An Intelligent Brokering System to Support Multi-Agent Web-Based 4th-Party Logistics

Hoong Chuin LAU Yam Guan GOH
The Logistics Institute Asia Pacific, National University of Singapore
lauhc@comp.nus.edu.sg g0201003@nus.edu.sg

Abstract

An intelligent agent-based framework that supports fourth-party logistics (4PL) operations on the web is proposed in this paper. In our system, customers specify job requests over the web dynamically. An e-Market Place allows intelligent third-party logistics (3PL) agents to bid for customers' job requests. The intelligence lies on the e-Market Place to optimally decide which agents' bids should be satisfied based on a set of pre-determined factors (pricing, preferences and fairness). We model the underlying brokering problem as a Set Packing Problem (SPP), which is an NP-hard optimization problem. An iterative greedy approximation algorithm is proposed to solve the SPP, and experimental results show its effectiveness against the classical greedy method proposed by Chvatal.

1. Introduction

The term "fourth-party logistics" (4PL) is a trademark owned by Accenture (formerly known as Andersen Consulting) [1]. It refers to the evolution in logistics from suppliers focused on warehousing and transportation (commonly known as third-party logistics providers or 3PL) to suppliers offering a more integrated supply chain solution. With the advent of the Internet era in the late 1990s, 4PL grew in popularity as web and other IT technology provided the leverage.

With the Internet, literally all data of logistics activities such as customer demands, warehousing, inventory and transportation information can be made available electronically. This opens the possibility for logistics activities to be streamlined and optimized across company boundaries on a macro basis. For this vision to be realized, 3PL operators will form alliance or consortium, which is to be managed by a 4PL provider whose role is to provide optimized plans and schedules for the member companies. One such management function is to efficiently and intelligently match the demands of customers with the available capacities of the 3PLs in real-time.

In this paper, we propose an intelligent and robust system for the customers, 3PL agents and 4PL to work seamlessly together over the web. In our system, customers submit their job requests via the web to an e-Procurement server. The consolidated jobs at the e-Procurement server will then be transmitted to the e-Market Place server, which serves as a broker that allows 3PL agents to submit bids on combinations of customers' jobs. Agents may join and leave the system dynamically. Successful bids are then communicated to the 3PL agents and the customers dynamically.

The system is developed using a hybrid of technologies, including web technologies, multi-agent systems, and optimization. Central to the intelligence of this system is a model for solving a complicated logistics optimization problem faced by the 4PL provider. The underlying computational problem is in determining the winning set of bids based on 3 factors, namely: *pricing*, *preference* and *fairness*. This problem can be modeled as a *Set Packing Problem* (SPP). The proposed algorithm to solve the SPP is based on an iterative greedy method whereby each iteration builds a new cover based on part of the best solution obtained so far. In addition, the effectiveness of two different heuristic approaches (deterministic and probabilistic) for divergent measure and partial cover construction used in the iterative greedy algorithm were compared and investigated. Experimental results obtained had shown that these iterative greedy approaches yield better solutions than the classical greedy algorithm, indicating the effectiveness of our approach.

2. Literature Review

The web technology used in deploying the web-based enterprise systems and applications is based on *Java™ 2 Platform, Enterprise Edition* (J2EE™) [4], which simplifies enterprise applications by developing them on standardized, modular components, by providing a complete set of services to those components, and by handling many details of application behavior automatically without complex programming.

Multi-agent systems have its root in distributed artificial intelligence [10]. An intelligent agent is a computational entity (such as a software program) that is autonomous in that it operates intelligently and rationally

in an environment. In this multi-agent framework, agents interact with one another to cooperate in solving complex logistics problems. In our earlier related work [7] for example, we represent each 3PL by an agent, and 4PL as a supervisor agent.

In [3], the authors modeled the intelligent matchmaking behavior of an agent as a constraint satisfaction problem. In [5], a two-layered multi-agent framework is proposed for brokerage between buyers and sellers. The brokerage is processed in two layers for efficient linking between buyers and sellers: the competition layer and the constraint satisfaction layer. In the competition layer, the seller agents, as requested by the buyer agents, are selected through the competition in the competition layer. In the constraint satisfaction layer, the CSP solver finds an optimal solution by choosing the best brokerage to satisfy various preferential requirements for users. In [6], the authors propose that each bid and job corresponds to a point in a multi-dimensional attribute space and the "distance" between each pair of bid and job request is a measure of how each pair matches. However, in our implementation, the term "gain" is used to represent "distance", as by its intuitive meaning, a higher value of "gain" will denote better matches, and vice versa.

