The Pennsylvania State University

The Graduate School

Department of Industrial & Manufacturing Engineering

# MULTI-AGENT AND MARKET BASED DYNAMIC OPTIMIZATION AND ITS EXTENSIONS TO DISTRIBUTED SUPPLY CHAIN PROCUREMENT PLANNING PROBLEMS

A Thesis in

Industrial Engineering

by

Kaizhi Tang



Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

May, 2005

We approve the thesis of Kaizhi Tang.

Date of Signature

| | |
|---|---|
| Soundar R. T. Kumara<br>Distinguished Professor of Industrial Engineering<br>Professor of School of Information Sciences and Technology<br>Professor of Computer Science and Engineering<br>Thesis Advisor<br>Chair of Committee | |
| Kalyan Chatterjee<br>Distinguished Professor of Economics and Management Science | |
| Natarajan Gautam<br>Associate Professor of Industrial Engineering | |
| E. Emory Enscore<br>Professor Emeritus of Industrial Engineering | |
| Richard J. Koubek<br>Professor of Industrial Engineering<br>Head of the Department of Industrial Engineering | |

# Abstract

This thesis discusses distributed computational methodologies used for solving supply chain procurement planning problems with information systems naturally distributed at different locations. The planning problem is heavily influenced by transportation costs, whose calculation is based on a stochastic environment with random service and travel times. The main contributions of this thesis are the computational methodologies that solve the problem under different conditions. All these methodologies are developed in the context of supply chain procurement planning, but they are not necessarily restricted to this context. They can be modified and adapted to other distributed planning and scheduling problems.

The Double Auction Market Mechanism (DAMM) is an innovative distributed negotiation mechanism for supply chain procurement planning. It is developed to achieve a common objective among multiple individual decision makers by modeling their respective interests in a virtual market for trading transportation tasks. The trading mechanism emulates the double auction institutions of a real market. For trading, double auction institutions enable exchange of transportation tasks among a transportation company and its trucks, which are modeled as software agents. An agent's decision making among market interactions is based on a set of stochastic rules, which consider its self-interest with the support of transportation cost evaluation. Iterated task exchange among agents drives the task allocation to Pareto optimality, which is derived from the

concept of market equilibrium. DAMM is an any-time negotiation protocol, so it is useful for dynamic optimization and contingency planning.

A Coordinated Reinforcement Learning (CRL) method is developed to refine the distributed transportation plans generated by distributed heuristics based on DAMM. Through analyzing the drawbacks of the distributed double auction heuristic due to its strict market rules, we enable the task agents in the virtual market to return to their previous stages and reconsider their decisions for market trading interactions. This is realized in the framework of reinforcement learning (RL). To be adaptive in the framework of RL, the task allocation derived from the transportation plans is defined as a *state*; the collective proposals of the task agents are defined as an *action*; the cost saving during one round of market interaction is defined as a *reward*; the update of transportation plans resulting from updating task allocation is defined as a *state transition*. CRL with tabular representations effectively refines the distributed transportation plans generated by DAMM.

A meta-heuristic enabled multi-agent optimization architecture is developed for dynamic supply chain procurement planning. This development is based on analyzing drawbacks of DAMM and the curse of dimensionality of CRL. To escape from local optimality towards a higher quality solution, we introduce meta-heuristics over agent interactions to advise agents' searching processes. We mainly propose the variable neighborhood search meta-heuristic (VNS-MH) over the distributed market based heuristic (DMBH), which is based on DAMM. VNS-MH not only performs better on achieving optimality than DMBH, but also requires less memory and computational time than its predecessor – coordinated reinforcement learning (CRL).

A multi-player game with multiple stages is designed to model task delegations among multiple self-interested transportation companies. The delegations are useful for dealing with infeasible or extremely expensive tasks because of transportation companies' limitations on vehicle capacities or restrictions on time windows. Task delegations have to address the self-interests of transportation companies. This turns into a game with multiple stages, each of which incurs a round of double auction interactions. In this game, multiple players, representing a set of transportation companies, aim to maximize their own payoffs summed over multiple stages. By playing the game repeatedly, we develop an evolutionary method, combining reinforcement learning and fictitious playing, to seek a solution. This method is based on the equilibrium concept of evolutionary stable strategy to successfully achieve a solution to the complex game.

**Keywords:** double auction, market mechanism, supply chain procurement, transportation planning, multi-agent system, contingency planning, reinforcement learning, meta-heuristic, variable neighborhood search, dynamic optimization, distributed heuristic, multi-stage game, evolutionary method, fictitious playing, evolutionary stable strategy

# Table of Contents

# List of Tables

# List of Figures

# Acknowledgments

Among all the acknowledgements, first of all, I would like, from the bottom of my heart, to thank my Lord, Jesus Christ, who made, supports and saves my life. He saved me from the eternal punishment and saves me from daily sins, evils and attempts. He also gives me energy to live bravely in this world. He bestows an ultimate meaning to my life and hears my prayer. He loves and attracts me to live as his beloved son toward a holy goal. In a word, He made my life not only possible, but also promising.

I am deeply indebted to my advisor, Dr. Soundar Kumara, Distinguished Professor of Industrial and Manufacturing Engineering, The Pennsylvania State University, for his constant guidance, support, and encouragement throughout my research and thesis writing. It is he who introduced me to the wonderful interdisciplinary area of distributed information systems, transportation optimization, complex systems and macroeconomics theory. This area interests me a great deal. He has shown me a great deal of insights in those areas, given me confidence during my research and shown great patience in enabling me to explore new ideas and to enjoy my research work. I would also like to thank other committee members Dr. Kalyan Chatterjee, Dr. Emory Enscore, and Dr. Natarajan Gautam for their insightful guidance. Especially, I would like to thank Dr. Chatterjee for the help in the area of auction, game theory and macroeconomics, and Dr. Gautam for his profound mathematical help, as well as his insightful and critical comments during my research.

My great appreciate also goes to all the professors and graduate students in LISQ (Laboratory for Intelligent Systems and Quality) for their support and patience shown to me during my PhD study. I would like to thank Dr. Yong-Han Lee for his kind help in my work and nice cooperation during our research projects. I also thank Jindae Kim, Seogchan Oh and Xuequn Ding for their insights during our discussion.

I am profoundly thankful to my wife Xuequn Ding, for her support and care toward my research and my life in the USA. I am grateful to my parents Mr. Tang and Mrs. Jiang, and my younger sister Huiming Tang for their constant encouragement, support, patience, and hopes on me over many years.

Finally, I would like to thank my brothers and sisters in Christ. Especially, I would like to thank some Christian leaders – Rev. David Lin, Dr. Weifan Chen, Dr. Rongchii Duh – in State College Chinese Alliance Church, for their prayer, and spiritual encouragement during my difficult time. I would also like to thank Cathy for her spiritual encouragement. I would also like to thank other Christians in my Bible study group, Chinese Student, Scholar, and Family Ministries, and Chinese Church for their consistent sharing and lovely friendship.

