Distributed Lagrangean Relaxation Protocol for the Generalized Mutual Assignment Problem

Katsutoshi Hirayama

Faculty of Maritime Sciences, Kobe University
5-1-1 Fukaeminami-machi, Higashinada-ku, Kobe 658-0022, JAPAN
Email: hirayama@maritime.kobe-u.ac.jp, URL: http://www.kobe-u.ac.jp/hirayama/

Abstract. The generalized assignment problem (GAP) is a typical NP-hard problem and has been studied for many years mainly in the operations research community. The goal of the GAP is to find an optimal assignment of jobs to agents such that the assignment satisfies all of the resource constraints imposed on individual agents. This work provides a distributed formulation of GAP, the generalized mutual assignment problem (GMAP), to deal with the problems of job assignment in multi-agent systems. We present a distributed lagrangean relaxation protocol that enables agents to simultaneously solve a GMAP instance on peer-to-peer message exchange mechanisms. In the protocol we introduce a parameter that controls the range of "noise" mixed with increment/decrement in a lagrangean multiplier. This parameter can be used to produce quick agreement between the agents on a feasible solution with reasonably good quality. Our experimental results imply that the parameter may also allow us to control tradeoffs between the quality of a solution and the cost of finding it.

1 Introduction

The *generalized assignment problem* (GAP) is a typical combinatorial optimization problem that has been studied for many years mainly in the operations research community. The goal of the GAP is to find an optimal assignment of jobs to agents such that a job is assigned to exactly one agent and the assignment satisfies all of the resource constraints imposed on individual agents. Since the GAP is NP-hard, numerous attempts have been made to develop not only exact methods for finding an optimal solution but also heuristic methods for finding a near optimal solution [7].

This work provides a distributed formulation of GAP, the *generalized mutual assignment problem* (GMAP), to deal with the problems of job assignment in multi-agent systems (MAS). We can view the GAP as a problem where a *coordinator* tries to optimally assign jobs to a set of agents. On the other hand, the GMAP can be considered a problem where a set of coordinators/agents, each of whom has an individual set of jobs, tries to optimally assign their jobs to each other.

The GMAP can be solved by a *centralized method* if all the agents agree to pass on their jobs and related information to a *super coordinator* responsible for a solution to the entire problem. However, a centralized method, even though efficient in many cases, is generally considered inappropriate for MAS problems because it forces agents to reveal private information and imposes extra labor (for example, maintenance of the

super coordinator) on an administrator at a site. Thus, in this work we seek a distributed solution to the GMAP by presenting a *distributed lagrangean relaxation protocol* that enables agents to simultaneously solve a GMAP instance on peer-to-peer message exchange mechanisms.

In distributed problem solving literature, distributed task assignment protocols have been widely studied, and many have their roots in the well-known *contract net protocol* [9]. To the author's knowledge, no studies have ever tried to develop such a protocol under the distributed formulation of GAP. Furthermore, only few attempts have so far been made at a distributed solution of combinatorial optimization problems, with the notable exception of Androulakis and Reklaitis's work on a decentralized solution to the general optimization problem [1] and a series of distributed solution protocols for the constraint optimization problem [2, 6, 4].

The paper is organized as follows. First, in Section 2 we formalize the GMAP as a collection of integer programs sharing decision variables with each other and introduce a lagrangean relaxation problem obtained by dualizing assignment constraints. In Section 3 we provide the key ideas of the distributed lagrangean relaxation protocol including methods for solving primal/dual problems and detecting convergence. We also introduce a parameter to produce quick agreement between the agents on a feasible solution with reasonably good quality. In Section 4 we report our experiments being conducted to see the actual performance of the protocol and finally conclude this work in Section 5.

2 Formalization

2.1 Generalized Mutual Assignment Problem

The GAP can be formulated as the following integer program:

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

where $A = \{1, \ldots, m\}$ is a set of agents, $J = \{1, \ldots, n\}$ is a set of jobs, p_{ij} is the profit of assigning job j to agent i, w_{ij} is the resource requirement for agent i to perform job j, and c_i is the available resource capacity of agent i. Decision variable x_{ij} takes 1 if agent i performs job j, and 0 otherwise. The objective is to maximize the profit of the assignment such that each job is assigned to exactly one agent (*assignment constraints* (2) are satisfied), the total resource requirement of each agent does not exceed the available resource capacity (*knapsack constraints* (3) are satisfied), and each job is assigned or not assigned to an agent (*01 constraints* (4) are satisfied). The maximal value of the

profit is called the *optimal value* and an assignment that provides the optimal value is called an *optimal solution.* The problem of finding an optimal solution to the GAP is known to be NP-hard.

The GMAP consists of a set of agents, each of which (say agent $k \in \{1, \ldots, m\}$) has its own GAP that includes a set of agents A_k and a set of jobs J_k. Obviously, agent k wants all of the jobs in J_k to be assigned to the agents in A_k. If we view $\bigcup_{k=1}^{m} A_k$ as A and $\bigcup_{k=1}^{m} J_k$ as J, we can easily formulate this problem as $\mathcal{GAP}$.

