum of $g_j^{(t)}$ over the goods, each agent k must diffuse $\sum_{j\in R_k}\{g_j^{(t)}\}^2/|S_j|$ as its local information and compute their total sum in the same way. We should point out that if this square sum of $g_j^{(t)}$ over the goods proves to be zero, each agent comes to know the fact that a global optimal solution of $\mathcal{GAP}$ was found at round t. ADisLRP makes use of this as a termination condition.

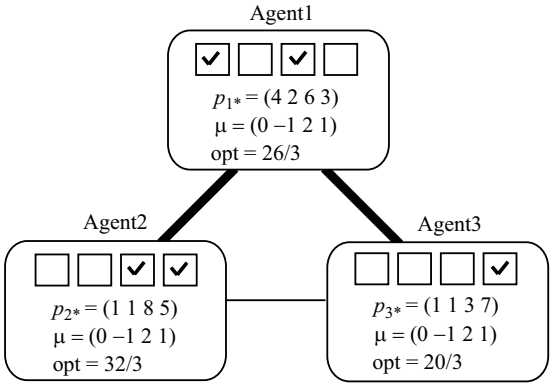

Figure 1: Snapshot at round t

4.2 Simultaneous Update of Step Length

With this method on any type of spanning tree, it will take at least two more rounds for each agent to know global information of round t because any agent spends at least one round to collect its neighbors' local information of round t and another round to collect their $\#^{(t)}$ messages. We should note that the agents know this global information at different rounds. For example, Figure 1 shows a GMAP instance with four goods and three agents. An edge indicates a neighborhood relation among agents and a thick edge indicates a spanning-tree link. We assume that, at a round t, agent 1 selects goods 1 and 3 as its optimal solution, whose optimal value is 26/3; agent 2, on the other hand, does goods 3 and 4, whose optimal value is 32/3; agent 3 does only good 4, whose optimal value is 20/3. The upper bound $ub^{(t)}$ of this round is actually 26 (=(26+32+20)/3), but when the agents collect this local information using the protocol described in Section 4.1, agent 1 will know this value at round $t+2$ while agents 2 and 3 will know it at round $t+3$.

As illustrated in this example, time delays in knowing global information are generally different among the agents. Therefore, if each agent changed its step length with (10) immediately after knowing the latest global information, agents would update the price of some good j by using (9) with different step length and, as a result, the (distributed) price of j would not become identical among the involved agents. Due to these inconsistent views on the price, the agents might fail to get a valid upper bound after that because Proposition 1 no longer holds for such inconsistent prices. Therefore, in ADisLRP, we incorporate a *synchronization protocol* over a spanning tree to make an agent wait for all the other agents to know global information and make the agents update prices with the same step length simultaneously.

We now outline this synchronization protocol, which is divided into three phases. In the first phase, when sending/relaying local information over a spanning tree, an agent adds a *hop count* to it that represents how many times it has been transmitted so far over a spanning tree. Each agent also puts this hop count into the list that keeps collected local information of a certain round. When all

local information of a certain round is collected, an agent k can identify the maximal hop counts, denoted by $MyMaxHop_k$, which represents the number of hops required to reach from k to the furtherest agent on the spanning tree. For example, in Figure 1, agent 1 will eventually come to know $MyMaxHop_1 = 1$ through this first phase of this protocol. Similarly, agents 2 and 3 will also come to know $MyMaxHop_2 = 2$ and $MyMaxHop_3 = 2$, respectively.

In the second phase, an agent k also diffuses this $MyMaxHop_k$ over a spanning tree using the above-mentioned messages for collecting pieces of global information. By this, each agent will eventually come to know the maximal hop counts of all agents and be able to identify its maximal value $MaxHop$ over those maximal hop counts. Obviously, $MaxHop$ is inherent in a spanning tree and indicates the maximal hop counts on a spanning tree over those of all possible pairs of agents. We can see $MaxHop = 2$ for a spanning tree in Figure 1. Note that $MaxHop$ is also global information, which may be known by the agents at different rounds.