# Chapter 1

# Introduction

A supply chain consists of multiple components including customers, distribution centers, manufacturing centers and suppliers (see Fig. 1.1) [Satapathy, 1999]. All these components are connected through two networks with their respective flows: information flow in the communication network and material flow in the transportation network. Information flow originates from customers and, with respective transformations at different levels, spreads to distribution centers, manufacturing centers, and finally suppliers. In contrast, material flow starts from suppliers and material is transported in the opposite direction to information flow. Although the fluidity of these two flows is substantially different, they interact with each other closely. In other words, the material flow is made to satisfy the requirements conveyed in the information flow; the contents in the information flow are affected by the material flow, especially after the customer orders flow to the distribution center. Supply chain planning is driven by information flow, but constrained by material flow.

With the development of Information Technology (IT), especially wireless communication technology and distributed information systems, different components and functions of supply chains can be integrated in a large-scale and information-rich network. In this network, not only have the internal processes of an organization been integrated into a supply chain [Szuprowicz, 2000], but also the external business processes between

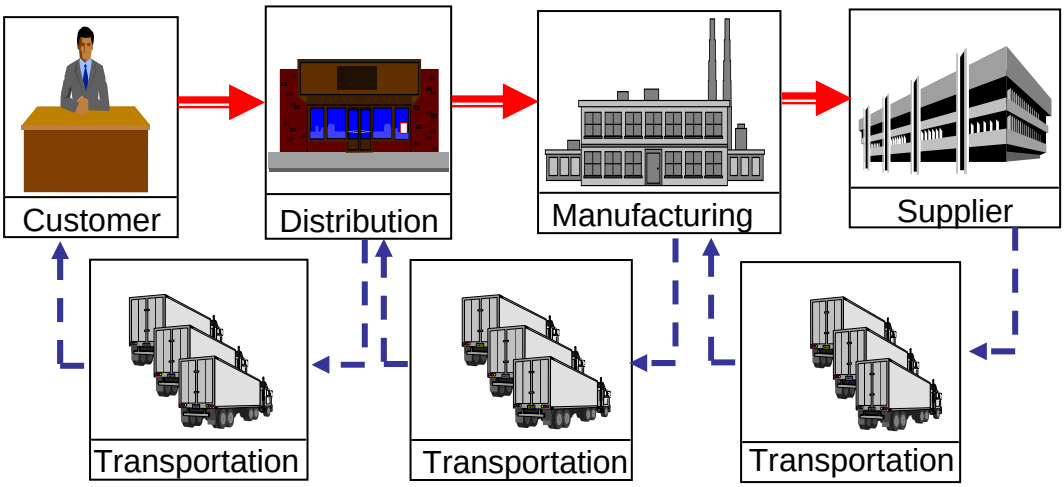


Fig. 1.1. The overview of a supply chain

different organizations have begun to form a global supply chain, especially as new business models, B2B (Business to Business), B2C (Business to Customer) and Net-Market [Olsen, 2000] are becoming an integral part of the global supply chains. The concept of supply chains has entered daily life due to developments in mobile agents and wireless communication [Brugali et al., 1998]. This integration deeply impacts supply chain transportation and provides more opportunities to improve the performance of transportation, especially through modeling dynamic and contingency environments. The integration can be related to the following processes:

- **Order acceptance process**

  Because new incoming orders can spread easily in the integrated information system, the optimized transportation plans may be subject to these new orders. With the new orders considered, the transportation plans may be further optimized by

exchanging new orders with planned orders or improving the utilization of the transportation facilities by filling vacancies after planning.

- **Order cancelation process**

  With quick notification of canceled orders, queued orders may be considered in the transportation planning and plans may be further optimized by exchanging queued orders with already planned orders.

- **Contingency planning process**

  Any contingency condition in the operational system may seriously delay or prevent delivery. With the integrated transportation planning system, alternative routines will be generated to avoid delaying or preventing delivery.

## 1.1 Drawbacks of traditional approaches

In practice, transportation planning, together with other planning and scheduling problems, is solved periodically using global optimization methodologies. The whole time horizon is divided with a fixed interval, which ranges from several minutes to several months depending on different operational environments, according to operational convenience. At the beginning of every period, information about customers' orders and available shipment schedules is collected. Based on the information, an optimization problem is formulated to minimize the total transportation cost under the constraints of the availabilities of transportation facilities and requirements of the customers' orders. This optimization problem is usually modeled as an Integer Programming (IP) problem with NP-hard properties. Problems with a small number of decision variables can

be solved with optimization approaches such as branch-and-bound and cutting planes [Salkin et al., 1989] [1], but large-scale problems are usually solved with heuristic approaches [Tan et al., 2001]. According to the solution produced by IP solvers, the transportation plans are executed as planned during the period between two consecutive planning instants, without being monitored or subject to any dynamic change in decision variables and constraints.

The previous periodical planning strategy can be classified in the category of static optimization, without modeling the dynamics of the business world. However, since the business world is always changing, static optimization of supply chain transportation planning leads to the following drawbacks.

- No further optimization on new incoming orders during the execution period. The strategy of static optimization is very conservative, only considering the assured information up to the instant of planning time. However, any plan or schedule is allowed to be updated before its execution if the information system can coordinate the execution and planning properly. The plan update can cause the overall performance to be improved. If performances across all periods are accounted, the periodical planning strategy only produces periodical optimization results, whose summation could be far away from the global optimization results.

- Unnecessary vacancies in the transportation facilities. Without a planning ability after order cancelations, the vacancies left by those canceled orders may be wasted.

[1] Commercial optimization software such as CPLEX, LINGO, GAMS and etc., provides these solvers; users just need to generate inputs and interpret the results.

Failure to consider new incoming orders may also cause this problem, though not apparently.

- Delayed or failed delivery. Transportation plans are always subject to an unpredictable and dynamic execution environment. For example, weather conditions, traffic, vehicle maintenance, vehicle breakdown, etc., may delay or prevent deliveries. Since static plans are not to be changed, these problems cannot be avoided.

From the perspective of optimization, these drawbacks fail to satisfy either dual or primal feasibility or both for the overall optimization problem, though these two feasibilities are satisfied within each periodical optimization problem. From a business perspective, all drawbacks lead to unnecessary costs if compared to the optimal solution of the global optimization problem across all periods.

## 1.2 Research Discussion

In this thesis, we address the supply chain planning problem in a dynamic and distributed environment. The details of the problem are discussed in Chapter 2. In this section, discussion goes to the relationship between periodical planning and our proposed approach.

For the perfect optimization solution to be sought, a complete global optimization problem across all periods should be taken as a whole to be solved. Unfortunately, this is too ideal and it is impossible to collect all the necessary information. In fact, only a small portion of the orders and shipment schedules is known at the beginning and they are revealed gradually during the future planning process. The periodical solution strategy

is to decompose the whole problem into many subproblems in fixed intervals based on the assured order and shipment information. The solution quality is compromised at the expense of having complete information. From a theoretic perspective, shortening the duration of each period would improve the overall solution quality, because a change in the dynamic environment can be reflected quickly in planning. The shortening goes to an extreme if we apply replanning upon any dynamic change in the system, instead of fixing each planning period. However, this doesn't necessarily improve the solution when compared to traditional optimization approaches. In reality, solution quality may worsen, because the decision variables in each periodical problem do not have enough alternatives to choose from. This requires new types of approaches that can keep the solution optimal at any time, but subject to the dynamic changes in the environment.