Example 1 There are two transportation companies (agents) 1 and 2. Agent 1 has to deliver some goods from Kobe to Tokyo. We will refer to this job as job 1. On the other hand, agent 2 has to deliver some goods from Tokyo to Kyoto and other goods from Kyoto to Kobe. We will refer to the former as job 2 and the latter as job 3. For each job, an agent is deliberating whether to perform it by himself or to outsource it another agent. We assume that agent 1 has 4 available resources (ex. trucks) while agent 2 has 3 available resources, and that, under the current configuration of these resources, each agent i estimates its resource requirement w_{ij} and its profit p_{ij} when accepting job j like this: for agent 1 $w_{11} = 2,\ p_{11} = 5,\ w_{12} = 2,\ p_{12} = 6$, and $w_{13} = 1,\ p_{13} = 5$; for agent 2 $w_{21} = 2,\ p_{21} = 4,\ w_{22} = 2,\ p_{22} = 2$, and $w_{23} = 2,\ p_{23} = 2$. This example can be described as the following GAP instance.

$$
\begin{aligned}
\mathcal{GAP}\ & (\text{decide } x_{11}, x_{12}, x_{13}, x_{21}, x_{22}, x_{23}) : \\
\max.\ & 5x_{11} + 6x_{12} + 5x_{13} + 4x_{21} + 2x_{22} + 2x_{23} \\
\text{s. t.}\ & x_{11} + x_{21} = 1, \\
& x_{12} + x_{22} = 1, \\
& x_{13} + x_{23} = 1, \\
& 2x_{11} + 2x_{12} + x_{13} \leq 4, \\
& 2x_{21} + 2x_{22} + 2x_{23} \leq 3, \\
& x_{ij} \in \{0, 1\},\ \forall i \in \{1, 2\},\ \forall j \in \{1, 2, 3\}.
\end{aligned}
$$

However, since we seek a distributed solution to the GMAP, this formulation must be decomposed into pieces, each of which is uniquely mapped to an agent. We can obtain one natural decomposition based on the idea that the value of decision variable x_{kj} should be determined by agent k (in other words, agent k has the right to decide whether it will undertake job j), where agent k has

$$
\begin{aligned}
\mathcal{GMP}_k\ & (\text{decide } x_{kj}, \forall j \in R_k) : \\
\max.\ & \sum_{j \in R_k} p_{kj} x_{kj} \\
\text{s. t.}\ & \sum_{i \in S_j} x_{ij} = 1,\ \forall j \in R_k, \\
& \sum_{j \in R_k} w_{kj} x_{kj} \leq c_k, \\
& x_{kj} \in \{0, 1\},\ \forall j \in R_k,
\end{aligned}
$$

and tries to determine the values of x_{kj} ($\forall j \in R_k$). In this formulation, R_k is a set of jobs that may be assigned to agent k and S_j is a set of agents to whom job j may be assigned. The remaining elements are the same as the ones in $\mathcal{GAP}$.

Example 2 The GAP instance of Example 1 can be decomposed into the following pieces:

$$
\begin{aligned}
\mathcal{GMP}_1 \quad & (\text{decide } x_{11}, x_{12}, x_{13}) : \\
\text{max.} \quad & 5x_{11} + 6x_{12} + 5x_{13} \\
\text{s. t.} \quad & x_{11} + x_{21} = 1, \\
& x_{12} + x_{22} = 1, \\
& x_{13} + x_{23} = 1, \\
& 2x_{11} + 2x_{12} + x_{13} \leq 4, \\
& x_{1j} \in \{0, 1\},\ \forall j \in \{1, 2, 3\};
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{GMP}_2 \quad & (\text{decide } x_{21}, x_{22}, x_{23}) : \\
\text{max.} \quad & 4x_{21} + 2x_{22} + 2x_{23} \\
\text{s. t.} \quad & x_{11} + x_{21} = 1, \\
& x_{12} + x_{22} = 1, \\
& x_{13} + x_{23} = 1, \\
& 2x_{21} + 2x_{22} + 2x_{23} \leq 3, \\
& x_{2j} \in \{0, 1\},\ \forall j \in \{1, 2, 3\}.
\end{aligned}
$$

Note that in the assignment constraints, $\mathcal{GMP}_k$ includes decision variables whose values are decided by other agents. Therefore, agent k has to refer to these agents for their values when solving $\mathcal{GMP}_k$.

By *all of the optimal solutions to* $\{\mathcal{GMP}_k \mid k \in A\}$, we mean the state that every agent k finds an optimal solution to $\mathcal{GMP}_k$ that does not violate any constraint. On the relation between all of the optimal solutions to $\{\mathcal{GMP}_k \mid k \in A\}$ and an optimal solution to $\mathcal{GAP}$, we can obtain the following lemma.

Lemma 1 *All of the optimal solutions to* $\{\mathcal{GMP}_k \mid k \in A\}$ *constitute an optimal solution to* $\mathcal{GAP}$.

Proof: This obviously follows because the feasible region of $\mathcal{GAP}$ is equal to the intersection of all the feasible regions of $\{\mathcal{GMP}_k \mid k \in A\}$ and because the objective function of $\mathcal{GAP}$ is equal to the sum of all the objective functions of $\{\mathcal{GMP}_k \mid k \in A\}$. □

It must be noted that this does not indicate that the entire $\mathcal{GAP}$ can be solved by a *divide-and-conquer* method, because an assignment constraint in $\mathcal{GMP}_k$ has the variables derived from other agents.