In the combinatorial auction setting that has been discussed in [9,10] and many other papers, each bidder can bid on combinations of indivisible items, and her bids are implicitly joined with non-exclusive OR, meaning that any number of her bids can be accepted. Winner determination in this case is the problem of deciding which bids win so as to maximize the sum of the bid prices, under the constraint that every item is allocated to at most one bid. The underlying problem is the weighted set packing problem (SPP), which is known to be *NP*-complete. Although combinatorial auctions are a subject of intense study, much of it has focused on algorithms for finding an optimal (or approximately optimal) set of winning bids subject only to pricing considerations. In this paper, we extend the considerations to multiple criteria; and particularly, we focus our attention on pricing, customer preferences and fairness (load balancing). These factors may not be of importance in combinatorial auctions in general, but plays an important role in 4PL operations.

In [11], the authors consider two other closely studied relatives of SPP: the *Set Partitioning Problem* (SPA) and the *Set Covering Problem* (SCP). A specific algorithmic implementation in SCP is presented in [8]. It proposes an iterative greedy solution approach for solving large scale SCP with application to airline crew scheduling, which our algorithm is largely based on, except that our application is based on SPP. Comparatively little attention has been paid to methodical evaluation and experimental comparison of these algorithms. In this paper, besides comparing our method with a plain naive greedy algorithm, we will also be comparing four different variants of an iterative greedy method. Extensive computational experiments were conducted to determine the best method in terms of solution quality.

3. System Overview

In this section, we give an overview of the proposed system. The **Administrator** manages the e-Market Place and hence the brokering process. An **Agent** represents a 3PL provider. The **Customer** refers literally to one who submits a **job request**, which comprises a customer ID, job specification, and customer's preferences for pricing and 3PL agent. A **bid** consists of a set of job requests that a particular agent can fulfill. Figure 1 gives an overview of the proposed system.

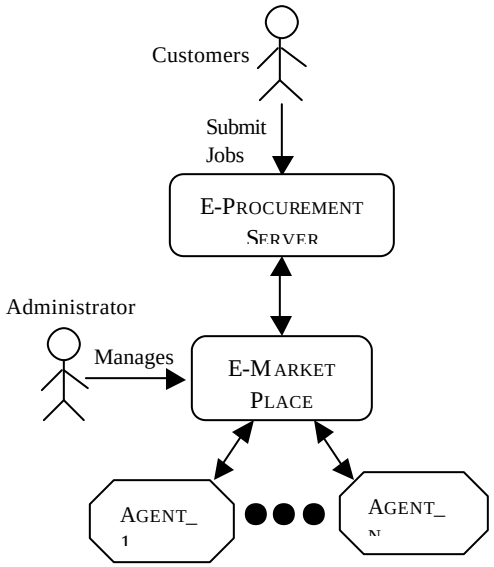

Figure 1. System Overview

The following is a normal scenario that describes the entire workflow of the system:

1. Customer submits a job request to the system.
2. Agent registers itself with the e-Market Place.
3. Administrator connects to the e-Procurement Server to download newly arrived jobs.
4. These jobs are broadcast to all registered agents.
5. Registered Agents submit their bids.
6. Administrator will intelligently assign jobs to bids.
7. Detailed information on the assignments is pushed to the agents and (via the e-Procurement Server) to the Customers.
8. Agent can decide to de-register with the e-Market Place at any time.

4. The Broker Model

In this section, we propose the broker model for the e-Market Place. It focuses on modeling 4PL operations, while ignoring the game-theoretic incentive aspects commonly discussed in combinatorial auctions papers.

4.1. Set Partitioning Problem (SPP)

SPP is a well-studied combinatorial optimization problem. Given a ground set M of elements and a collection N of subsets with non-negative weights, the problem is to find the largest weighted sub-collection of N that are pairwise disjoint, such that each element in M can appear in at most one subset.