## 1.3 Market and multi-agent based approach and its extension

According to the discussion above, a distributed and dynamic supply chain transportation planning problem requires a dynamic optimization approach in an information-rich environment. This approach needs to have the following features: (1) the arrivals of new orders are continually considered in the optimization process; (2) the cancelation of orders evokes incremental optimization; (3) alternative plans will be provided for failed orders; (4) the execution of transportation plans will be monitored and controlled. In this thesis, we propose the market and multi-agent based approach and its several extensions to model and solve distributed and dynamic supply chain transportation problems.

The Multi-Agent System (MAS) is introduced to model the information-rich environment with the provision of the computation and communication environment for

incremental optimization. A software agent is a software component emulating a human being's behaviors: responsive, pro-active and sociable [Jennings et al., 1998]. A responsive agent can detect and update change in its environment; a social agent can interact with other agents through synchronous and asynchronous communication; a pro-active agent is intelligent and can predict and plan for future actions. We envision that MAS can provide a systematic model for a dynamic and distributed environment. Due to cheap communication costs via the Internet, MAS offers a large scale and powerful computation environment. For this reason, MAS provides opportunities to optimize supply chain transportation beyond traditional approaches.

A market based distributed heuristic is incited by an observation that the equilibrium of the market enables the achievement of dynamic optimization. An artificial and virtual market is constructed to emulate the transportation planning process. In the emulation, the transportation tasks are traded between agents who represent transportation facilities, usually trucks. The convergence of task trading leads to an optimal solution that is subject to dynamic changes. In this thesis, this approach is termed marked based distributed heuristic (MBDH).

Usually, the strict market heuristic easily converges to local optimal solutions because of the intrinsic NP-hard properties of the transportation planning problem. We developed two types of upper level control over the distributed heuristic to avoid local optimality. In the first type, distributed search is kept and reinforcement learning is introduced to learn from the market interactions so as to avoid local optimality. In

the second type, the variable neighborhood search (VNS) heuristic is developed to advise the distributed market search in the architecture of meta-heuristic enabled MAS optimization.

The market based mechanism can also be used to model the task delegation between different transportation companies. In this model, the self-interest of each transportation company is addressed. From the perspective of optimization, this is a multiple objective optimization problem. We propose a combination of fictitious play and reinforcement learning to solve the problem.

## 1.4 Thesis organization

The rest of the thesis is organized as follows. Chapter 2 presents the definition of supply chain planning problems. The problem is defined from two different views: one is from the perspective of a single transportation company, and the other is from the perspective of multiple transportation companies. Chapter 3 summarizes the solution methodology. Chapter 4 summarizes the conclusion and future work. Solution strategy details are elaborated in Chapters 5, 6, 7 and 8 respectively. Each of these chapters result in the papers that are being sent out for publications. Chapter 5 presents the detailed development of the multi-agent and market based distributed heuristic. Chapter 6 presents an improved approach with the combination of the market heuristic and reinforcement learning. Chapter 7 present another improvement based on the variable neighborhood search meta-heuristic. Chapter 8 presents the combination of fictitious play and reinforcement learning to solve multi-player and multi-stage games in market interactions.

## Bibliography

D. Brugali, G. Menga, and S. Galarraga. Intercompany supply chains integration via mobile agents. Jacucci, G. (ed.), Globalization of Manufacturing in the Digital Communications Era of the 21st Century: Innovation, Agility and the Virtual Enterprise, Kluwer Academic Publisher, 1998.

N. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Journal of Autonomous Agents and Multi-Agent Systems*, 1:275–306, 1998.

G. Olsen. An overview of b2b integration. *EAI Journal*, May 2000.

H. M. Salkin, K. Mathur, and R. Haas. *Foundations of Integer Programming*. New York : North-Holland, 1989. ISBN 0-4440-1231-1.

G. Satapathy. *Distributed and Collaborative Logistics Planning and Replanning Under Uncertainty: A Multiagent based approach*. PhD thesis, Industrial and Manufacturing Engineering Department, PA, December 1999.

B.O. Szuprowicz. *Supply Chain Management for E-business Infrastructures*. Computer Technology Research Corp., first edition, 2000.

K.C. Tan, L.H. Lee, Q.L. Zhu, and K. Ou. Heuristic methods for vehicle routing problem with time windows. *Artificial Intelligence in Engineering*, 15:281–295, 2001.

## Chapter 2

# Problem definition

A general supply chain procurement problem [Satapathy, 1999] can be stated as the following optimization problem. Given that (1) transportation networks, the availability of the transportation facilities (trucks) and cost functions are provided; (2) a set of suppliers declares availability of unfinished products or a set of vendors declares the readiness of goods to be delivered, within required time windows; and (3) a set of manufacturers declares the capability of producing finished products from the unfinished products or a set of consumers declares the need for the ordered goods with windows, the objective is to determine the traveling sequence for each transportation facility (truck) to minimize the total transportation cost while guaranteeing the time window constraints of pickups and deliveries. This problem can be categorized as a planning and scheduling problem, whose planning process may cover a lot of organizations in supply chains and may involve multiple transportation companies with their respective objectives and interests. In this chapter, we define two problems from different perspectives. The first one relates to the supply chain procurement for a single transportation company; the second one relates to the procurement among several transportation companies in a collaborative and competitive environment.

## 2.1 Supply chain procurement for a single transportation company

For a single transportation company, the supply chain procurement problem is very similar to a Pickup and Delivery Problem with Time Windows (PDPTW) [Snezana, 1998] in a stochastic environment, where the traveling and service times are modeled as stochastic variables.

### 2.1.1 Transportation orders and tasks

Suppose that the transportation company receives $n^o$ orders. Order $k \in [1, n^o]$ has $m_k^t$ transportation tasks. In total, the transportation company needs to deal with $n = \sum_{k=1}^{n^o} m_k^t$ tasks, each of which is a pickup and delivery activity with time windows.

For those $n$ tasks, let node $i$ and $i+n$ denote the pickup and delivery locations of task $i$. Let $T_i^o$ denote the time when the order containing task $i$ is placed[1]. Let $P^+ = \{i | i \in [1, n]\}$ denote all the pickup nodes, $P^- = \{i | i \in [n+1, 2n]\}$ denote all the delivery nodes, and $P = P^+ \bigcup P^-$. Let $[e_i, l_i]$ $(i \in P)$ denote the earliest and latest arrival time of node $i$. Let $q_i$ $(i \in P)$ denote the number of units to be loaded at location $i$ ($q_i$ is negative if $i$ is a delivery location). In short, the tuple $< q_i, [e_i, l_i] >$ denotes a pickup or delivery transportation task node at $i$. The combination of $< q_i, [e_i, l_i] >$ and $< -q_i, [e_{n+i}, l_{n+i}] >$ describes a transportation task.