2.2 Lagrangean Relaxation

By dualizing the assignment constraints (2) of $\mathcal{GAP}$, we obtain the following lagrangean relaxation problem:

$$\begin{aligned}
\mathcal{LGAP}(\mu)\ & (\text{decide } x_{ij},\ \forall i \in A,\ \forall j \in J):\\
\text{max.}\ & \sum_{i\in A}\sum_{j\in J} p_{ij}x_{ij} + \sum_{j\in J}\mu_j(1-\sum_{i\in A} x_{ij})\\
\text{s. t.}\ & \sum_{j\in J} w_{ij}x_{ij} \le c_i,\ \forall i \in A,\\
& x_{ij} \in \{0,1\},\ \forall i \in A,\ \forall j \in J,
\end{aligned}$$

where $\mu = (\mu_1, ..., \mu_n)$ is a *lagrangean multiplier vector* of which element (called a *lagrangean multiplier*) takes a real number ranging over $(-\infty, \infty)$. It is well-known that the optimal value of $\mathcal{LGAP}(\mu)$ provides an upper bound for the optimal value of $\mathcal{GAP}$.

As with the decomposition of the $\mathcal{GAP}$ into $\{\mathcal{GMP}_k \mid k \in A\}$, $\mathcal{LGAP}(\mu)$ can be decomposed into a set of lagrangean relaxation problems where each agent k has

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

Example 3 From Example 1, we obtain the following two lagrangean relaxation problems:

$$\begin{aligned}
\mathcal{LGMP}_1(\mu)\ & (\text{decide } x_{11}, x_{12}, x_{13}):\\
\text{max.}\ & 5x_{11} + 6x_{12} + 5x_{13} + \mu_1(\frac{1}{2} - x_{11}) + \mu_2(\frac{1}{2} - x_{12}) + \mu_3(\frac{1}{2} - x_{13})\\
\text{s. t.}\ & 2x_{11} + 2x_{12} + x_{13} \le 4,\\
& x_{1j} \in \{0,1\},\ \forall j \in \{1,2,3\};
\end{aligned}$$

$$\begin{aligned}
\mathcal{LGMP}_2(\mu)\ & (\text{decide } x_{21}, x_{22}, x_{23}):\\
\text{max.}\ & 4x_{21} + 2x_{22} + 2x_{23} + \mu_1(\frac{1}{2} - x_{21}) + \mu_2(\frac{1}{2} - x_{22}) + \mu_3(\frac{1}{2} - x_{23})\\
\text{s. t.}\ & 2x_{21} + 2x_{22} + 2x_{23} \le 3,\\
& x_{2j} \in \{0,1\},\ \forall j \in \{1,2,3\}.
\end{aligned}$$

Notice that in $\mathcal{LGMP}_k(\mu)$ the variables derived from other agents no longer appear in the constraints or in the objective functions. This can greatly simplify each agent's

search process, because when solving $\mathcal{LGMP}_k(\mu)$, agent k does not have to refer to other agents for their variable values.

Assuming that all of the agents find optimal solutions to their respective lagrangean relaxation problems, the following lemma holds.

Lemma 2 *All of the optimal solutions to* $\{\mathcal{LGMP}_k(\mu) \mid k \in A\}$ *constitute an optimal solution to* $\mathcal{LGAP}(\mu)$.

Proof: This also follows because the feasible region of $\mathcal{LGAP}(\mu)$ is equal to the intersection of all the feasible regions of $\{\mathcal{LGMP}_k(\mu) \mid k \in A\}$ and because the objective function of $\mathcal{LGAP}(\mu)$ is equal to the sum of all the objective functions of $\{\mathcal{LGMP}_k(\mu) \mid k \in A\}$. □

On the optimal values of the lagrangean relaxation problems, we can obtain the following.

Theorem 1 *The sum of the optimal values of* $\{\mathcal{LGMP}_k(\mu) \mid k \in A\}$ *provides an upper bound for the optimal value of* $\mathcal{GAP}$.

Proof: This follows from Lemma 2 and because the optimal value of $\mathcal{LGAP}(\mu)$ provides an upper bound for the optimal value of $\mathcal{GAP}$. □

Note that in the above, different upper bounds result from different values for μ. An upper bound should be lower (closer to the optimal) and hopefully minimized by setting appropriate values for μ. This minimization problem, whose goal is to minimize an upper bound for the optimal value of $\mathcal{GAP}$, is called the *lagrangean dual problem*. In solving the lagrangean dual problem, we can use the following as a termination criterion.

Theorem 2 *If all of the optimal solutions to* $\{\mathcal{LGMP}_k(\mu) \mid k \in A\}$ *satisfy the assignment constraints,* $\sum_{i \in A} x_{ij} = 1,\ \forall j \in J$, *for specific values of* μ, *then these optimal solutions constitute an optimal solution to* $\mathcal{GAP}$.

Proof: Theorem 1 proclaims that these optimal solutions constitute a solution that provides an upper bound for the optimal value of $\mathcal{GAP}$. Since these optimal solutions satisfy the assignment constraints, each of which is an equality constraint relaxed in $\mathcal{LGMP}_k(\mu)$ for any agent k, they also constitute a feasible solution that provides a lower bound for the optimal value. Hence, these optimal solutions constitute an optimal solution to $\mathcal{GAP}$. □

3 Protocol

Theorem 2 prompted the development of a distributed solution scheme, the *distributed lagrangean relaxation protocol*, where the agents start with $t = 0$ and $\mu^{(0)} = (0, \ldots, 0)$; then, they alternate the following series of actions (called a *round*) in parallel until all of the assignment constraints are satisfied.

1. Each agent k finds an optimal solution to $\mathcal{LGMP}_k(\mu^{(t)})$ (solves the *primal problem*).

2. Agents exchange these solutions with each other.
3. Each agent k updates the lagrangean multiplier vector from $\mu^{(t)}$ to $\mu^{(t+1)}$ (solves the *(lagrangean) dual problem*) and increases t by one.

We describe the key ideas and details of the protocol in this section.