To formulate SPP as an integer program, we can view SPP as a m-row, n-column matrix, zero-one matrix (a_{ij}), and an n-dimensional integer vector (w_j), and the problem consists of finding a subset of columns the rows and that they are pairwise disjoint and having the largest total weight. Defining decision variable $x_j = 1$ if column j is in the solution and $x_j = 0$ if otherwise, the SPP is formulated as a constrained optimization problem as follows:

$$\text{maximize} \sum_{j \in N} w_j x_j$$

$$\text{subject to} \begin{cases} \sum_{j \in N} a_{ij} x_j \le 1 \; \forall i \in M & (1) \\ x_j \in \{0,1\} \; \forall j \in N & (2) \end{cases}$$

where $M = \{1, \ldots, m\}$ and $N = \{1, \ldots, n\}$ and a row i is covered by a column j if the entry a_{ij} is equal to 1, otherwise it is 0. Equations (1) ensure that each row is covered by at most one column while the integrality constraint (2) ensures that x_j is equal to 1, iff column j is in the solution.

SPP is directly related to bid allocation in the following sense. The rows represent the submitted jobs to be fulfilled, and the columns represent the bids for a subset of rows (jobs). The problem is to find a set of bids having maximum weight, which covers a given set of jobs not more than once. The weights w is to be derived based on three factors (*Pricing*, *Preference* and *Fairness*),

Weight w is representative of how good a bid is for it to be included in the solution. Hence, it is important that the value of w for a bid is a correct and accurate representation of how good the match is for all the jobs covered under the bid.

We introduce the terminology "gain" g that is representative of how well an agent's bid and a job match. In formal terms, g_{ij} represents how good a match is between bid_j and job_i. Finally, we can get w value for each column by summing up all the g values for each respective column. In other words, w_j for a particular bid_j is the summation of all g_{ij} for all job_i contained in bid_j. Mathematically, it can be expressed as:

$$w_j = \sum_{i \in M} g_{ij} \text{ , such that } a_{ij} = 1$$

4.2. Gain Resolution

Gain g determines how good a match between a bid and a job. Hence, in order for values of g to provide an accurate reflection, it will have to be dependent on the aggregation of these three factors: *Pricing*, *Preference* and *Fairness*.

1. *Pricing* – The allocation of the bids should be allocated in a manner so that the prices that the customers have to pay for servicing their job requests are minimized.
2. *Preference* – Customers have the freedom to choose their preferred 3PL agent to fulfill their job requests.
3. *Fairness* – There must be fairness measure to ensure load balancing of jobs among the 3PL agents.

Each factor is associated to its own weight *wt* that determines its relative importance among other factors. This weight component in our system provides the flexibility in determining the relative importance of each factor with regards to the rest. The value of g_{ij} is computed as follows:

$$g_{ij} = \frac{g_{ij}(Pricing) + g_{ij}(Preference) + g_{ij}(Fairness)}{wt_{ij}(Pricing) + wt_{ij}(Preference) + wt_{ij}(Fairness)}$$

In general, for an arbitrary set F, the equation is formulated as follows:

$$g_{ij} = \frac{\sum_{f \in F} g_{ij}(f)}{\sum_{f \in F} wt_{ij}(f)}$$

The values for $wt_{ij}(f)$ are range from 0 to 1 inclusive. The larger the value of $wt_{ij}(f)$, the more important it is for factor f, and vice versa.

4.3. Factors Resolution

The following notations are used in determining the values for each factor.

$Wt_i(Pricing)$: weight assigned to cost by a customer for a particular job_i.
$PMax_i$: maximum price a customer is willing to pay for job_i to be fulfilled.
$PMin$: minimum price an agent is allowed to bid.
$PBid_{ij}$: price of bid_j which an agent offered for job_i.

$Wt_i(Pref)$: weight assigned to preference by a customer for a particular job_i.

$PrefAgent_{ai}$: indicates if job_i's preferred agent is $agent_a$.

$Wt(Fairness)$: weight assigned to fairness by the 4PL provider.

$MaxJobs$: maximum upper threshold of jobs an agent can fulfill.

$JobsDone_a$: total jobs that $agent_a$ had so far fulfilled.

