A New Approach to Distributed Task Assignment using Lagrangian Decomposition and Distributed Constraint Satisfaction

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

Abstract

We present a new formulation of distributed task assignment, called *Generalized Mutual Assignment Problem* (GMAP), which is derived from an NP-hard combinatorial optimization problem that has been studied for many years in the operations research community. To solve the GMAP, we introduce a novel distributed solution protocol using *Lagrangian decomposition* and *distributed constraint satisfaction*, where the agents solve their individual optimization problems and coordinate their locally optimized solutions through a distributed constraint satisfaction technique. Next, to produce quick agreement between the agents on a feasible solution with reasonably good quality, we provide a parameter that controls the range of "noise" mixed with an increment/decrement in a Lagrange multiplier. Our experimental results indicate that the parameter may allow us to control tradeoffs between the quality of a solution and the cost of finding it.

Introduction

The distributed task assignment problem, which concerns assigning a set of jobs to multiple agents, has been one of the main research issues since the early days of distributed problem solving (Smith 1990). Recently, several studies have been reported in which they formalize the problem as an optimization problem to be solved by using certain general solution protocols (Gerkey & Matarić 2004; Kutanoglu & Wu 1999).

In this paper, we present a new formulation of distributed task assignment which is derived from the *Generalized Assignment Problem* (GAP). The GAP is a typical NP-hard combinatorial optimization problem that has been studied for many years in the operations research community (Cattrysse & Wassenhove 1992; Osman 1995). 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.

To adapt the GAP to a distributed environment, we first introduce the *Generalized Mutual Assignment Problem* (GMAP). The GMAP is the distributed problem in which each agent in the GAP has a set of jobs and tries to achieve the goal of the GAP. To put it another way, the agents themselves try to generate an optimal solution of the GAP from a non-optimal (and sometimes infeasible) solution of it. To solve the GMAP, we introduce a novel distributed solution protocol using *Lagrangian decomposition* (Guignard & Kim 1987) and *distributed constraint satisfaction* (Yokoo *et al.* 1998). In this protocol, the agents solve their individual optimization problems and coordinate their locally optimized solutions through a distributed constraint satisfaction technique. One important feature of this protocol is that, unlike other optimization protocols for distributed task assignment, it is a pure distributed protocol where neither the server nor the coordinator exists.

The paper is organized as follows. After mentioning related work, we describe the GAP formulation of the global problem and the set of problems produced by decomposing the Lagrangian relaxation problem of the GAP. Next, we describe the key ideas of the protocol, including the 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. Finally, we report our experiments that assessed the actual performance of the protocol and conclude this paper.

Related Work

Several solution protocols have been proposed for solving distributed optimization problems, but in our view they can be divided into three categories.

The first is a *centralized solution protocol*, where the agents send all of the subproblems to a server which then solves the entire problem by using a particular solver and delivers a solution to each agent. An example of this protocol can be found in multi-robot systems that try to optimize task allocations to multiple robots (Gerkey & Matarić 2004). The centralized solution protocol is very easy to realize and might have lower costs in terms of time and messages. However, for some applications embedded in open environments, the operation of gathering all the subproblems in the server may cause a security or privacy issue and the cost of building and running the server.

The second category is a *decentralized solution protocol*. In this protocol, each agent rather than the server is equipped

with a solver and the server plays the role of a coordinator to facilitate communication among the agents. An example of this protocol can be found in distributed scheduling. For example, Kutanoglu and Wu have presented an auction-based protocol for the resource scheduling problem, in which the agents solve the problem in a distributed fashion under the coordination of the *auctioneer* (Kutanoglu & Wu 1999). Similar approaches have been proposed for the nonlinear programming problem (Androulakis & Reklaitis 1999) and the supply-chain optimization problem (Nishi, Konishi, & Hasebe 2005). Although the decentralized solution protocol allows the agents to keep their subproblems private, the server can obtain all of the information on solving processes whereby the server may guess the entire problem. Furthermore, the protocol still needs a server that requires tedious maintenance.