3.1 Neighborhood and Communication Model

For each job j in R_k (a set of all jobs that may be assigned to k), agent k needs the values of lagrangean multiplier $\mu_j^{(t)}$, the size of set S_j (a set of all agents to which j may be assigned), and the decision variables of the related agents (i.e., $\{x_{ij} \mid i \in S_j,\ i \neq k\}$) in order to solve primal and dual problems. The value of $\mu_j^{(t)}$ is locally computed as we will explain later, and the size of set S_j is given to agent k as prior knowledge. On the other hand, the values of the decision variables are obtained through communication with a set of agents denoted as $\bigcup_{j \in R_k} S_j \setminus \{k\}$. We refer to the set of agents as agent k's *neighbors* and allow an agent to communicate only with its neighbors. Thus, the protocol assumes a peer-to-peer message exchange model, in which a message is never lost and, for any pair of agents, messages are received in the order in which they were sent.

3.2 Primal Problem

Once agent k gets the values of the associated lagrangean multipliers as well as the decision variables from its neighboring agents, agent k searches for an optimal solution to $\mathcal{LGMP}_k(\mu^{(t)})$ to determine the values of its decision variables $\{x_{kj} \mid j \in R_k\}$. This search problem is equivalent to the *knapsack problem* whose goal is to select the most profitable subset of R_k such that the total resource requirement of the subset does not exceed c_k. In the problem, for each job j in R_k, the profit is $p_{kj} - \mu_j^{(t)}$, and the resource requirement is w_{kj}. Since the knapsack problem belongs to NP-hard problems, we cannot expect, in principle, an efficient exact solution method. However, recent progress in exact solution methods for the knapsack problem is so remarkable that they can readily solve even a very large instance (up to around 10,000 jobs) in many cases [5].

After finding an optimal solution, agent k sends the solution (along with other information explained later) to its neighbors via an *assign* message.

3.3 Dual Problem

After receiving all of the current solutions of neighbors, agent k solves the lagrangean dual problem by updating the related lagrangean multipliers. In this work we use a *subgradient optimization method*, a well-known technique for systematically updating a lagrangean multiplier vector [8]. In the method, under the current values of the related decision variables, agent k first computes *subgradient* g_j for an assignment constraint of each job $j \in R_k$ as follows:

$$g_j = 1 - \sum_{i \in S_j} x_{ij}.$$

Basically, g_j provides an updating direction of multiplier μ_j by taking a positive value when no agent in S_j currently selects job j, a negative value when two or more agents in S_j currently select that job, and zero when an exactly one agent currently selects it. Then, using *step length* $l^{(t)}$, which decays at rate r ($0 < r \leq 1$) as time t, the round in this case, proceeds (i.e., $l^{(t+1)} \leftarrow r l^{(t)}$) and $|S_j|$, agent k updates the multiplier of job j as

$$\mu_j^{(t+1)} \leftarrow \mu_j^{(t)} - l^{(t)} \frac{g_j}{|S_j|}. \tag{5}$$

A basic idea behind this updating rule is simple. A multiplier of a job becomes higher when the job invites too many agents and lower when the job does not invite a required number of agents (one in this case). In this sense multiplier μ_j can be interpreted as the *price* of job j. Note that the degree of update depends on both the step length decaying at rate r and what percentage of the potentially involved agents should be encouraged/discouraged to select that job.

Notice that since multiplier μ_j is attached to job j all of the agents in S_j must agree on a common value for it. If the agents in S_j assign different values to μ_j, then neither theorem 1 nor 2 holds any more. To set such a common value, we give all of the agents a common initial value for μ, a common value for initial step length $l^{(0)}$, and a common value for decay rate r as their prior knowledge, and prohibit each agent from working at round $t+1$ until it receives all of the assign messages issued from its neighbors at round t. By doing this instead of implementing explicit communication among S_j, we can force agents to automatically set a common value to a lagrangean multiplier.

On the other hand, as we mentioned above, if the agents in S_j assign different values to μ_j, then neither theorem 1 nor 2 holds. However, in practice, this *intentional* assignment of different values seems to cause the agents to rapidly converge to a near optimal solution to $\mathcal{GAP}$. This topic will be dealt with in detail in Section 3.6.

3.4 Convergence Detection

We can terminate the protocol when all of the optimal solutions to $\{\mathcal{LGMP}_k(\mu^{(t)}) \mid k \in A\}$ satisfy the assignment constraints, $\sum_{i \in A} x_{ij} = 1,\ \forall j \in J$. This fact can be discovered by a solution detection procedure of the *distributed breakout algorithm* [3] (locally synchronized solvers for the distributed constraint satisfaction problem [10]). Thus, we incorporate a similar detection process into the protocol, which is briefly described as follows.

In the protocol, agent k has boolean variable cs_k and non-negative integer variable tc_k (initialized by zero). Intuitively, cs_k represents whether the assignment constraints of agent k is locally satisfied while tc_k represents how far from agent k all of their respective assignment constraints are satisfied. Agent k sends the values of cs_k and tc_k to its neighbors via an assign message together with its current solution. Suppose that agent k receives all of the assign messages from its neighbors. Agent k can now detect whether all of its assignment constraints, $\sum_{i \in S_j} x_{ij} = 1,\ \forall j \in R_k$, are satisfied. Thus, agent k sets cs_k as true if they are satisfied and false otherwise. On the other hand, if its cs_k becomes true and all of the received css are true, then agent k identifies a

minimum value over k's and its neighbors' tcs and sets tc_k as a minimum value plus one; otherwise, it sets tc_k to zero. In the above, if the value of tc_k reaches a diameter of the communication network, whose nodes represent agents and links represent neighborhood relationships, then all of the assignment constraints are satisfied. Note that the diameter can be replaced by the number of agents, which is an upper bound of the diameter.