### 2.1.2 Transportation facilities

We mainly consider shipment using trucks. Let $V$ denote the set of available vehicles and $|V|$ the fleet size. Let $Q_v$ $(v \in V)$ denote the capacity of vehicle $v$. Let node

[1] Usually, this time is automatically recorded by the information system, e.g. database system.

0 denote the starting location and node $2n+1$ the ending location for all the vehicles considered. In our supply chain procurement scenario, the starting and ending locations for all the vehicles in the same transportation company share the same location.

### 2.1.3 Stochastic transportation network

Let $\mathbf{N} = \{0\} \bigcup P \bigcup \{2n+1\}$ denote all the locations in the transportation network. Let $d_{i,j}$ denote the distance between locations $i$ and $j$ ( $i, j \in \mathbf{N}$ ), where $d_{i,j}$ is usually approximated with the Euclidean distance. Let $\tau_{i,j}$ denote the traveling time between $i$ and $j$. $\tau_{i,j}$ is random with the probability density function (pdf) $f^{\tau}_{i,j}(\cdot)$. Let $s_i$ ($i \in P$) denote the service (loading or unloading) time at location $i$. $s_i$ is random with the probability density function (pdf) $f^{s}_{i}(\cdot)$. The historical data of $\tau_{i,j}$ and $s_i$ are used to estimate the parameters of the probability density functions $f^{\tau}_{i,j}(\cdot)$ and $f^{s}_{i}(\cdot)$.

### 2.1.4 Cost functions

Cost is linear to traveling distance and driver working time. Let $c^d$ denote the cost per unit distance, and $c^w$ the cost per unit driver working time.

### 2.1.5 State and decision variables

The variables representing system states and decisions to be made are as follows:

*Flow variables:* Let $x_{v,i,j}$ denote the flow variable from node $i$ to node $j$ ($i, j \in \mathbf{N}$) for vehicle $v$. If vehicle $v$ travels from location $i$ location $j$, $x_{v,i,j}$ is 1; otherwise, it is 0.

*Scheduled time variables:* Let $T_{v,i}$ denote the arrival time of vehicle $v$ at node $i$ ($i \in \mathbf{N}$).

*Load variables:* Let $y_{v,i}$ be the load on vehicle $v$ when it departs node $i$.

### 2.1.6 Mathematical formulation of the problem

**Minimize**

$$\mathbf{\Phi} = c^d \sum_{v \in V, i,j \in \mathbf{N}} d_{i,j} x_{v,i,j} + c^w \sum_{v \in V} (T_{v,2n+1} - T_{v,0}) \tag{2.1}$$

**Subject to:**

$$\sum_{v \in V} \sum_{j \in \mathbf{N}} x_{v,i,j} = 1, \qquad \forall i \in P^+ \tag{2.2}$$

$$\sum_{j \in \mathbf{N}} x_{v,i,j} - \sum_{j \in \mathbf{N}} x_{v,j,i} = 0 \qquad \forall v \in V, i \in P \tag{2.3}$$

$$\sum_{j \in P^+} x_{v,0,j} = 1, \qquad \forall v \in V \tag{2.4}$$

$$\sum_{i \in P^-} x_{v,i,2n+1} = 1, \qquad \forall v \in V \tag{2.5}$$

$$\sum_{j \in \mathbf{N}} x_{v,i,j} - \sum_{j \in \mathbf{N}} x_{v,j,n+i} = 0, \qquad \forall v \in V, i \in P^+ \tag{2.6}$$

$$T_{v,i} \leq T_{v,n+i}, \qquad \forall v \in V, i \in P^+ \tag{2.7}$$

$$e_i \leq T_{v,i} \leq l_i, \qquad \forall v \in V, i \in \mathbf{N} \tag{2.8}$$

$$x_{v,i,j} = 1 \Rightarrow$$

$$\int_0^{T_{v,j} - T_{v,i}} f_i^s(x) f_{i,j}^\tau (T_{v,j} - T_{v,i} - x) dx \leq \alpha, \qquad \forall v \in V, i, j \in \mathbf{N} \tag{2.9}$$

$$x_{v,i,j} = 1 \Rightarrow y_{v,i} + q_i = y_{v,j}, \qquad \forall v \in V, i, j \in \mathbf{N} \tag{2.10}$$

$$y_{v,0} = 0, \qquad \forall v \in V \tag{2.11}$$

$$y_{v,i} \leq Q_v, \qquad \forall v \in V, i \in \mathbf{N} \tag{2.12}$$

$$x_{v,j,i} = 1 \Rightarrow T_i^o \leq T_{v,0}, \quad \forall v \in V, i \in P^+, j \in \mathbf{N} \tag{2.13}$$

$$x_{v,i,j} \in \{0, 1\}, \qquad \forall v \in V, i, j \in \mathbf{N} \tag{2.14}$$

The mathematical programming formulation of PDPTW in a stochastic environment is a non-linear integer program (IP), as described by Equation (2.1) to (2.14). Its

objective function is defined in Equation (2.1), which is the summation of all possible costs caused by total traveling distance and drivers' working time.

Equations (2.2) to (2.5) describe the multi-commodity flow constraints. Equation (2.2) states that one vehicle leaves each pickup location for a unique next location. Equation (2.3) states that for each location $i$ in the pickup and delivery network, each vehicle entering location $i$ leaves location $i$. Equation (2.4) states each vehicle enters only one pickup location from its starting location. Equation (2.5) states that, for each vehicle, one and only one delivery location is the ending location. All these constraints make sure that each vehicle starts from the depot, and continually pick up parcels and delivery parcels in the pickup and delivery network until it ends with the depot without conflicting with other vehicles.

Equations (2.6) and (2.7) designate pickup and delivery relationship constraints. Equation (2.6) states that a pickup node has to be paired with its corresponding delivery node, because these two nodes belong to the same transportation task. Equation (2.7) states that a task's pickup node must be visited before its corresponding delivery node.

Equations (2.8) and (2.9) describe arrival time constraints. Equation (2.8) states the time window constraints, where each vehicle must visit its pickup and delivery locations between the earliest and latest allowable times. Equation (2.9) describes the reliability constraints in the stochastic environment. Let $\alpha$ denote the threshold of reliability for all the traveling arcs. Equation (2.9) states that for a traveling arc $(i, j)$, the probability a vehicle travels from location $i$ at time $T_{v,i}$ to $j$ at time $T_{v,j}$ should not be less than $\alpha$ with the uncertainty of the service time at $i$ and traveling time between $i$

and $j$. Specially, the probability $\Pr(T_{v,i} + s_i + \tau_{i,j} \leq T_{v,j})$ is calculated by convolution as

$$\Pr(T_{v,i} + s_i + \tau_{i,j} \leq T_{v,j}) = \int_0^{T_{v,j}-T_{v,i}} f_i^s(x) f_{i,j}^{\tau}(T_{v,j} - T_{v,i} - x)dx.$$

Equations from (2.10) to (2.12) describe the constraints related to vehicle capacities. Equation (2.10) defines how vehicle load is updated. Basically, if a vehicle visits an arc between locations $i$ and $j$, the load at location $j$ is to be increased with the volume of the task node $j$. This volume has been adjusted with the types of transportation task nodes. It is positive for a pickup node and negative for a delivery node. Equation (2.11) initializes the load to zero at the depot for each vehicle. Equation (2.12) states that the load in each vehicle at any node can not exceed the capacity of the vehicle.