4.3.1. Factor Pricing

The price that a customer has to pay for servicing his job requests should be taken into consideration in determining the allocation of bids. In our system, we will allow customers to specify their maximum price that they are willing to pay in fulfillment for each of their job requests. They also have to specify the weight value for each job request submitted in order to signify its importance with respect to other factors. On the other hand, the Administrator sets the minimum price an agent can bid for all jobs.

The gain value for the Pricing factor, $g_{ij}(Pricing)$, is formulated as:

$$g_{ij}(Pricing) = Wt_i(Pricing) \times \left[1 - \frac{PMax_i - PBid_{ij}}{PMax_i - PMin}\right]$$

4.3.2. Factor Preference

Our system also allows customers to specify their preferred 3PL agent to fulfill their job requests. Similarly, they also have to specify the weight value attached to this factor. In addition, $PrefAgent_{ai}$ is subject to:

$$PrefAgent_{ai} = \begin{cases} 1, \text{if } job_i \text{ prefers } agent_a \\ 0, \text{otherwise} \end{cases}$$

The gain value for the Preference factor, $g_{ij}(Pref)$ is formulated as:

$$g_{ij}(Pref) = Wt_i(Pref) \times PrefAgent_{ai}$$

4.3.3. Factor Fairness

A fairness measure in the distribution of jobs among the agents is enforced by the Administrator. The Administrator sets the weight for the *Fairness* factor and the maximum upper threshold of jobs an agent can fulfill. The gain value for the Fairness factor, $g_{ij}(Fairness)$ is formulated as:

If $MaxJobs \geq JobsDone_a$,

$$g_{ij}(Fairness) = Wt(Fairness) \times \left[\frac{MaxJobs - JobsDone_a}{MaxJobs}\right]$$

Otherwise, $g_{ij}(Fairness) = 0$.

5. Algorithm

5.1. Notations

In order to describe our solution approach, we use the following notations:

Indexes i, j denote a generic row and column, respectively. Set S denotes the set of columns in the working solution.

$$S = \{ j \mid x_j = 1 \text{ for some } j \in N \}$$

Candidates has the following meaning:

$$Candidates = \{ j \mid j \notin S \text{ and } j \in N \}$$

value(S) returns the objective value of SPP and it is formulated as follows:

$$value(S) = \sum_{i \in N} \sum_{j \in S} a_{ij} w_j$$

Conflict(j) denotes the set of columns which do not contain any overlapping rows with respect to column *j*. That means if column *j* is contained in the solution, the set of columns in *Conflict(j)* cannot be included in the solution. Moreover, if *Conflict(j)* = {}, this means that we can include column *j* as part of the solution. Another interesting property is: if a column j' is in the set *Conflict(j)*, then there is also a column *j* in the set *Conflict(j')*. For this situation, columns *j* and j' are said to be in "conflict".

$$Conflict(j) = \{ j' \mid a_{ij} = 1 \text{ and } a_{ij'} = 1 \text{ for some } j' \in N \}$$

AJobs(j) returns the set of rows that are covered only by column *j*. This set of rows *i* returned can be considered the opportunity cost if column *j* is not part of the solution as they will not be covered by the rest of the columns.

$$AJobs(j) = \{ i \mid a_{ij} = 1 \text{ and } a_{ij'} = 0 \text{ for all } j' \in N \}$$

Unfilled(j) returns the set of rows that cannot be covered if column *j* is in the solution.

$$Unfilled(j) = \{ i \mid i \in AJobs(j') \text{ for some } j' \in Conflict(j) \}$$

5.2. Overall Method

Given a problem instance, the algorithm extracts an initial core from the set of columns given in the input. The algorithm consists of the iterated application of the following three steps:

1. An approximate solution to the actual *SPP* core constructed by a greedy construction heuristic.
2. A local search optimization algorithm is applied to the resulting solution.
3. Some columns that occur in the best solution found in all iterations up to now are selected for forming the initial partial solution for the next iteration.