The agents must go on to know the fact that every agent knows the value of $MaxHop$ (that is, the fact that it becomes common knowledge). So, in the third phase, an agent also adds this $MaxHop$ into the messages to be spread along a spanning tree. If an agent has not yet been aware of $MaxHop$, it diffuses any symbol instead, say 0, indicating that it does not know $MaxHop$ at that time. At some round in the future, each agent will eventually know the fact that all of the agents have become aware of $MaxHop$ (that is, the fact that $MaxHop$ becomes common knowledge). Note also that the agents may know this fact at different rounds.

The key rule of this synchronization protocol is as follows.

- Assume that $MaxHop$ was certainly common knowledge at round t and an agent k knows this fact at round t_k ($> t$), then the agent k must update its step length using global information of round t exactly at a round of $(t_k + MaxHop - MyMaxHop_k)$.

Consequently, at this round, the agents in ADisLRP simultaneously update their step length with exactly the same global information of round t. In Figure 1, agent 1, for example, will come to know an upper bound of 26 at round $t + 2$, but will change its step length using this upper bound at round $t + 3$, computed by $t + 2 + MaxHop - MyMaxHop_1$. On the other hand, both agents 2 and 3 will come to know the upper bound at round $t + 3$ and immediately change their step length using this upper bound, since both $t + 3 + MaxHop - MyMaxHop_2$ and $t + 3 + MaxHop - MyMaxHop_3$ are $t + 3$.

Note that, in the above, once an agent k identifies $MyMaxHop_k$ and $MaxHop$ after going through those three phases, it does not have to go through them again because these values are static. On the other hand, in the beginning where these three phases are performed, the agents cannot change their step length simultaneously because each agent k may not know $MyMaxHop_k$ or $MaxHop$. Thus, before accomplishing these three phases, the agents in ADisLRP proceed with a common default value, one, for their step length.

4.3 On-line Estimation of Lower Bounds

In updating step length with (10), the centralized Lagrangian relaxation approach also exploits a lower bound lb, which is usually obtained through preprocessing. Namely, it starts with an off-line heuristic method, such as a greedy algorithm or a local search algorithm, to find a feasible solution with a lower bound followed by the main procedure, in which this lower bound is consistently used to compute step length [23]. However, finding a feasible solution is as hard as finding an optimal solution in GAP/GMAP, and moreover, it is more desirable that a protocol for a distributed problem be comprised of one-stage procedures. Therefore, in ADisLRP, we combine an on-line protocol that can efficiently estimate a lower bound on the optimal value of GMAP.

We will first provide an overview of this protocol. In addition to the local information on $ub^{(t)}$ and the square sum of $g_j^{(t)}$, the agents collect the individual optimal solutions of round t using a spanning tree. With these collected optimal solutions, an agent computes an estimated lower bound $\hat{lb}^{(t)}$ by using the method detailed below and, when updating step length with (10), uses the greatest estimated lower bound among those computed up to this round. We should point out here that with this method an agent may overestimate the bounds and, as a result, the greatest estimated lower bound can exceed the actual optimal value. An agent will become aware of such overestimation when an estimated lower bound gets larger than the current least upper bound. In order to address this issue, each agent keeps a series of the greatest estimated lower bounds in a stack and, when becoming aware that the greatest bound is overestimated, retrieves from the stack the latest bound that is smaller than the current least upper bound. In our current implementation, once this retrieval occurs, an agent stops estimating the bounds and sticks to this retrieved bound until finding this is again overestimated.

Next, we will detail the method for estimating a lower bound. With the optimal solutions of round t collected by the CollectTree protocol, an agent k computes an estimated lower bound $\hat{lb}^{(t)}$ as follows.

(Step 1) Divide a set R_k of goods into:

J_0: a set of goods that were not selected by any agent at round t,

J_1: a set of goods that were selected by exactly one agent at round t,

J_2: a set of goods that were selected by more than one agent at round t.

For example, we can see $J_0 = \{2\}$, $J_1 = \{1\}$, and $J_2 = \{3, 4\}$ for any agent in Figure 1.