3.5 Example

We will trace how the protocol works using Example 3. Let the initial values for lagrangean multiplier vector (μ_1, μ_2, μ_3) be $(0, 0, 0)$, a common value for initial step length $l^{(0)}$ be 1, and decay rate r be 1.

First, agent 1 solves the knapsack problem instance with three jobs: job 1 (profit: 5, resource requirement: 2), job 2 (profit: 6, resource requirement: 2), and job 3 (profit: 5, resource requirement: 1), and 4 available resources, and finds $\{\mathrm{job}\ 1, \mathrm{job}\ 2\}$ the most profitable subset to send to agent 2 via an assign message. On the other hand, agent 2 solves the knapsack problem instance with three jobs: job 1 (profit: 4, resource requirement: 2), job 2 (profit: 2, resource requirement: 2), and job 3 (profit: 2, resource requirement: 2), and 3 available resources, and finds $\{\mathrm{job}\ 1\}$ the most profitable subset to send to agent 1 via an assign message.

After receiving the assign message from agent 2, agent 1 computes subgradients (g_1, g_2, g_3) as $(-1, 0, 1)$ under the current values of the decision variables and updates the lagrangean multiplier vector as $(0.5, 0, -0.5)$. On the other hand, agent 2 does the same and also updates the vector in the same way. Next, using this updated multiplier vector, agents 1 and 2 identify their new knapsack problem instances, respectively. Agent 1 solves the knapsack problem instance with job 1 (profit: 4.5, resource requirement: 2), job 2 (profit: 6, resource requirement: 2), job 3 (profit: 5.5, resource requirement: 1), and 4 available resources, and finds $\{\mathrm{job}\ 2, \mathrm{job}\ 3\}$ the most profitable subset to send to agent 2 via an assign message. On the other hand, agent 2 solves the knapsack problem instance with job 1 (profit: 3.5, resource requirement: 2), job 2 (profit: 2, resource requirement: 2), job 3 (profit: 2.5, resource requirement: 2), and 3 available resources, and finds $\{\mathrm{job}\ 1\}$ the most profitable subset to send to agent 1 via an assign message.

After receiving this assign message, agents 1 finds (g_1, g_2, g_3) to be $(0, 0, 0)$ (meaning all of its assignment constraints are satisfied), and thus does not update the lagrangean multiplier vector. Therefore, agent 1 finds the previous subset $\{\mathrm{job}\ 2, \mathrm{job}\ 3\}$ the most profitable and resends it to agent 2 via an assign message. In this case, the message also includes true as cs_1, since all of its assignment constraints are currently satisfied. On the other hand, agent 2 does the same and sends $\{\mathrm{job}\ 1\}$ and true as cs_2 to agent 1 via an assign message.

These messages cause both agents to increase their respective non-negative integer variables, tc_1 for agent 1 and tc_2 for agent 2, to one and send their respective assign messages. Finally, in the next round, both agents increase their respective tcs to two (which is equal to the number of agents) and can terminate their procedures. The final job assignment, $\{\mathrm{job}\ 2, \mathrm{job}\ 3\}$ for agent 1 and $\{\mathrm{job}\ 1\}$ for agent 2, is obviously the optimal solution.

```
procedure init
1.  round_k := 1;
2.  cs_k := false;
3.  tc_k := 0;
4.  leng := someCommonValue;
5.  r := someCommonRatio;
6.  δ := someCommonRatio;
7.  cutOffRound := someCommonValue;
8.  for each job j ∈ R_k do μ_j := 0 end do;
9.  Jobs_k := optimal solution to knapsack problem;
10. send assign(k, round_k, cs_k, tc_k, Jobs_k) to neighbors;
11. WaitL := neighbors;
12. cs_N := true;
```

Fig. 1. Distributed lagrangean relaxation protocol: init procedure

3.6 Convergence to Feasible Solution

As mentioned above, the protocol terminates when all of the assignment constraints have been satisfied. But, is there any guarantee that they all are eventually satisfied? The answer is unfortunately no, and the protocol must be forced to terminate after a certain number of rounds on the way to an optimal solution. However, the protocol has not yet discovered any feasible solution to $\mathcal{GAP}$ on the way to an optimal solution.

For combinatorial optimization problems in a centralized context, even if terminated on the way to an optimal solution, the lagrangean relaxation method can usually find a feasible solution because it contains a domain-specific technique, generally called *lagrangean heuristics*, that transforms the "best" infeasible solution into a feasible one. However, it may be difficult to devise such a heuristic in our protocol because such a "best" infeasible solution, which we believe belongs to global information, is not at hand in a distributed context.