The algorithm *SPP* is illustrated below, where **Sbest** represents the best cover found so far and **S** denotes the actual partial cover.

```
FUNCTION SPP()
    S ← {};
    SBest ← ComputeCover();
    SConfirm ← AssignNonConflict();
    FOR 1 … number_of_iterations DO
            S ← GREEDY(S);
            S ← LOCAL_SEARCH(S);
            IF (value(S) = value (SBest)) THEN
                    SBest ← S;
            ENDIF
            S ← PARTIAL_COVER(S, SBest);
    ENDFOR
    RETURN SBest + SConfirm;
```

5.3. Greedy Heuristic

Our greedy heuristic *GREEDY* is given below. The algorithm constructs a solution (a cover), starting from a (possibly empty) partial cover **S**. Columns are added to (resp. removed from) **S** until no columns can be added without conflicting with the rest of the columns in **S**.

```
FUNCTION GREEDY(set S)
    WHILE (NOT Conflict(S, Candidates)) DO
            S ← S + SelectAdd(Candidates);
            IF (Remove()) THEN
                    S ← S – SelectRemove(S);
            ENDIF
    ENDWHILE
    RETURN S;
```

Conflict(**S**, *Candidates*) returns true when there is a column *j* in *Candidates* that does not conflict with any columns in **S**, otherwise it returns false. Following that, *Select_Add*(*Candidates*) selects a column *j* with the largest associated weight that does not conflict with any columns in **S**.

Divergent measure is implemented in *GREEDY* to escape the trap of local optimality through the use of *Remove*() and *SelectRemove*(**S**). *Remove*() determines whether columns should be removed from **S**. With a probability of 0.3, it returns true, otherwise false.

Finally, *Select_Remove*(**S**) selects a "bad" column in **S** for removal. There are basically two different approaches to accomplish this task. They are:

1. Deterministic – Returns a column *j* in **S** with maximum *UnfilledJobs(j)*
2. Probabilistic – Selection of a column *j'* from **S** is based on the following probability that is associated with each column:

$$\text{P(selecting } j') = \frac{UnfilledJobs(j')}{\sum_{j \in S} UnfilledJobs(j)}$$

For the second approach, the columns in **S** are evaluated in descending order of the column's probability, and its returns the first column that is evaluated to be true based on its probability.

5.4. Local Search

The local optimization procedure LOCAL_SEARCH improves a solution by introducing a better column into the solution not found in set **S**. The algorithm LOCAL_SEARCH is illustrated below. **TempSBest** represents the temporary best cover found so far, while **TempS** represents the temporary working partial cover.

```
FUNCTION LOCAL_SEARCH(set S)
    TempSBest ← S;
    FOR EACH COLUMN j in Candidates DO
            TempS ← RemoveConflict(S, j);
            TempS ← TempS + {j};
            TempS ← GREEDY(TempS);
    IF (Value(TempS) ≥ Value(TempSBest)) THEN
                    TempSBest ← TempS;
    ENDIF
    ENDFOR
    RETURN TempSBest;
```

The RemoveConflict(**S**, *j*) method remove the set of columns in *Conflict(j)* from set **S**. Essentially, each time a column *j* in *Candidates* is introduced into solution **S**, the new value of *Value*(**S**) is checked against the highest obtained *Value*(**S**) so far. If the *Value*(**S**) obtained is higher, as a result of the introduction of column *j*, then *j* will be adopted into solution **S**.

5.5. Selecting SPP Partial Cover

In the first iteration of *SPP*, the heuristic *GREEDY* constructs a cover starting from the empty set; in the following iterations, *GREEDY* builds a cover starting from a subset of the best cover found so far. For a column *j*, we keep track of the number *chosen(j)* of times that *j* has been part of a best solution. *PARTIAL_COVER* is as follows:

```
FUNCTION PARTIAL_COVER(set S, set SBest)
    E ← {};
    FOR EACH COLUMN j in SBest DO
        IF (AJobs(j) ≥ UnfilledJobs(j)) THEN
            E ← E + {j};
        END IF
    ENDFOR
    SELECT_PARTIAL_COVER(S, E);
    RETURN S;
```

SELECT_PARTIAL_COVER considers the set **E** of so-called elite columns, consisting of those columns *j* of the best solution **SBest** such that $AJobs(j) \geq UnfilledJobs(j)$. Then, *SELECT_PARTIAL_COVER* selects from **E** the set of columns having low *chosen(j)* based on one of the following methods:

1. Deterministic – Set of columns $j\not\in$ from **E** is selected if:
$$chosen(j\not\in) < \frac{\sum_{j\in E} chosen(j)}{E.size() \times c}$$
where **E**.size() is the number of elements in **E** and *c* is a constant which is arbitrarily set to 10 in our implementation.
2. Probabilistic – Selection of a column $j\not\in$ from **E** is based on the probability:
$$\text{P(selecting } j\not\in\text{)} = 1 - \frac{chosen(j')}{\sum_{j\in E} chosen(j)}$$

6. Experimental Results

We have shown that the divergent measure and selection of partial cover for our algorithm can be implemented by either a probabilistic or deterministic approach. By enumerating these different combinations of approaches, we can have up to four variant algorithms as follows:

1. **P**robabilistic Divergent Search & **P**robabilistic Partial Cover (*PP*)
2. **D**eterministic Divergent Search & **D**eterministic Partial Cover (*DD*)
3. **P**robabilistic Divergent Search & **D**eterministic Partial Cover (*PD*)
4. **D**eterministic Divergent Search & **P**robabilistic Partial Cover (*DP*)

We compare experimentally the above set of algorithms with Chvatal's greedy algorithm [2] with respect to the solution quality. In our implementation of Chvatal's algorithm, we select each column based on the highest weight per row covered by the inspected column.

We generated test cases as follows. For simplicity, we assume that all jobs do not have any stated preferred agent and the *Fairness* factor is set to "Not Important" (i.e. 0). Hence, the value of Gain *g* is totally determined by the *Pricing* factor. The *Pricing* factor is calculated based on randomly generated bid prices that range from *PMin* to $PMax_i$, and $Wt_i(Pricing)$ is assigned a random value from 0 to 1. The results of each set of experiments are based on runs made on 50 different data sets, with total iterations fixed at 100 for the iterative greedy approach.

In the table shown in Figure 3, the first column contains the abbreviation of the algorithms used. The second column displays the average total weight of the solutions based on the 50 runs and the last column denotes the number of best solutions obtained among the five algorithms. In Figures 2 and 3, the experiments are conducted with each test case containing 100 columns (bids) and 200 rows (jobs). The number of rows each column can cover is randomly generated from 1 to 10.

Experiments	Ave Total Wt.	Best Sol Out of 50
Chvatal	65.14	2
PP	69.57	7
DD	69.07	10
PD	70.37	22
DP	69.24	10

Figure 2. Experimental Evaluation

In addition, we explore experimentally how the solution quality changes with respect to the number of iterations for our greedy heuristics. In the following graph, the x-axis represents the number of iterations (up to 1000) and the y-axis, the average total weight obtained for 50 runs.

The results of the experiments indicate that our iterative greedy approach produces much superior results compared to the Chvatal's classical greedy approach. All our four variants of the iterative approach are closely matched in terms of solution quality, although we observe

that *PD* outperforms the rest of the algorithms in terms of number of best solutions produced.

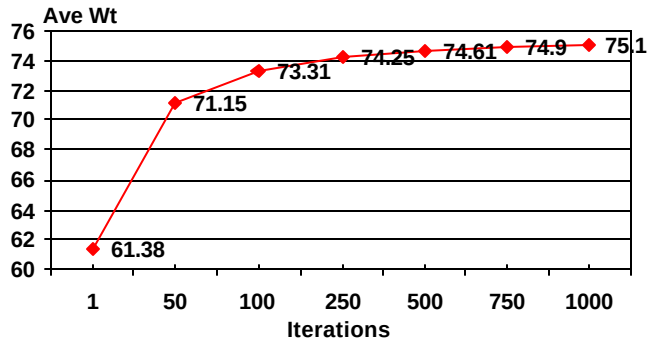

Figure 3. Iterations Results

7. System Execution

In this section, we briefly describe our system in execution.