(Step 2) Create a partial assignment $PA^{(t)}$ such that:

- each good in J_1 is assigned to the agent who selects it,
- each good in J_2 is assigned, among those who select it, to the agent who gives the maximal GMAP profit to this good[1],

and then compute the objective value $lb_{PA}^{(t)}$ of $PA^{(t)}$ using the GMAP profits collected together with the optimal knapsack solutions. For example, in Figure 1, each agent creates its $PA^{(t)}$ such that good 1 is assigned to agent 1, good 3 to agent 2, and good 4 to agent 3. The objective value $lb_{PA}^{(t)}$ of this partial assignment is $19\,(= 4 + 8 + 7)$. Clearly, $PA^{(t)}$ satisfies all of the knapsack constraints, and the assignment constraints on the goods in both J_1 and J_2 (but not in J_0).

(Step 3) Guess a lower bound by extending $PA^{(t)}$ over J_0 as follows.

If $J_0 = \emptyset$, then $PA^{(t)}$ is a feasible assignment, a full assignment that satisfies all of the constraints. Thus, its objective

[1] In this on-line estimation protocol, we assume that the agents will reveal their GMAP profits of selected goods.

value $lb_{PA}^{(t)}$ provides a valid lower bound. Furthermore, if this lower bound is equal to the currently-known least upper bound, an agent can prove $PA^{(t)}$ to be a global optimal solution.

On the other hand, if $J_0 \neq \emptyset$, then it is generally not clear whether $PA^{(t)}$ can be extended to a feasible assignment. However, assuming that the search space under $PA^{(t)}$ includes a feasible assignment, its objective value should be not less than

$$lb_{PA}^{(t)} + \sum_{j \in J_0} \min_{k \in S_j} p_{kj}, \tag{11}$$

where the first term is the objective value of $PA^{(t)}$; the second term is computed by identifying, for each good j in J_0, the minimal GMAP profit over those provided by its involved agents and then taking the sum of all these minimal profits over J_0. In this protocol, an agent estimates a lower bound at (11) (computes $\hat{lb}^{(t)}$ by (11)) even if the search space under $PA^{(t)}$ does not include a feasible assignment. In Figure 1, since the formula (11) is $19 + \min\{2, 1, 1\}$, we can see the estimated lower bound $\hat{lb}^{(t)}$ to be 20.

Obviously, an agent may overestimate a lower bound when the search space under $PA^{(t)}$ does not include a feasible assignment. We should point out that the problem of deciding if this search space includes a feasible assignment is NP-complete because this problem is equivalent to finding a feasible solution to GAP whose goal is to assign J_0 optimally to the agents with remaining capacities. Our on-line estimation, on the other hand, is very efficient and the issue of overestimation can be easily compensated with the above retrieval method.

5. EXPERIMENTAL EVALUATION

We made experiments to show the robustness of the proposed protocol. In these experiments, we made the agents solve the GAP benchmark instances[2]. It must be pointed out that these instances are so hard that, for some of them, even centralized algorithms have not yet proved their optimal solutions. Note that since these instances are a minimization problem, we translated each of them into an equivalent maximization problem by multiplying the costs by -1.

Our experiments were conducted on the simulator, written in JAVA, which simulates concurrent activities of multiple agents. As a problem solver by which an agent solves the local knapsack problem, we used the commercial solver, ILOG CPLEX11.0.

5.1 Comparison with the Previous Protocols

First, we compared ADisLRP and the previous protocol DisLRP_U [10] on the upper bounds to which they converge. Note that, as a spanning tree for both protocols, we used the *breadth-first search tree* illustrated in Figure 2.

In updating step length with (10), ADisLRP also uses a positive scalar parameter $\pi^{(t)}$. As with the centralized Lagrangian relaxation approach, $\pi^{(t)}$ takes 2 as its initial value and is reduced by half when the least upper bound has not been updated during some specified interval of rounds. By doing this, $\pi^{(t)}$ is gradually reduced as rounds proceed and eventually becomes almost close to zero. In our experiments, we set this interval to be 100 rounds and terminated a run when $\pi^{(t)}$ got smaller than a value of 10^{-6}. We measured the least upper bound at this terminated round.