Equation (2.13) defines the constraints in the dynamic environment. If location $i$ is visited by vehicle $v$, it is known that task $i$ is allocated to vehicle $v$. The order time of task $i$ should be before the time that vehicle $v$ leaves the depot. After a vehicle leaves its depot, its plan will be fixed.

Among all the constraints, Equations (2.9) and (2.10) look special, but the "If-Then" relationships can be transferred easily into linear relationships [Winston and Venkataramanan, 2002], because the decision variable $x_{v,i,j}$ is a binary integer. The only non-linear part may exist in the integral part of Equation 2.9, whose calculation is extremely expensive.

## 2.2 Task delegation among multiple transportation companies

For infeasible or extremely expensive transportation tasks, task delegation can be enabled among multiple transportation companies with overlapping transportation networks. Different from the single-company problem, which has a unique objective function, the problem of task delegation addresses the self interest and objective of each transportation company. In fact, task delegation is modeled as a game with multiple players in multiple stages. Details of the problem definition appear in Section 8.2.

## Bibliography

G. Satapathy. *Distributed and Collaborative Logistics Planning and Replanning Under Uncertainty: A Multiagent based approach.* PhD thesis, Industrial and Manufacturing Engineering Department, PA, December 1999.

Mitrovic-Minic Snezana. Pickup and delivery problem with time windows: A survey. Technical report, Simon Fraser University, 1998.

W.L. Winston and M. Venkataramanan. *Introduction to mathematical programming.* Belmont, Calif : Duxbury ; London : Thomson Learning, 2002. ISBN 0-5343-5964-7.

# Chapter 3

# Solution methodology

The supply chain procurement problem discussed in Chapter 2 can be formulated as a non-linear integer programming problem. However, it cannot be solved with any optimization package, due to its complexity. To illustrate, for a problem with 25 vehicles and 53 transportation tasks, $53^{25}$ combinations must be considered in the worst case and each combination involves solving multiple TSPs (Travel Salesman Problem). TSP optimization packages are based on the concept of Branch-and-Bound or Cutting Plane Techniques [Salkin et al., 1989], whose computational expense grows exponentially with the problem scale. To deal with the complexity, we are mainly interested in the development of heuristic algorithms from the perspective of optimal searching. On the other hand, the methodology development needs to cover the requirements of the real system. The requirements follow. First, the entities and functionalities involved in the procurement planning process are physically distributed. Second, a dynamic optimization or any-time algorithm is needed because the processes of order placement, planning and execution are interwoven. Finally, contingency planning is needed for handling execution failures. All these require the capability of real-time responsibility in a distributed system. The distributed system can be modeled with a multi-agent system and the real-time capabilities are achieved through heuristic algorithms that can be adapted to MAS.

In the following sections, we describe several methods that solve the supply chain procurement problem under different situations. Section 3.1 summarizes the method of the double auction market mechanism (DAMM), whose interactions constitute a distributed market based heuristic (DMBH) for supply chain procurement planning. Its details are in Chapter 5. Section 3.2 summarizes the method of coordinated reinforcement learning, which improves DMBH from a machine learning perspective. Its details are in Chapter 6. Section 3.3 summarizes the Variable Neighborhood Search (VNS) method, which improves DMBH from a meta-heuristic perspective and upper controlling in MAS optimization. The details are given in Chapter 7. Section 3.4 summarizes the combinatory method of reinforcement learning and fictitious play to solve a game with multiple players in multiple stages with cooperation and competition among multiple transportation companies. The details are given in Chapter 8.

## 3.1 Double auction market mechanism

The design of MAS not only reflects the the properties of distributed physical entities, but also is prepared for solving the supply chain procurement planning problem. An agent is a software system that emulates the behaviors of a human being [Jennings et al., 1998]. Usually, addressed are three behaviors: social behavior requiring communication among agents, responsive behavior requiring that the environment is monitored and changed by agents, and pro-active behavior requiring that agents can predict and pre-plan for the future. According to the behavior of an agent, the criteria to define a physical entity as an agent entail that the agent has to represent an automatic decision maker. The criteria are used to model the process of supply chain procurement planning,

and we mainly design two types of agents: company agents and vehicle agents. A company agent represents a transportation company who receives transportation orders from its customers and wants to allocate its transportation tasks, which are decomposed from orders, to its vehicles. A vehicle agent represents a transportation facility, e.g. a truck, which knows how to route among a set of locations and wants to reallocate its tasks for better performance in a dynamic environment. The task allocation and reallocation are enabled through interactions between company agents and vehicle agents.

Market theory and its interactive mechanisms, especially double auction mechanism, are borrowed to enable distributed interactions among a group of software agents. To emulate a market, the following relationships are associated. Company or vehicle agents are designated as traders or trading agents; transportation tasks are designated as products or commodities in the market; task exchanges between traders are designated as transactions; cost-saving from task exchanges are designated as profit or cash flow. In other words, we model transportation companies and vehicles as human beings whose their behaviors for solving the transportation problem are similar to the behaviors of those traders in a market. Besides traders, the double auction mechanism requires the involvement of another virtual entity, namely a monitor, which corresponds to a market specialist. In MAS, a monitor is designed as a software agent. For agents to interact in the virtual market, three important subproblems need to be solved: first, how to evaluate the marginal cost for a vehicle agent; second, how to enable the market interactions continuously; finally, how to set up convergence criteria for the continuously interactive process.

The evaluation of marginal cost for a vehicle agent is crucial. Suppose that a vehicle has a set of tasks which are sequenced optimally in the routing plan of the vehicle. Marginal cost is defined as the cost increase or decrease associated with deleting or inserting a task from or into the set of tasks. A cost increase is an insertion cost and a cost decrease is a deletion cost. During market interactions, a marginal cost represents the reverse price of its corresponding task. Ideally, the evaluation of marginal cost involves solving two TSPs. One of them has the original set of tasks; the other one has a union task set with a new task not included in the original set or a different task set with an existing task included in the original set. The difference between two subtotal costs is the marginal cost. However, due to the high calling frequency of cost evaluation, this method is still too computationally expensive. Based on the idea of a heuristic algorithm that addresses the time windows constraints [Solomon, 1987], we develop an iterative and direct way to evaluate insertion cost. The complexity of this algorithm is $O(n^2)$, where $n$ is the number of locations in the routing plan ( a pickup and delivery task has two locations). At first, all the positions before which the pickup location can be inserted are identified without violating the constraints of time windows. Then, based on the set of insertion positions for the pickup location, the insertion positions corresponding to each insertion position for the delivery location are identified. The cost differences are directly evaluated for all combinations and the minimum one is chosen as the insertion cost. The evaluation of deletion cost is much simpler in this heuristic. Only the related pickup and delivery locations are removed from the routing sequence and the deletion cost is calculated according to their difference.