The third category is a *distributed solution protocol*. Unlike the other protocols, this protocol does not need a server because the agents themselves try to solve the entire problem through peer-to-peer local communication. It is noteworthy that with this protocol, every agent knows neither what the entire problem is nor how the entire solving processes are run. Recently, this type of protocol has been proposed for solving the distributed partial constraint satisfaction problem (Hirayama & Yokoo 1997) and the distributed constraint optimization problem (Modi *et al.* 2003; Petcu & Faltings 2005). We believe that it is a promising direction for distributed problem solving because it is well suited to optimization tasks in open environments.

Formalization

A GMAP instance consists of multiple agents each having a finite set of jobs that are to be assigned. The agents as a whole solve the following *integer programming problem*, denoted as $\mathcal{GAP}$.

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

where $A = \{1, ..., m\}$ is a set of agents; $J = \{1, ..., n\}$ is a set of jobs; p_{kj} and w_{kj} are the profit and amount of resource required, respectively, when agent k selects job j; c_k is the capacity, i.e., the amount of available resource, of agent k. x_{kj} is a decision variable whose value is set to 1 when agent k selects job j and 0 otherwise. The goal of the agents is to find a job assignment that maximizes the total sum of profits such that (1) each job is assigned to exactly one agent (denoted as *assignment constraints*), (2) the total amount of resource required for each agent does not exceed its capacity (denoted as *knapsack constraints*), and (3) each job is assigned or not assigned to an agent (denoted as *01 constraints*). We refer to the maximal total sum of profits as the *optimal value* and the assignment that provides the optimal value as the *optimal solution*.

We obtain the following Lagrangian relaxation problem $\mathcal{LGAP}(\mu)$ by dualizing the assignment constraints of $\mathcal{GAP}$ (Fisher 1981).

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

where μ_j is a real-valued parameter called a *Lagrange multiplier* for (the assignment constraint of) job j. Note that the vector $\mu = (\mu_1, \mu_2, \ldots, \mu_n)$ is called a *Lagrange multiplier vector*. Based on the idea of Lagrangian decomposition (Guignard & Kim 1987), we can divide this problem into a set of subproblems $\{\mathcal{LGMP}_k(\mu) | k \in A\}$, where each subproblem $\mathcal{LGMP}_k(\mu)$ of agent k is as follows.

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

where R_k is a set of jobs that may be assigned to agent k and S_j is a set of agents to whom job j may be assigned. We can assume that $S_j \neq \emptyset$ because a job with an empty S_j does not have to be considered in the problem. When we consider that agent k decides the value of x_{kj} for $\forall j \in R_k$ (in other words, the agent that may be assigned job j has a right to decide whether it will undertake the job or not), agent k is able to solve $\mathcal{LGMP}_k(\mu)$ completely by itself since $\mathcal{LGMP}_k(\mu)$ includes only k's decision variables.

Regarding the relation between the entire problem and a set of subproblems, the following properties are very useful in designing a distributed solution protocol for the GMAP.

Property 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 of the optimal value of $\mathcal{GAP}$.*

Property 2 *For any value of μ, if all of the optimal solutions to $\{\mathcal{LGMP}_k(\mu) | k \in A\}$ satisfy the assignment constraints, $\sum_{i \in A} x_{ij} = 1,\ \forall j \in J$, then these optimal solutions constitute an optimal solution to $\mathcal{GAP}$.*

Protocol

Property 2 prompted the development of a distributed solution protocol, called the *distributed Lagrangian relaxation protocol*, where the agents start with $t = 0$ and

$\mu^{(0)} = (0, \ldots, 0)$ and 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 their neighboring agents.
3. Each agent k updates the Lagrange multiplier vector from $\mu^{(t)}$ to $\mu^{(t+1)}$ (solves the *dual problem*) and increases t by one.

It must be noted that the overall behavior of this protocol is similar to the *distributed breakout algorithm* for solving the DisCSP (Hirayama & Yokoo 2005). We describe the key ideas and details of the protocol in this section.