[2]http://www.al.cm.is.nagoya-u.ac.jp/~yagiura/gap/

Table 1: ADisLRP vs. DisLRP_U on Benchmark Instances

Instance	Opt[a]	Protocol	Round	UB	EstLB
c10200 in gapc	-2806	ADisLRP	4542	$\mathbf{-2804}$	-2939
		DisLRP_U^1	10000	$\mathbf{-2804}$	–
		DisLRP_U^2	10000	-2799	–
		DisLRP_U^3	3208	$\mathbf{-2804}$	–
c20200 in gapc	-2391	ADisLRP	5327	$\mathbf{-2391}$	-2503
		DisLRP_U^1	10000	-2390	–
		DisLRP_U^2	10000	-2384	–
		DisLRP_U^3	3804	$\mathbf{-2391}$	–
c10200 in gapd	-12432^*	ADisLRP	4405	$\mathbf{-12426}$	-12559
		DisLRP_U^1	10000	-12425	–
		DisLRP_U^2	10000	-12415	–
		DisLRP_U^3	3576	$\mathbf{-12426}$	–
c20200 in gapd	-12241^*	ADisLRP	4422	$\mathbf{-12230}$	-12595
		DisLRP_U^1	10000	-12229	–
		DisLRP_U^2	10000	-12201	–
		DisLRP_U^3	4045	$\mathbf{-12230}$	–
c10200 in gape	-23307^*	ADisLRP	4400	$\mathbf{-23303}$	-24487
		DisLRP_U^1	10000	-23301	–
		DisLRP_U^2	10000	-23299	–
		DisLRP_U^3	3927	$\mathbf{-23303}$	–
c20200 in gape	-22379^*	ADisLRP	4344	$\mathbf{-22378}$	-28070
		DisLRP_U^1	10000	-21979	–
		DisLRP_U^2	10000	-22365	–
		DisLRP_U^3	3748	-22377	–

[a]The figures marked by * are the best known lower bounds instead of the optimal values.

On the other hand, to compare with ADisLRP, we made the following three versions of DisLRP_U:

DisLRP_U^1, where step length $l^{(t)}$ is fixed to $1/|S_j|$, as with the original DisLRP_U in [10];

DisLRP_U^2, where step length $l^{(t)}$ is fixed to 1;

DisLRP_U^3, where step length $l^{(t)}$ is replaced by the aforesaid $\pi^{(t)}$, which is gradually reduced as rounds proceed.

As to both DisLRP_U^1 and DisLRP_U^2, we set a limit of 10000 rounds and measured the least upper bound at this limit round. But, as to DisLRP_U^3, we terminated a run in the same way with ADisLRP (when $\pi^{(t)}$ got smaller than 10^{-6}) and measured the least upper bound at this terminated round.

Due to space constraints, we show only the results on two large instances, c10200 (assigning 200 goods to 10 agents) and c20200 (assigning 200 goods to 20 agents), for each problem class of *gapc*, *gapd*, and *gape* in Table 1. We should point out that these instances are critically hard even for centralized GAP algorithms. For the four instances of gapd and gape in particular, optimal solutions have not been discovered so far even with centralized algorithms. Thus, at the Opt column, showing usually optimal values, in Table 1 we describe the best known lower bound marked with * for these instances. On top of that, for ADisLRP, we show at the EstLB column in Table 1 the greatest estimated lower bounds being computed by the method described in Section 4.3. As indicated in Table 1, ADisLRP and DisLRP_U^3, both of which reduce step length by using a scalar parameter $\pi^{(t)}$, can provide tighter upper bounds consistently. On the other hand, looking at the Round column that shows the rounds at which the protocols are terminated, we can see that DisLRP_U^3 seems to converge a little faster than ADisLRP.