Market interactions are continued round by round until the convergence criteria are reached. At the beginning of a round, the virtual market needs to be cleared. In the clearance, the monitor agent discards all the proposals not transacted and the trading agents clear their proposals. Then each trading agent makes a decision whether to sell, buy or keep silent for this round. The company agent always sells. A vehicle agent generates an action based on the predefined uniform discrete probability distribution. Then each selling agent submits its ask (a proposal to sell one of its tasks) based on a stochastic model, which assigns a probability to each of its tasks. Each probability reflects a willingness to sell the corresponding task. Simultaneously, these asks are collected by the monitor agent and are broadcast to the buying agents immediately after the selling agents respond. Then each buying agent submits a bid based on a stochastic model, which assigns a probability to each selling task. This probability reflects a willingness to buy the corresponding task. After all the bids are collected by the monitor agent, the monitor agent combines the previous selling proposals to suggest task exchanges. The suggestion decision is based on a simple market rule: the highest ask and the lowest bid are matched for the same task. After the task exchange signal is sent to the corresponding agents, they update their transportation plan accordingly. This marks the end of one round of interaction.

The convergence criteria of the virtual market for the supply chain procurement planning problem is related to the equilibrium of a real market. In a perfect market, equilibrium is reached if no one is interested in selling or buying because each trader is self-interested. However, in the virtual market, we haven't modeled the self-interested strictly due to the low quality of a self-interested market. In fact, each trading agent

expresses their interests in some random process. So, the convergence criteria needs to be adapted accordingly. Suppose that randomly chosen selling agents submit their asks without any restrictions. If no buying agents respond, it is possibly a signal that the virtual market is converging. This signal is not absolute due to randomness. We will wait for a strong signal, i.e. the signal of no-response repeats many times, and issue the convergence of the virtual market.

## 3.2 Reinforcement Learning for refinement

Although randomness is introduced to the market mechanism to avoid local optimality during the distributed heuristic search, there is always a gap between the heuristic solution and the optimal solution. In general, computation in MAS is very fast due to its distributed nature. However, the physical execution process is very slow as compared to the computation. So, we develop a refinement method in the framework of reinforcement learning, namely Coordinated Reinforcement Learning (CRL), because reinforcement learning plays the coordination role. We envision that the search history of the market heuristic can help find a higher quality solution.

CRL is a marriage of centralized reinforcement learning and the distributed market heuristic. From the perspective of machine learning, CRL accumulates knowledge and the distributed market heuristic is used for training as a simulation platform. To build the bridge between learning and simulation, the congregate representation of task allocations derived from the transportation plans is defined as a *state*; the collective proposals of the task agents are defined as an *action*; the cost savings during one round of market interaction are defined as a *reward*; the update of transportation plans resulting

from updating task allocation is defined as a *state transition*. In CRL, a hash table is used to represent the tree-like structure of the accumulated knowledge. In the tree, a state is a node, an action is an arc, and a Q value is associated with each arc. A usual learning round can be described as follows. First, a starting state is chosen based on the depth of the state in the tree. Then, this state is transferred to the transportation plans which are distributed among multiple vehicle agents. Thus, a virtual market with an initial allocation is set up and the market interactions automatically start to work. The interactions are repeated round by round until the virtual market heuristic converges. A state transition and reward are extracted from the result of each market interaction and then sent to the centralized learning component for monitoring. At the same time, the reinforcement knowledge tree is updated accordingly with the state transition and reward. The learning process goes round by round until the convergence criteria of the learning is reached.

Although the convergence criteria of normal reinforcement learning requires the convergence of all the Q values, only a small group of Q values are required in the refinement method for supply chain procurement planning. The solution quality of the whole problem relies on the Q values whose arcs start from the original state, because the maximum of them corresponds to the optimal solution. If this value converges, the learning process can be recognized as converged.

## 3.3 Meta-heuristic Enabled MAS Optimization

This part of the work is an extension of Section 3.1 and 3.2. Although CRL helps to improve the solution and the result can be useful for execution at any time

even without convergence, it faces the difficulty of the curse of dimensionality. This is because there are too many interaction possibilities at a given state to store the Q values approximately. Thus, the method requires too much memory and computing time. However, meta-heuristics, which have been applied in global optimization for combinatorial problems, do not require unlimited memory. Based on this observation, we attempt to introduce meta-heuristic methods to improve DAMM by avoiding the curse of dimensionality.

From the perspective of MAS optimization, introducing meta-heuristics provides an upper controller for myopic agents. MAS optimization architecture is usually featured with a distributed heuristic algorithm for optimization in a dynamic environment. The introduction of meta-heuristics is not to change the existing distributed algorithm, but to monitor and control it at a high level. To be adaptive to the MAS information infrastructure, the high-level controller is modeled as a software agent to interact with other existing agents in a distributed manner.

Considering the scenario of DAMM for the supply chain procurement problem, we develop a meta-heuristic control agent using Variable Neighborhood Search (VNS). VNS is a recent meta-heuristic method which systematically changes neighborhoods to avoid local optimality. The distributed heuristic based on DAMM is termed the Distributed Market Based Heuristic (DMBH). In controlling DMBH, VNS switches its starting state in order to direct distributed local heuristic by utilizing the knowledge of an artificial neighborhood structure. In this structure, a memory pool with limited size is used to store the most representative states visited by DMBH and these representatives are separated into multiple clusters according to their objective values. Each cluster is a

neighborhood. In the controlling mechanism, VNS sets the starting state of DMBH in the neighborhood of the highest objective values, and any new state explored is used to update the memory pool by keeping the properties of the most representative states. If the search improves the objective value, the search starts over from the first neighborhood; otherwise, the next neighborhood with a lower objective value than the previous one is chosen as the searching domain. This process continues until no improvement can be found among all the neighborhoods.

## 3.4 Multi-player and multi-stage game

For overly expensive transportation tasks, a game with multiple players in multiple stages is used to model cooperative and competitive task delegation among multiple transportation companies with overlapping transportation networks. Starting from an initial state in which each company agent holds a list of expensive tasks, the company agents determine their market actions: to sell a task, to buy a task, or to keep silent. The payoffs of their actions are determined by the market rule. Then the company agents update their task lists according to their payoffs and move to another state. This process will continue until each company agent decides to keep silent. Although this process is very close to the distributed heuristic search for a single transportation company, the company agents actually play matrix games at each state with their instant payoffs. The accumulated payoff over all the states for a company agent is its self-interest.