Therefore, we introduce a simple technique that produces quick agreement between the agents on a feasible solution with reasonably good quality by simply replacing multiplier updating rule (5) with another. As mentioned in Section 3.3, for any job j, all the agents in S_j must agree on a common value to the respective multiplier μ_j. This is necessary for theorems 1 and 2 to be true. On the other hand, we observed that a common value to μ_j sometimes yields an oscillation where some of the agents in S_j "cluster and disperse" around job j, causing the protocol to fail to find an optimal solution. Thus, we relax this requirement by letting the agents in S_j assign slightly different values to μ_j by introducing a parameter δ $(0 \leq \delta \leq 1)$ that controls the range of "noise" mixed with increment/decrement in a lagrangean multiplier. The noise, denoted as N_δ, is a random variable whose value is uniformly distributed over $[-\delta, \delta]$. Multiplier updating rule (5) is thus replaced by

$$\mu_j^{(t+1)} \leftarrow \mu_j^{(t)} - (1 + N_\delta) l^{(t)} \frac{g_j}{|S_j|}. \tag{6}$$

This rule diversifies agents' views on the value of μ_j (the price of job j) and thus can break oscillations. On the other hand, since this rule violates the theorems, the protocol may converge to a non-optimal (but feasible) solution. Note that rule (6) is equal to rule (5) if δ is set to zero.

```
when k receives assign(i, round_i, cs_i, tc_i, Jobs_i) from i do
 1.  if round_k < round_i then
 2.    add this message in DeferredL;
 3.  else
 4.    if cs_i then
 5.      tc_k := min(tc_k, tc_i);
 6.    else
 7.      cs_N := false;
 8.      update AgentView with Jobs_i;
 9.    end if;
10.    delete i from WaitL;
11.    if WaitL is empty then
12.      stop := localcomp;
13.      if stop then
14.        terminate the procedure;
15.      else
16.        WaitL := neighbors;
17.        cs_N := true;
18.        restore each message in DeferredL to process;
19.      end if;
20.    end if;
21.  end if;
end do
```

Fig. 2. Distributed lagrangean relaxation protocol: message processing procedure

All of these ideas are merged into a series of procedures described in Figs. 1–3.

4 Experiments

Since, to the author's knowledge, there has been no comparable distributed solution method for the problem, we conducted experiments to clarify the characteristics of the protocol instead of to show its competitiveness. Thus, in our experiments, we observed the performance of the protocol when varying the values of δ and the *assignment topologies* that will be described below.

We adopted two problem suites: one is hand-made but has various assignment topologies; the other is from the GAP benchmark instances of OR-Library[1] but has a specific assignment topology.

The first problem suite, which is hand-made but has various assignment topologies, includes instances in which there exist $m \in \{5, 7\}$ agents (numbered from 1 to m), each of which has 5 jobs, meaning that the total number of jobs n was $5m$, and tries to assign each of its jobs to some agent (that may be itself) using one of the following *assignment topologies*:

chain The i^{th} agent ($i \in \{2, \ldots, m-1\}$) tries to assign each of its jobs to the $(i-1)^{th}$ agent, the $(i+1)^{th}$ agent, or itself. On the other hand, the 1^{st} agent tries to assign each job to either the 2^{nd} agent or itself; the m^{th} agent tries to assign each job to either the $(m-1)^{th}$ agent or itself.

[1] http://people.brunel.ac.uk/~mastjjb/jeb/orlib/gapinfo.html

```
procedure localcomp
1.  round_k := round_k + 1;
2.  leng := leng * r;
3.  if round_k > cutOffRound then
4.    return true;
5.  else
6.    cs_k := true;
7.    for each job j ∈ R_k do
8.      calculate subgradient g_j based on AgentView;
9.      if g_j ≠ 0 then
10.       cs_k := false;
11.       ε := randomly chosen value from [−δ, δ];
12.       μ_j := μ_j − (1 + ε) * leng * g_j/|S_j|;
13.     end if;
14.   end do;
15.   if cs_k ∧ cs_N then
16.     tc_k := tc_k + 1;
17.     if tc_k = #agents then return true end if;
18.   else
19.     tc_k := 0;
20.     Jobs_k := optimal solution to knapsack problem;
21.   end if;
22.   send assign(k, round_k, cs_k, tc_k, Jobs_k) to neighbors;
23.   return false;
24. end if;
```

Fig. 3. Distributed lagrangean relaxation protocol: localcomp procedure

ring The i^{th} agent ($i \in \{2, \ldots, m-1\}$) tries to assign each of its jobs to the $(i-1)^{th}$ agent, the $(i+1)^{th}$ agent, or itself. On the other hand, the 1^{st} agent tries to assign each job to the m^{th} agent, the 2^{nd} agent, or itself; the m^{th} agent tries to assign each job to the $(m-1)^{th}$ agent, the 1^{st} agent, or itself.

cmplt Each agent tries to assign each of its jobs to any of the agents (including itself).

rndm3 Each agent tries to assign each of its jobs to any of the three agents, two of which are randomly selected from the other agents for each job and the other is itself.

For a set of m agents with one of the above assignment topologies, a random instance was made by randomly selecting an integer value from $[1, 10]$ for both resource requirement w_{ij} and profit p_{ij}. We fixed available resource capacity c_i to 20 for any agent i in every instance. To ensure that all of the problem instances are feasible, we pre-checked if generated instances are feasible with a centralized exact solver (ILOG CPLEX mip solver) and screened out infeasible ones.

The second problem suite includes those obtained from the GAP benchmark instances of OR-Library. The instances of the library are originally designed as GAP instances and do not include any topological information among agents. We therefore assume a complete topology (cmplt in the above) among agents and translate a GAP instance into a GMAP instance. It is worth noting that the GMAP with a complete topology is equivalent to the problem in which the agents search for an optimal partition of public jobs that does not violate their individual (and private) knapsack constraints.