Next, with the other properties remaining the same, we created a new instance, called c**k, from an instance of c** by making all the profits 1000 times and performed the same experiments for these new instances. Since this scale transformation of profits does not change any inherent property of instances, the performance of

Table 2: ADisLRP vs. DisLRP_U on Benchmark Instances with all the profits multiplied by 1000 times

Instance	Opt	Protocol	Round	UB	EstLB
c10200k in gapc	-2806k	ADisLRP	4962	$\mathbf{-2804k}$	-2924k
		DisLRP_U^1	10000	-200k	–
		DisLRP_U^2	10000	-2000k	–
		DisLRP_U^3	10000	-2650k	–
c20200k in gapc	-2391k	ADisLRP	5698	$\mathbf{-2391k}$	-2497k
		DisLRP_U^1	10000	-100k	–
		DisLRP_U^2	10000	-2000k	–
		DisLRP_U^3	10000	-2355k	–
c10200k in gapd	-12432k*	ADisLRP	5190	$\mathbf{-12426k}$	-12544k
		DisLRP_U^1	10000	-200k	–
		DisLRP_U^2	10000	-1840k	–
		DisLRP_U^3	10000	-3089k	–
c20200k in gapd	-12241k*	ADisLRP	5259	$\mathbf{-12230k}$	-12589k
		DisLRP_U^1	10000	-100k	–
		DisLRP_U^2	10000	-1746k	–
		DisLRP_U^3	10000	-2735k	–
c10200k in gape	-23307k*	ADisLRP	4380	$\mathbf{-23303k}$	-23622k
		DisLRP_U^1	10000	-200k	–
		DisLRP_U^2	10000	-1963k	–
		DisLRP_U^3	10000	-3767k	–
c20200k in gape	-22379k*	ADisLRP	5220	$\mathbf{-22377k}$	-28038k
		DisLRP_U^1	10000	-100k	–
		DisLRP_U^2	10000	-1956k	–
		DisLRP_U^3	10000	-3696k	–

protocols should ideally be invariant under that transformation. The results are shown in Table 2. One can see that the performance of all three versions of DisLRP_U deteriorates under this transformation. However, ADisLRP does not since it determines step length with (10), whose scale is also transformed by exploiting the information on the bounds. We may say that ADisLRP is robust since it is effective for the larger range of problem instances.

5.2 Impact of Communication Delay

Since there is no global control in ADisLRP, no agent can be aware of global information at the time it actually comes into being. In order to compute global information, the agents must spend some additional rounds in diffusing their local information over a spanning tree and, as described in Section 4.2, synchronizing their actions. We will refer to these additional rounds as *communication delay*. During this delay period, the agents continue the usual process of rounds, in which they repeat the knapsack problem solving and price updating with known (but actually out-of-date) global information, until they are ready to use the latest global information. Clearly, the larger the communication delay is, the longer the rounds in which the agents use such obsolete global information are. We should remark that the communication delay depends on the topology of a spanning tree. For example, on the same complete graph of five agents in Figure 2, we can see the communication delay to be 3 rounds for the breadth-first search tree (left) and 5 rounds for the depth-first search tree (right).

Our next experiments examine the impact of communication delay on the performance of ADisLRP. In these experiments, we compared ADisLRP with the breadth-first search tree (BFS) and ADisLRP with the depth-first search tree (DFS) on the benchmark problem instances. In addition, we will also report the results of the ideal method (RT) for reference, in which any global information can be computed in real-time (with no communication delay), and the best-known lower bounds are given in advance. The results are shown in Table 3. These results clearly indicate that there is virtually no impact of communication delay on the upper bound found by ADisLRP.