Neighborhood and Communication Model

In order to solve the primal and dual problems, agent k needs, for each job j in R_k, the values of $\mu_j^{(t)}$, the size of S_j, and the decision variables of the related agents (i.e., $\{x_{ij} \mid i \in S_j,\ i \neq k\}$). The value of $\mu_j^{(t)}$ is locally computed as we will explain later, and the size of 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 each agent in a set denoted as $\bigcup_{j \in R_k} S_j \setminus \{k\}$. We refer to this 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.

Primal Problem

Once agent k decides the values of the associated Lagrange multipliers, it searches for an optimal solution to $\mathcal{LGMP}_k(\mu^{(t)})$ to determine the values of its own 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 is an NP-hard problem, we cannot expect, in principle, an efficient exact solution method. However, recent attempts at exact solution methods for the knapsack problem have been so remarkable that they can readily solve even a very large instance in many cases (Martello, Pisinger, & Toth 2000).

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

Dual Problem

After receiving all of the current solutions of neighbors, agent k solves the dual problem, whose goal is to minimize an upper bound of the optimal value of $\mathcal{GAP}$, by updating the related Lagrange multipliers. Here, we used a *subgradient optimization method*, which is a well-known technique for systematically updating a Lagrange multiplier vector (Fisher 1981). In the method, using 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 μ_j by taking a positive integer when no agent in S_j currently selects job j, a negative integer when two or more agents in S_j currently select that job, and zero when exactly one agent currently selects it. Then, using $|S_j|$ and *step length* $l^{(t)}$, which may decay at rate r ($0 < r \leq 1$) as the number of rounds t increases (i.e., $l^{(t+1)} \leftarrow r l^{(t)}$), agent k updates the multiplier of job j as

$$\mu_j^{(t+1)} \leftarrow \mu_j^{(t)} - l^{(t)} \frac{g_j}{|S_j|}. \tag{4}$$

The 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 invites too few agents. In this sense, μ_j can be viewed 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 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, neither property 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 the initial step length $l^{(0)}$, and a common value for the decay rate r as their prior knowledge, and prohibit each agent from working in round $t+1$ until it receives all of the assign messages issued from its neighbors in round t. By doing this, instead of implementing explicit communication among S_j, we can force agents to automatically set a common value as a Lagrange multiplier.

Convergence Detection

The protocol should be terminated when all of the optimal solutions to $\{\mathcal{LGMP}_k(\mu^{(t)}) \mid k \in A\}$ satisfy all of the assignment constraints, but it is not so simple to detect this fact because no agent knows the entire solving processes. The protocol uses a termination detection procedure of the distributed breakout algorithm (Hirayama & Yokoo 2005) to detect this fact, which is summarized as follows.

Agent k has a boolean variable cs_k and a 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 that all of their respective assignment constraints are satisfied. Agent k sends 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. It can now determine whether all of its assignment constraints, $\sum_{i \in S_j} x_{ij} = 1,\ \forall j \in R_k$, are satisfied. Thus, it 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, it identifies a minimum value over k's and its neighbors' tcs and sets tc_k as the minimum value plus one; otherwise, it sets tc_k to zero. In the above, if the value of tc_k

```
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;
```

Figure 1: Distributed Lagrangian relaxation protocol: init procedure

```
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
```

Figure 2: Distributed Lagrangian relaxation protocol: message processing procedure

reaches the diameter of the communication network, whose nodes represent agents and links represent neighborhood relationships between pairs of agents, 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.