To solve the game with multiple players in multiple stages, we use the equilibrium concept of game theory. The solution concept of game theory can be described from different perspectives. The first one is the Nash Equilibrium, which assumes infinite

rationality, i.e. common knowledge. In other words, I know your payoff matrix, and you know my payoff matrix; I know that you know my payoff matrix and you know that I know your payoff matrix. This can go on infinitely. For multiple players, to model this infinite rationality is computationally infeasible. The Nash equilibrium is very important from a theoretical perspective, but the solution for a general game is very hard to find due to the assumption of infinite rationality, which results in infinite memory. As a modification, bounded rationality is proposed. It only models finite depth of rationality, so it only uses finite memory. One important methodology in this category is the fictitious play in which only the statistics of each action are remembered. But it only finds a local optimal solution. Another modification is the evolutionary stable strategy, which is derived from evolutionary computation. Based on some criteria, the converged solution of randomized game playing is the solution for the game. This is the equilibrium concept we used to seek the computational solution for this multi-stage and multi-player game.

We applied a reinforcement learning and fictitious play combination to seek an evolutionary stable strategy. In the combination, reinforcement learning provides a framework for the multi-stage game learning. At each node of the reinforcement learning tree, the agents share a common state, but they have their own Q values for each of their specific actions. These values in different levels construct the reinforcement learning knowledge of a specific agent. During the process of game learning, each agent chooses his strategy based on his own knowledge and update his own Q values. We have two update processes, one updates the Q values and the other updates the level of each state. Q values measure the maximum accumulated payoff for a state and action pair. The

levels of a state measure the maximum distance between the state and a leaf state. The levels are useful for state exploration, i.e. after a game reaches a leaf node, where to start the game next. We use Bolzmann exploration with a stochastic preference for the state with the highest level. The Q values are used for strategy or action exploration. Stochastic preference is given to the action with the highest Q value. Fictitious play approximately models the preferences of other players. Similar to reinforcement learning, each agent has his own fictitious play knowledge at each state. Its knowledge is very simple, remembering the number of action occurrences of other agents at this state. This information will always be updated during play, whether it is reinforcement play or fictitious play. When fictitious play is invoked, the agent at first guesses the actions of other agents based on its belief profile. If a deterministic strategy is used, the agent only needs to find the action with the maximum number of occurrences. If a stochastic strategy is used, the stochastic preference is given to action with the maximum number of occurrences. After the agent guesses all the actions of other agents, it can find out the best response of his own by simulating one round of market interaction.

Using reinforcement play and fictitious play alone will produce non-equilibrium solutions. We combine these two game learning algorithms to run in an interwoven manner. This combination is successful, because fictitious play provides an approximate model of the action preferences of other agents. In the two-player game, other agent's action preferences can be modeled explicitly, but in a game with multiple players, explicit modeling is too expensive, especially in a multi-stage game.

## Bibliography

N. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Journal of Autonomous Agents and Multi-Agent Systems*, 1:275–306, 1998.

H. M. Salkin, K. Mathur, and R. Haas. *Foundations of Integer Programming*. New York : North-Holland, 1989. ISBN 0-4440-1231-1.

M. Solomon. Algorithms for vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35:254–265, 1987.

# Chapter 4

# Conclusions and future work

This chapter summarizes the contributions and points out possible future work. More details about contributions and future work are in Chapters 5, 6, 7, and 8.

## 4.1 Contributions

The contributions of the thesis are in the area of market based heuristic search, high-level control over MAS optimization, and computational game learning.

### 4.1.1 MAS based heuristic search

The supply chain procurement planning problem in a stochastic environment is an extremely difficult problem due to its NP-hard and non-linear properties. The development of the double auction market mechanism (DAMM) toward this problem is a valuable contribution, combining the disciplines of macro-economics, computer science, and operations research. Compared to the related work in the literature, the uniqueness of this approach is the following:

- **Simultaneous double auction institutions**

  The double auction institutions in the same market interaction round allow multiple sellers and multiple buyers to release their transportation task values. A trading agent can be a seller during the current round of market interaction, a buyer

during the next round, and a mute during another round. All these diversities create more opportunities for trading agents to exchange their transportation tasks than an auction protocol so that better cost savings can be achieved with market interactions.

- **Heterogeneous physical entities incorporated in the same market**

  From a management point of view, the relationship between a transportation company and its vehicles is a master and slave relationship. Vehicles will accept whatever transportation tasks their company assigns. In distributed task allocation literature, it is common for a transportation company to auction tasks to its vehicles without repeating announcement according to the master and slave relationship. In DAMM, however, a transportation company and its vehicles are all modeled as trading agents at the same level of the virtual market so that their master and slave relationship disappears. Instead, the original master/slave relationship is modeled implicitly through the restricted behavior design of the company agent. This not only provides an innovative idea for transferring the master and slave relationship to the peer to peer relationship, but also helps seek optimal solutions due to its loose market interaction.

- **Flexible proposal strategies**

  For a self-interested trading agent, it is rational to choose its optimal strategy during proposing stages. However, this strategy may cause the distributed heuristic search to converge easily and lose good opportunities to improve the solution further. On the other hand, the yield to myopically optimal solutions may help

the whole distributed heuristic search. The proposing strategy with stochastic properties offers this flexibility and our numerical results support the intuition.

- **Fast marginal cost evaluation model**

  For high efficiency computation, the marginal cost evaluation model is based on time window constraints. This greatly reduces the complexity of solving two TSPs. Computational experiments show that this model is valuable for solving large scale problems.

- **Convergence criteria related to market equilibrium**

  The convergence criteria of a normal heuristic is usually based on the convergence of objective values or the consumption of a large number of steps. In contrast, the convergence criteria of DAMM is based on the internal market equilibrium, which can assure the convergence of the heuristic better than outside observations of objective value changes.

- **Capability for dynamic optimization and contingency planning**

  Dynamic changes, including new incoming orders or contingency conditions, can invoke the distributed market based heuristic. Moreover, the heuristic not only quickly finds a feasible solution for dynamic changes, which is very common in many existing approaches, but also attempts to search the best solution with dynamic changes through market interactions, which is very rare in the existing approaches. Therefore, DAMM enables dynamic optimization and contingency planning through keeping optimality subject to any dynamic change.

### 4.1.2 High-level control over MAS optimization

In the MAS optimization literature, researchers usually hold two extremes. At one extreme, they develop purely distributed protocols and algorithms to eliminate any centralized components when solving optimization problems in MAS. At the other extreme, they use MAS as a means of collecting information and generating an initial solution and then use a global optimization approach for problem solving. Although the work of DAMM is at the first extreme, this thesis stands between the two. Actually, the development of algorithms in this thesis always assumes the existence of the distributed information infrastructure and, at the same time, attempts to involve limited global control to enable the distributed heuristic search to perform better. The introductions of coordinated reinforcement learning and variable neighborhood search are two attempts from two different perspectives: machine learning and meta-heuristics. Conceptually, these two perspectives may represent two main categories of approaches that may be adapted to MAS optimization for high-level control. From an application domain perspective, it is unique to develop RL and VNS enabled MAS optimization for supply chain procurement and transportation planning.