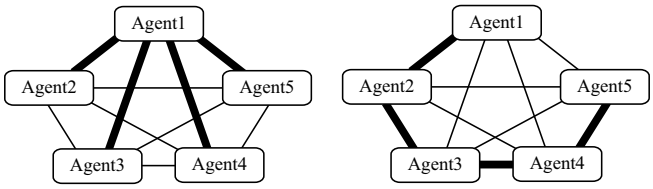

Figure 2: Breadth-first search tree (left) and Depth-first search tree (right)

Table 3: Impact of the Communication Delay

Instance	Opt	STree	Round	UB	EstLB
c10200 in gapc	-2806	BFS	4542	$\mathbf{-2804}$	-2939
		DFS	4606	$\mathbf{-2804}$	-2931
		RT	3268	$\mathbf{-2804}$	-2806
c20200 in gapc	-2391	BFS	5327	$\mathbf{-2391}$	-2503
		DFS	4631	$\mathbf{-2391}$	-2498
		RT	5373	$\mathbf{-2391}$	-2391
c10200 in gapd	-12432^*	BFS	4405	$\mathbf{-12426}$	-12559
		DFS	3791	$\mathbf{-12426}$	-12540
		RT	3157	$\mathbf{-12426}$	-12432
c20200 in gapd	-12241^*	BFS	4422	$\mathbf{-12230}$	-12595
		DFS	4751	$\mathbf{-12230}$	-12620
		RT	4138	$\mathbf{-12230}$	-12241
c10200 in gape	-23307^*	BFS	4400	$\mathbf{-23303}$	-24487
		DFS	4532	$\mathbf{-23303}$	-24450
		RT	2768	$\mathbf{-23303}$	-23307
c20200 in gape	-22379^*	BFS	4344	$\mathbf{-22378}$	-28070
		DFS	3586	$\mathbf{-22378}$	-27549
		RT	3492	-22377	-22379

6. CONCLUSIONS

We have presented a new distributed optimization protocol called Adaptive DisLRP (ADisLRP) to solve GMAP, in which the agents compute step length, the degree of price update, by exploiting global information obtained during the search. Since we did not assume any central control in this protocol, we have introduced three main ideas to create this approach in a distributed environment.

First, in order to make each agent find global information such as the upper bounds on the optimal value of GMAP, we have incorporated a distributed data collection algorithm over a spanning tree, called the CollectTree protocol. This protocol was also used in DisLRP_U [10] for computing the upper bounds. But, in ADisLRP, it has been used to compute not only the upper bounds but also other global information needed for computing step length. Second, for agents to have a consistent view of the price of goods throughout the execution of the protocol, we have incorporated a synchronization protocol, by which the agents involved with the same goods can simultaneously update their price using the same step length. Third, in order to avoid the need to pre-compute a lower bound on the optimal value of GMAP, we have incorporated an on-line protocol that can efficiently estimate the lower bounds.

Our experimental results on the critically hard GAP benchmark instances showed that ADisLRP had a clear advantage over the previous protocol DisLRP_U, since it converged to the states providing tight upper bounds in the larger range of instances. On top of that, they also showed that the communication delay in computing global information had virtually no impact on the performance of the protocol.

On the other hand, we may say that one drawback of ADisLRP is privacy loss [9]. An agent in DisLRP_U reveals the optimal value

of its local problem at every round. However, in ADisLRP, an agent must reveal, in addition to this optimal value, the GMAP profits of selected goods to perform the on-line estimation of lower bounds. Future work may include addressing this privacy issue on ADisLRP.

Although our primary goal in developing ADisLRP was to find tighter upper bounds, one might want to obtain tighter lower bounds (along with feasible solutions) as well. Currently, ADisLRP is certainly able to estimate a lower bound at every round, which is exploited in computing step length. As indicated in our experiments, these estimated lower bounds worked quite well for the protocol to find tight upper bounds. However, those estimated lower bounds are generally not so tight, and furthermore, they do not necessarily come with feasible solutions. Future work may also include incorporating ADisLRP with some distributed *Lagrangian heuristics*, which aims to transform an infeasible solution with a (tight) upper bound into a feasible solution.

7. ACKNOWLEDGMENTS

The authors wish to thank Hiroshi Matsuo for his helpful comments on this work. This work was supported by Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research (B), 19300048.