Convergence to a Feasible Solution

Unfortunately, the protocol may be trapped in an infinite loop depending on the parameter values of rule (4). In that case, the protocol must be forced to terminate after a certain number of rounds, despite that it has not yet discovered any feasible solution on the way to an optimal solution. Therefore, we make rule (4) stochastic so that the protocol can break an infinite loop and produce quick agreement between the agents on a feasible solution with reasonably good quality. More specifically, we let 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 an increment/decrement in the Lagrange multiplier. The noise, denoted as N_δ, is a random variable whose

```
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;
```

Figure 3: Distributed Lagrangian relaxation protocol: localcomp procedure

value is uniformly distributed over $[-\delta, \delta]$. The multiplier updating rule (4) is thus replaced by

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

This rule diversifies the agents' views on the value of μ_j (the price of job j) and thus the protocol can break an infinite loop. On the other hand, since properties 1 and 2 do not hold anymore under this rule, the protocol may converge to a non-optimal (but feasible) solution. Note that rule (5) is equal to rule (4) if δ is set to zero.

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

Experiments

Through experiments, we assessed the performance of the protocol when varying the values of δ and the *assignment topologies*. We chose two problem suites, a hand-made one that had various assignment topologies and the other from the GAP benchmark instances of OR-Library[1], which had a specific assignment topology.

The first problem suite 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) by 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 of its jobs to either the 2^{nd} agent or itself while the m^{th} agent tries to assign each of its jobs to either the $(m-1)^{th}$ agent or itself.

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

ring The story is the same as that in the above except that the 1^{st} agent tries to assign each of its job to the m^{th} agent, the 2^{nd} agent, or itself while the m^{th} agent tries to assign each of its job to either 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).

rndm 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.

Note that this yields GMAP instances by considering the agent who receives a job offer decides whether it will do the job or not. 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 the resource requirement w_{ij} and profit p_{ij}. We fixed the available resource capacity c_i to 20 for any agent i in every instance. To ensure that all of the problem instances were feasible, we pre-checked the generated instances with a centralized exact solver and screened out the infeasible ones.

Regarding the second problem suite, the instances of OR-Library were 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 a problem in which agents search for an optimal partition of public jobs that does not violate their individual 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 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 was 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 for 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 solution qualities;

Avg.Cost average number of rounds at which feasible solutions were found.

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

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, and instance identifier

Pr.ID	δ	O.R.	F.R.	A.Q.	A.C.	Pr.ID	δ	O.R.	F.R.	A.Q.	A.C.
chain-	0.0	0/1	0/1	N/A	2500	chain-	0.0	0/1	0/1	N/A	3500
525-0	0.3	5/20	20/20	0.989	269.6	735-0	0.3	3/20	19/20	0.969	810.7
	1.0	1/20	20/20	0.964	148.0		1.0	0/20	19/20	0.959	473.8
chain-	0.0	0/1	0/1	N/A	2500	chain-	0.0	0/1	0/1	N/A	3500
525-1	0.3	11/20	20/20	0.995	84.2	735-1	0.3	5/20	20/20	0.983	359.0
	1.0	3/20	20/20	0.985	72.4		1.0	3/20	20/20	0.966	283.5
ring-	0.0	0/1	0/1	N/A	2500	ring-	0.0	0/1	0/1	N/A	3500
525-0	0.3	1/20	19/20	0.966	832.6	735-0	0.3	3/20	18/20	0.959	993.7
	1.0	1/20	20/20	0.957	362.9		1.0	0/20	20/20	0.946	164.0
ring-	0.0	0/1	0/1	N/A	2500	ring-	0.0	0/1	0/1	N/A	3500
525-1	0.3	5/20	20/20	0.970	373.6	735-1	0.3	1/20	20/20	0.949	1013.0
	1.0	1/20	20/20	0.939	161.6		1.0	0/20	20/20	0.935	278.3
cmplt-	0.0	0/1	0/1	N/A	2500	cmplt-	0.0	0/1	0/1	N/A	3500
525-0	0.3	0/20	20/20	0.934	423.9	735-0	0.3	0/20	20/20	0.929	559.3
	1.0	0/20	20/20	0.881	182.5		1.0	0/20	20/20	0.865	151.4
cmplt-	0.0	0/1	0/1	N/A	2500	cmplt-	0.0	0/1	0/1	N/A	3500
525-1	0.3	1/20	20/20	0.954	245.7	735-1	0.3	0/20	20/20	0.938	488.4
	1.0	0/20	20/20	0.908	111.4		1.0	0/20	20/20	0.883	179.9
rndm-	0.0	0/1	0/1	N/A	2500	rndm-	0.0	0/1	0/1	N/A	3500
525-0	0.3	8/20	20/20	0.977	527.3	735-1	0.3	1/20	18/20	0.971	1213.0
	1.0	2/20	20/20	0.951	288.6		1.0	1/20	18/20	0.951	966.3
rndm-	0.0	0/1	0/1	N/A	2500	rndm-	0.0	0/1	0/1	N/A	3500
525-1	0.3	19/20	20/20	0.998	40.6	735-2	0.3	3/20	16/20	0.967	1507.0
	1.0	10/20	20/20	0.957	53.8		1.0	1/20	20/20	0.916	507.9