The protocol was implemented in Java. The agents in the protocol solved knapsack problem instances using a branch-and-bound algorithm with LP bounds and were able

Table 1. Experimental results on hand-made problem suite with various assignment topologies. The Pr.ID column shows a label of a problem instance that indicates, from left to right, assignment topology, number of agents, total number of jobs, available resource capacity, and instance identifier

Pr.ID	δ	O.R.	F.R.	A.Q.	B.Q.	A.C.	Pr.ID	δ	O.R.	F.R.	A.Q.	B.Q.	A.C.
chain-5-25-20-000	0.0	0/1	0/1	N/A	N/A	2500	chain-7-35-20-000	0.0	0/1	0/1	N/A	N/A	3500
	0.3	5/20	20/20	0.989	1.000	269.6		0.3	3/20	19/20	0.969	1.000	810.7
	1.0	1/20	20/20	0.964	1.000	148.0		1.0	0/20	19/20	0.959	0.986	473.8
chain-5-25-20-001	0.0	0/1	0/1	N/A	N/A	2500	chain-7-35-20-001	0.0	0/1	0/1	N/A	N/A	3500
	0.3	11/20	20/20	0.995	1.000	84.2		0.3	5/20	20/20	0.983	1.000	359.0
	1.0	3/20	20/20	0.985	1.000	72.4		1.0	3/20	20/20	0.966	1.000	283.5
ring-5-25-20-000	0.0	0/1	0/1	N/A	N/A	2500	ring-7-35-20-000	0.0	0/1	0/1	N/A	N/A	3500
	0.3	1/20	19/20	0.966	1.000	832.6		0.3	3/20	18/20	0.959	1.000	993.7
	1.0	1/20	20/20	0.957	1.000	362.9		1.0	0/20	20/20	0.946	0.996	164.0
ring-5-25-20-001	0.0	0/1	0/1	N/A	N/A	2500	ring-7-35-20-001	0.0	0/1	0/1	N/A	N/A	3500
	0.3	5/20	20/20	0.970	1.000	373.6		0.3	1/20	20/20	0.949	1.000	1013.0
	1.0	1/20	20/20	0.939	1.000	161.6		1.0	0/20	20/20	0.935	0.983	278.3
cmplt-5-25-20-000	0.0	0/1	0/1	N/A	N/A	2500	cmplt-7-35-20-000	0.0	0/1	0/1	N/A	N/A	3500
	0.3	0/20	20/20	0.934	0.980	423.9		0.3	0/20	20/20	0.929	0.977	559.3
	1.0	0/20	20/20	0.881	0.959	182.5		1.0	0/20	20/20	0.865	0.928	151.4
cmplt-5-25-20-001	0.0	0/1	0/1	N/A	N/A	2500	cmplt-7-35-20-001	0.0	0/1	0/1	N/A	N/A	3500
	0.3	1/20	20/20	0.954	1.000	245.7		0.3	0/20	20/20	0.938	0.987	488.4
	1.0	0/20	20/20	0.908	0.953	111.4		1.0	0/20	20/20	0.883	0.960	179.9
rndm3-5-25-20-000	0.0	0/1	0/1	N/A	N/A	2500	rndm3-7-35-20-001	0.0	0/1	0/1	N/A	N/A	3500
	0.3	8/20	20/20	0.977	1.000	527.3		0.3	1/20	18/20	0.971	1.000	1213.0
	1.0	2/20	20/20	0.951	1.000	288.6		1.0	1/20	18/20	0.951	1.000	966.3
rndm3-5-25-20-001	0.0	0/1	0/1	N/A	N/A	2500	rndm3-7-35-20-002	0.0	0/1	0/1	N/A	N/A	3500
	0.3	19/20	20/20	0.998	1.000	40.6		0.3	3/20	16/20	0.967	1.000	1507.0
	1.0	10/20	20/20	0.957	1.000	53.8		1.0	1/20	20/20	0.916	1.000	507.9

to exchange messages using TCP/IP socket communication on specific ports. In the experiments, we put m agents in one machine and let them communicate using their local ports. The parameters for the protocol were fixed as follows: $cutOffRound = 100n$, $l^{(0)} = 1.0$, and $r = 1.0$, where *cutOffRound* is the upper bound of rounds at which a run is forced to terminate, $l^{(0)}$ is an initial value for the step length, and r is a decay rate of the step length. Parameter δ, which controls the degree of noise, ranged over $\{0.0, 0.3, 1.0\}$. For each problem instance, 20 runs were made at each value of δ (except for $\delta = 0.0$) and the following data were measured:

Opt.Ratio ratio of the runs where optimal solutions were found;
Fes.Ratio ratio of the runs where feasible solutions were found;
Avg.Quality average value of solution qualities;
Bst.Quality best value of solution qualities;
Avg.Cost average value of the numbers of rounds at which feasible solutions were found.

Note that in Avg/Bst.Quality, solution quality was measured as the ratio of the profit of an obtained feasible solution to optimal value. Note also that, when a run finished with no feasible solution, we did not count the run for Avg/Bst.Quality, but we did count it using the value of *cutOffRound* for Avg.Cost. On the other hand, when δ is 0.0, we made only one run because there was no randomness in the protocol at that setting.

The results are shown in Tables 1 and 2.

Table 2. Experimental results on benchmark problem suite with a complete assignment topology. The Pr.ID column shows the label of a corresponding instance in the OR-Library (i.e., in "cmnn-i", m is number of agents, nn is total number of jobs, and i is an instance identifier).