8. REFERENCES

[1] I. P. Androulakis and G. V. Reklaitis. Approaches to asynchronous decentralized decision making. *Computers and Chemical Engineering*, 23:341–355, 1999.

[2] M. Bui, F. Butelle, and C. Lavault. A distributed algorithm for constructing a minimum diameter spanning tree. *Journal of Parallel and Distributed Computing*, 64:571–577, 2004.

[3] D. G. Cattrysse and L. N. V. Wassenhove. A survey of algorithms for the generalized assignment problem. *European Journal of Operational Research*, 60:260–272, 1992.

[4] M. B. Dias, R. Zlot, N. Kalra, and A. Stentz. Market-based multirobot coordination: a survey and analysis. *Proceedings of the IEEE*, 94(7):1257–1270, 2006.

[5] M. L. Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management Science*, 27(1):1–18, 1981.

[6] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning tree. *ACM Transactions on Programming Languages and Systems*, 5(1):66–77, 1983.

[7] F. C. Gärtner. A survey of self-stabilizing spanning-tree construction algorithms. Technical Report IC/2003/38, Swiss Federal Institute of Technology (EPFL), 2003.

[8] B. P. Gerkey and M. J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research*, 23(9):939–954, 2004.

[9] R. Greenstadt, J. P. Pearce, and M. Tambe. Analysis of privacy loss in distributed constraint optimization. In *Proceedings of the 21th National Conference on Artificial Intelligence (AAAI-2006)*, pages 647–653, 2006.

[10] K. Hirayama. A distributed solution protocol that computes an upper bound for the generalized mutual assignment problem. In *Proceedings of the 7th International Workshop on Distributed Constraint Reasoning*, pages 102–116, 2006.

[11] K. Hirayama. A new approach to distributed task assignment using Lagrangian decomposition and distributed constraint satisfaction. In *Proceedings of the 21th National Conference on Artificial Intelligence (AAAI-2006)*, pages 660–665, 2006.

[12] K. Hirayama. An α-approximation protocol for the generalized mutual assignment problem. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-2007)*, pages 744–749, 2007.

[13] E. Kutanoglu and S. D. Wu. On combinatorial auction and Lagrangian relaxation for distributed resource scheduling. *IIE Transactions*, 31(9):813–826, 1999.

[14] L. S. Lasdon. *Optimization theory for large systems*. Dover, 2002.

[15] R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *Proceedings of the Third International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS-2004)*, pages 438–445, 2004.

[16] T. Matsui, M. Silaghi, K. Hirayama, M. Yokoo, and H. Matsuo. Resource constrained distributed constraint optimization with virtual variables. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-2008)*, pages 120–125, 2008.

[17] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. In *Proceedings of the Second International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS-2003)*, pages 161–168, 2003.

[18] R. M. Nauss. The generalized assignment problem. In J. K. Karlof, editor, *Integer Programming: Theory and Practice*, pages 39–55. CRC Press, 2006.

[19] T. Nishi, M. Konishi, and S. Hasebe. An autonomous decentralized supply chain planning system for multi-stage production processes. *Journal of Intelligent Manufacturing*, 16:259–275, 2005.

[20] I. H. Osman. Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches. *OR Spektrum*, 17:211–225, 1995.

[21] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-2005)*, pages 266–271, 2005.

[22] M.-C. Silaghi and M. Yokoo. Nogood based asynchronous distributed optimization (ADOPTing). In *Proceedings of the Fifth International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS-2006)*, pages 1389–1396, 2006.

[23] M. Yagiura and T. Ibaraki. Generalized assignment problem. In T. F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*, Computer & Information Science Series. Chapman & Hall/CRC, 2006.

[24] W. Yeoh, A. Felner, and S. Koenig. BnB-ADOPT: an asynchronous branch-and-bound DCOP algorithm. In *Proceedings of the Seventh International Conference on Autonomous Agents & Multi-Agent Systems (AAMAS-2008)*, pages 591–598, 2008.

[25] R. Zivan. Anytime local search for distributed constraint optimization. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-2008)*, pages 393–398, 2008.