The results are shown in Tables 1 and 2.

As we mentioned, rule (5) is equal to rule (4) when δ is 0.0. The protocol with this setting, therefore, terminates only when an optimal solution is found (otherwise, it is forced to terminate when the cutoff round $100n$ is reached). However, in the experiments, we observed that the protocol with 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 a near-optimal solution (an infeasible solution giving a tight upper bound) very quickly, they eventually fell into a loop where some agents clustered and dispersed 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.Quality is kept at a reasonable level, suggesting that by using the protocol with those settings, the 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

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

3 complete topology instances of the hand-made problem suite (cmplt-525-0, cmplt-735-0, and cmplt-735-1) 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 instances, we can see that when δ increases, both Avg.Cost and Avg.Quality are reduced. This means that increasing δ may generally influence the agents to rush to reach a compromise of lower quality. This indicates 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 and apply rule (5) independently when it is time to update μ_j. These "noisy" updates by many agents generally accelerate the diversification of the agents' views on multipliers and may force the agents to rush to reach a compromise of lower quality.

Conclusion

We presented a new formulation of distributed task assignment and a novel distributed solution protocol, whereby the agents solve their individual optimization problems and coordinate their locally optimized solutions. Furthermore, to control the performance of the protocol, we introduced a parameter δ that controls the degree of noise mixed with an increment/decrement in a Lagrange 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 indicated that the parameter may allow us to control tradeoffs between the quality of a solution and the cost of finding it.

We believe in the potential of this approach because, unlike local search algorithms such as SA or GA, the Lagrangian relaxation method can provide both upper and lower bounds for the optimal value. Actually, our protocol can be used to compute a lower bound by setting δ to a non-zero value and adding up the objective values of final assignments; an 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 the Lagrange multiplier vector.

References

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

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

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

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

Guignard, M., and Kim, S. 1987. Lagrangean decomposition: A model yielding stronger Lagrangean bounds. *Mathematical Programming* 39:215–228.

Hirayama, K., and Yokoo, M. 1997. Distributed partial constraint satisfaction problem. In *Proc. 3rd CP*, 222–236.

Hirayama, K., and Yokoo, M. 2005. The distributed breakout algorithms. *Artificial Intelligence* 161(1–2):89–115.

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

Martello, S.; Pisinger, D.; and Toth, P. 2000. New trends in exact algorithms for the 0-1 knapsack problem. *European J. of Operational Research* 123:325–332.

Modi, P. J.; Shen, W.-M.; Tambe, M.; and Yokoo, M. 2003. An asynchronous complete method for distributed constraint optimization. In *Proc. 2nd AAMAS*, 161–168.

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

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

Petcu, A., and Faltings, B. 2005. A scalable method for multiagent constraint optimization. In *Proc. 19th IJCAI*, 266–271.

Smith, R. G. 1990. The contract net protocol: high-level communication and control in a distributed problem solver. *IEEE Trans. on Computers* 29(2):1104–1113.

Yokoo, M.; Durfee, E. H.; Ishida, T.; and Kuwabara, K. 1998. The distributed constraint satisfaction problem: formalization and algorithms. *IEEE Trans. on Knowledge and Data Engineering* 10(5):673–685.