Firstly, the e-Market Place Server can be initialized via the user interface as shown in Figure 4 (which is self-explanatory):

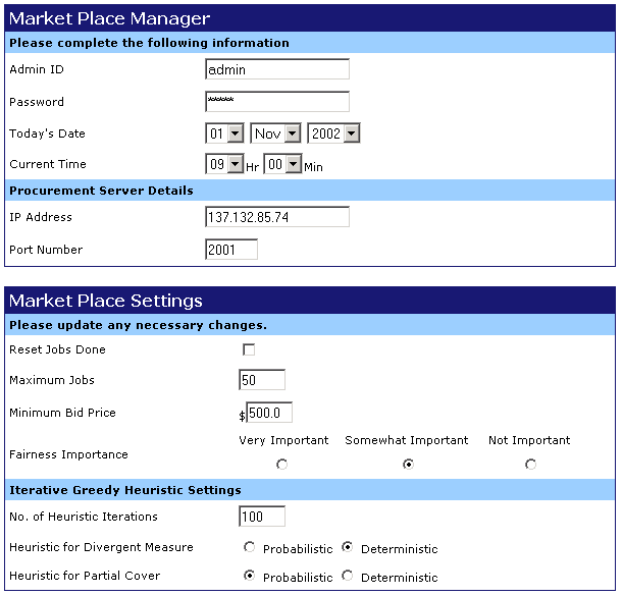

Figure 4. e-Market Place Server User Interface

Customers will submit their job requests to the e-Procurement Server through a web browser (see Figure 5). In our implementation, the customer can input preferred pricing and agent for their job requests; the degree of importance of these factors are specified through the radio boxes: "Not Important", "Somewhat Important" and "Very Important". Internally, we assign a value of 0 for "Not Important", 0.5 for "Somewhat Important" and 1 for "Very Important". These values are used in the computation of gain *g*, which was discussed in Section 4.

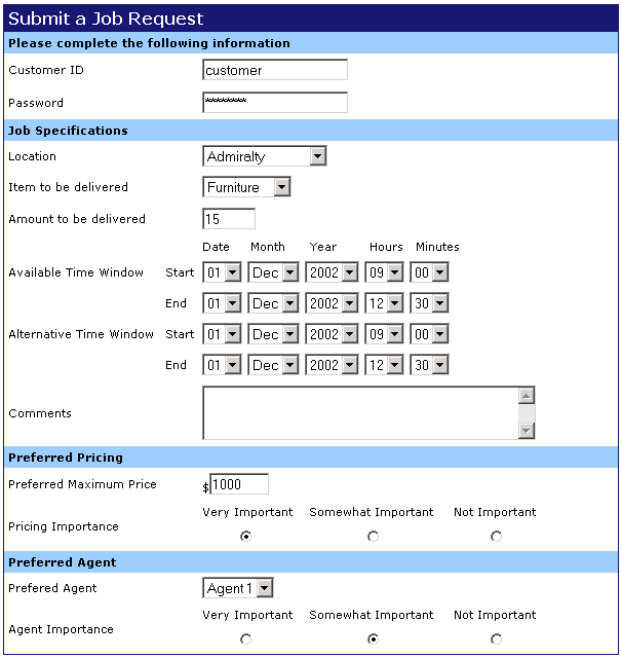

Figure 5. Job Submission User Interface

The job requests are stored in the database at the e-Procurement Server. Once the e-Market Place Server establishes connection with the e-Procurement Server to commence the bidding process, these jobs will be uploaded to the e-Market Place Server and in turn the e-Market Place will broadcast the jobs to the registered 3PL agents. After the bids allocation process had completed, jobs allocation results will be downloaded from the e-Market Place to the e-Procurement Server. An example of bid allocation is shown in Figure 6. Here, the rows represent customer jobs while the columns represent agent bids. A (bid, job) pair is colored by either a dark or a grey-colored cell, where an orange cell means that the job has been successfully allocated to the bid, while a grey cell means it has not[1]. The value within each cell stores the gain value g_{ij}, for the corresponding bid_j and job_i. One may obtain details by clicking on the value in the cell, which will pop-up a window displaying information such as the weight *wt* for each of the factor.

[1] In case the black & white printed page cannot distinguish the two shades, Bids 100, 102, 103, 105, 111, 113, 118 and 119 are successfully allocated while the rest are not.