Pr.ID	δ	O.R.	F.R.	A.Q.	B.Q.	A.C.	Pr.ID	δ	O.R.	F.R.	A.Q.	B.Q.	A.C.
c515-1	0.3	2/10	10/10	0.993	1.000	277.1	c824-1	0.3	0/10	10/10	0.979	0.996	321.8
	1.0	0/10	10/10	0.937	0.994	222.4		1.0	0/10	10/10	0.940	0.966	149.4
c520-1	0.3	2/10	10/10	0.987	1.000	456.2	c832-1	0.3	0/10	10/10	0.977	0.996	458.1
	1.0	0/10	10/10	0.955	0.979	193.0		1.0	0/10	10/10	0.925	0.954	183.7
c525-1	0.3	0/10	10/10	0.977	0.990	605.0	c840-1	0.3	0/10	10/10	0.974	0.988	653.2
	1.0	0/10	10/10	0.958	0.984	200.2		1.0	0/10	10/10	0.934	0.950	298.7
c530-1	0.3	1/10	10/10	0.979	1.000	660.4	c848-1	0.3	0/10	10/10	0.964	0.975	1304.4
	1.0	0/10	10/10	0.945	0.970	202.0		1.0	0/10	10/10	0.931	0.949	298.8

As we mentioned in Section 3.6, rule (6) is equal to rule (5) when δ is 0.0. The protocol with this setting, therefore, is to terminate only when an optimal solution is found (otherwise, it is forced to be terminated when the cutoff round $100n$ is reached). However, in the experiments, we observed that the protocol at that setting failed to find optimal solutions within the cutoff round for all instances of the hand-made problem suite. A close look at the behavior of agents revealed that although the agents as a whole could reach to a near-optimal solution (an infeasible solution giving a tight upper bound) very quickly, they eventually fell into a loop where some agents cluster and disperse around a specific set of jobs. Accordingly, we did not try the protocol with $\delta = 0.0$ for the instances of the benchmark problem suite.

The performance of the protocol dramatically changed when δ was set to one of the non-zero values. The results show that Opt.Ratio, Fes.Ratio, and Avg.Cost are obviously improved while Avg/Bst.Quality is kept at a reasonable level, suggesting that by using the protocol with those settings, agents can quickly agree on a feasible solution with reasonably good quality.

It is also true that the protocol with those settings may fail to find an optimal solution. In the experiments, it failed to find an optimal solution at every non-zero value of δ for 3 complete topology instances of the hand-made problem suite (cmplt-5-25-20-000, cmplt-7-35-20-000, and cmplt-7-35-20-001) and 5 instances of the benchmark problem suite. However, for each of the other instances, an optimal solution was found in at least one run at some value of δ.

For almost all of the instances, we can see that when δ increases both Avg.Cost and Avg.Quality are reduced. This means that increasing δ may generally influence agents to rush to reach a compromise of lower quality. This implies that parameter δ may allow us to control tradeoffs between the quality of a solution and the cost of finding it.

With these limited number of hand-made problem instances, we cannot say for certain whether the assignment topologies can affect the performance of the protocol. Our result, however, clearly shows that the instances with a complete assignment topology ended up with lower quality solutions than those with the other assignment topologies. In an instance with a complete assignment topology, all of the agents are involved in each job j (i.e., $S_j = All, \forall j \in J$) and apply rule (6) independently when it is time to update μ_j. These "noisy" updates by many agents generally accelerate the diversi-

fication of agents' views on multipliers and may force the agents to rush to reach a compromise of lower quality.

5 Conclusion

We presented the GMAP to deal with the problems of job assignment in multi-agent systems. To solve the GMAP, we also presented a distributed lagrangean relaxation protocol that enables agents to simultaneously solve a GMAP instance on a peer-to-peer message exchange mechanism. Furthermore, to control the performance of the protocol, we introduced a parameter δ that controls the degree of noise mixed with increment/decrement in a lagrangean multiplier. Our experimental results showed that if δ is set to a non-zero value, the agents can quickly agree on feasible solutions with reasonably good quality. The results also suggested that the parameter may allow us to control tradeoffs between the quality of a solution and the cost of finding it.

We believe the potential of this approach since, unlike the local search algorithms such as SA or GA, a lagrangean relaxation method can provide both upper and lower bounds of the optimal value. Actually, our protocol will be able to compute a lower bound by setting δ to a non-zero value and adding up the objective values of final assignments; a upper bound by setting δ to zero and adding up the objective values of interim assignments. In our future work, we would like to pursue a distributed method to compute upper bounds for the optimal value and a more sophisticated technique to update a lagrangean multiplier vector.

References

1. I. P. Androulakis and G. V. Reklaitis. Approaches to asynchronous decentralized decision making. Computers and Chemical Engineering 23, pp.341–355, 1999.
2. K. Hirayama and M. Yokoo. Distributed partial constraint satisfaction problem Proc. of CP-97, pp.222–236 (1997).
3. K. Hirayama and M. Yokoo. The distributed breakout algorithms. Artificial Intelligence Vol.161, No.1-2, pp.89–115 (2005).
4. R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. Proc. of AAMAS-04, pp.438–445 (2004).
5. S. Martello, D. Pisinger, and P. Toth. New trends in exact algorithms for the 0-1 knapsack problem. European Journal of Operational Research 123, pp.325–332, 2000.
6. P. J. Modi, W. M. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. Proc. of AAMAS-03, pp.161–168 (2003).
7. I. H. Osman. Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches. OR Spektrum 17, pp. 211–225, 1995.
8. C. R. Reeves. Modern heuristic techniques for combinatorial problems. Blackwell, 1993.
9. R. G. Smith. The contract net protocol: high-level communication and control in a distributed problem solver. IEEE Transaction on Computers 29(2), pp. 1104–1113, 1990.
10. M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: formalization and algorithms. IEEE Transactions on Knowledge and Data Engineering 10(5), pp. 673–685, 1998.