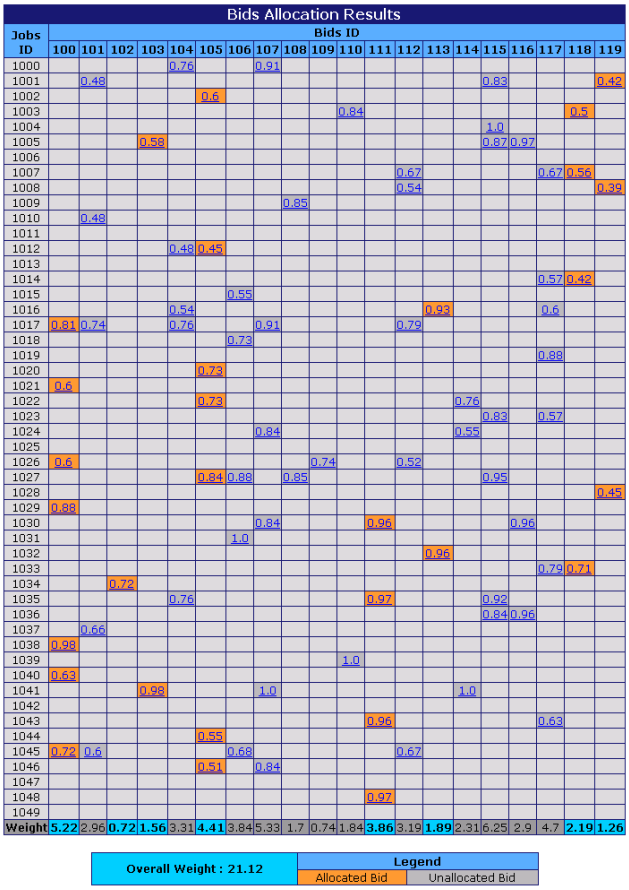

Figure 6. Bid Allocation User Interface

8. Conclusion

In this paper, we designed and implemented a J2EE™ based system for an intelligent 4th-party logistics operator coordinating the activities of many 3rd-party logistics providers against online job demands. We introduced a novel heuristic method for solving the underlying logistics optimization problem. The results of the experiments had indicated that our iterative greedy approach generally produce better solution quality than the classical Chvatal's greedy approach. We also concluded that the *PD* approach is generally more effective among the algorithms evaluated. Future works are given as follows.

In our system architecture, the 3PL agents are served by a 4PL provider (e-Market Place) in the procurement of customers' jobs and a combinatorial auction takes place where by the agents bid any combination of jobs intelligently with respect to its available resources. Currently in our implementation, we do not model game-theoretic aspects of agent behavior. A possible future improvement is to incorporate that into the model.

Another improvement is to improve the bids allocation algorithm. Current implementation only takes into consideration on pricing, fairness and preference factors. Other factors could be considered. New heuristic methods such as genetic algorithm and probabilistic search to solve SPP can be explored and benchmarked against the current implementation.

References

[1] D. Bauknight and J. Miller, "Fourth Party Logistics: The Evolution of Supply Chain Outsourcing", *CALM Supply Chain & Logistics Journal*, (1999)

[2] V. Chvatal. "A greedy heuristic for the set-covering problem", *Mathematics of Operations Research*, 4(3):233--235, (1979)

[3] E. Freuder and RWallace, "Suggestion Strategies for Constraint-Based Matchmaker Agents", In *Principles and Practice of Constraint Programming, CP'98*, (1998)

[4] J2EE Documentation, http://java.sun.com/j2ee/

[5] J. Jung and G Jo, "Brokerage between buyer and seller agents using Constraint Satisfaction Problem models", *Decision Support Systems*, Volume 28, Issue 4, (2000)

[6] P. Keskinocak, R. Goodwin, F. Wu, R. Akkiraju, S. Murthy, "Decision Support for Managing an Electronic Supply Chain", *Electronic Commerce Research*, Volume 1 (2001)

[7] H. C. Lau and Y. T. Lo, "A Multi-Agent Framework for Supporting Web-based Intelligent Fourth-Party Logistics", Proc. *Int'l Conf. on Integrated Logistics*, Singapore (2001)

[8] E. Marchiori and A. Steenbeek, "An Evolutionary Algorithm for Large Scale Set Covering Problems with Application to Airline Crew Scheduling", *EvoWorkshops*, (2000)

[9] T. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135:1-54, 2002.

[10] S. de Vries and R. Vohra, "Combinatorial Auctions: A Survey", *Technical Report, Northwestern University*, (2000)

[11] G. Weiss (ed.), *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, Reading, MIT Press, Massachusetts, (1999)