### 4.1.3 Computational game learning

Task delegation among multiple self-interested transportation companies is an optimization problem with multiple objectives. However, due to the existence of market interactions, games are used to describe and model the multiple objective optimization problem. It is the first attempt to model task delegation as a game with multiple players in multiple stages. This model not only enables a distributed heuristic to solve the

task delegation problem, but it also helps develop a computational method for solving a complex game based on the evolutionary stable strategy.

The game has two complexities. First, the game has more than two players, and in general analytic game theory contains only two players. Second, the game has multiple stages. However, an extensive game has multiple stages with only two players; at the same time, most computational solutions in game theory are developed for one stage. In a word, the existing approaches in game theory can not solve the game of task delegation with multiple players in multiple stages. Reinforcement learning and fictitious play are both applied in game learning in the literature. In the framework of reinforcement learning, it is very difficult to model others' strategies in a game with multiple players due to the exponential growth of the state and action space. It is well known that fictitious play is very sensitive to initial conditions and may converge to a non-equilibrium solution. In the combination of reinforcement learning and fictitious play, reinforcement learning models the self-interests of each player at different stages and fictitious play approximately models others' strategies. The combination is an effective approach for solving the game with multiple players in multiple stages.

## 4.2 Future work

We envision future work in the directions of application domains, distributed heuristics, limited centralized MAS controllers, and computational game learning.

### 4.2.1 Application domain

In the thesis, a distributed market based heuristic is applied to task allocation for supply chain procurement planning. Likewise, it can be extensively applied to other task allocation and resource allocation problems, especially in a distributed and real-time environment. This requires the design of trading agents, monitor agents, profit mechanisms, price or cost evaluation models, and convergence criteria, which are similar to what we did in Chapter 5. This extension is because of the intrinsic properties of task allocation and resource allocation, in which the relationships among multiple distributed decision makers are one-to-one, one-to-many, and many-to-many (seldom), and easy to be identified in market interactions.

From another problem extension perspective, it is significantly important to incorporate inventory control into the supply chain procurement problem. In the current version of the problem, infinite inventory is assumed at every location, which is too ideal for real applications. An innovative idea is to model the inventories at manufacturing centers, distribution centers, and warehouses with their constraints and policies modeled as trading agents, who join the virtual market of supply chain procurement planning.

### 4.2.2 Distributed heuristic mechanism

Although we have tried several different ways to improve the efficiency of market mechanisms in the framework of simultaneous and stochastic market interactions, other market institutions may improve the solution further. These market institutions are *bartering, bundles, and active buying.* Bartering enables direct task exchange between two trading agents, through they might be hard to be identified. Bundles enable several

tasks to be sold and bought together. Active buying enables buying proposals before selling proposals, because buying agents seem to be more informative than selling agents.

Another way to improve the distributed heuristic mechanism is to introduce intelligence to each individual trading agent. One possible way is to set up an association between environment patterns and trader proposing policies through artificial neural network based approaches. This machine learning model may help trading agents to make efficient decisions. If supply chain procurement planning happens periodically for the same set of locations, knowledge gained in machine learning may help problem solving from a long term perspective.

### 4.2.3 Limited centralized MAS control

The thesis introduces the MAS optimization architecture enabled by upper level control and identifies two promising directions: machine learning and meta-heuristics. From a machine learning perspective, an approximation model needs to be incorporated for solving large-scale problems. This requires the simplification of market interactions. For example, that only one transaction is allowed during an interaction round is a simplification, which can help set up an approximation model of between state-action pairs and Q values for reinforcement learning. From a meta-heuristic perspective, it may be necessary to study the performance of meta-heuristics such as simulated annealing, genetic algorithms, tabu search, etc. It would be valuable to study how these meta-heuristics can be adapted to MAS, while minimizing inter-agent communications.

### 4.2.4 Computational game theory

In the thesis, the combination of reinforcement learning and fictitious play is used to solve a complex game, which is very similar to the situation of a multiple criteria optimization problem. For future work, it may be useful to apply this approach to solve combinatorial optimization problems with multiple criteria. The key is to transform the optimization problem into a game with multiple players in multiple stages. Each objective corresponds to a player with its own self-interest and the decision variables correspond to the action profile of each player. This may open another way to solve combinatorial multiple criteria problems.

# Chapter 5

# Double auction market mechanism for supply chain procurement problem

**Abstract:** An innovative distributed negotiation mechanism – Double Auction Market Mechanism (DAMM) – is developed to model and solve an e-procurement problem, whose information is distributed in a stochastic environment with random service and traveling times. DAMM enables real-time exchange of transportation tasks among a transportation company and its trucks, which are modeled as software agents. Iterated task exchange among agents drives the task allocation to Pareto optimality. DAMM is an any-time negotiation protocol, so that contingency plans can be generated automatically.

**Keywords:** double auction, market mechanism, e-procurement, supply chains, multi-agent system, distributed negotiation, contingency planning

## 5.1 Introduction

Through computer networks, especially the Internet, not only have the internal processes of an organization been integrated into a supply chain [Szuprowicz, 2000], but also the external business processes between different organizations have begun to form a global supply chain, especially as new business models, B2B (Business to Business), B2C (Business to Customer) and Net-Market [Olsen, 2000] are becoming an integral part of the global supply chains. The concept of supply chains have entered the daily life due

to the developments in mobile agents and wireless communication [Brugali et al., 1998]. With the proliferation of high volume of data, managing or procuring the relationships among different organizations and/or individuals becomes challenging and is significantly important.

Though the developments in information technology (IT) has enabled the collection and storage of large amounts of data, to find an optimal way to manage or procure these relationships seems to be difficult due to the intrinsic complexity of the problem. In order to seek a solution with adaptive flexibility and scalability, researchers [Cook and Hamlin, 2000; Kalakota, 2000; Hull et al., 2000] began to develop distributed information systems. By modelling each B2B entity (organization, department, group, individual, etc.) as a software agent with the ability to compute and communicate, not only can the complexity of the problem be reduced, but also alternative plans under the contingent condition [Sauter and Parunak, 1999] are provided due to the distributed nature of processing.

Automatic negotiation protocols enable distributed agents to exchange information in order to reach mutually beneficial agreements. Each agent needs to be developed to emulate a human being's rationality and reasonability, which are the intrinsic mechanisms for negotiation among human beings. In addition, each agent needs to have a communication ability such that necessary information for decision making can be collected. An automatic negotiation protocol glues the rationality, reasonability and communication ability of the agents, so that a set of intelligent agents can be integrated into an organic information system.

Contract Net Protocol (CNP) [Davis and Smith, 1983] and Market Programming (MP) [Wellman, 1993] are two well-known automatic distributed negotiation protocols. In MP, computational software is programmed to simulate the trading process of goods in a real market organized by human beings. The methodology developed in this paper is a special case of MP, in which the double auction market mechanism (DAMM) is addressed.

DAMM is developed to solve supply chain procurement problems, where the procurement of suppliers and manufacturers is modelled as a stochastic pickup and delivery transportation problem. In the problem, each pickup and delivery transportation activity within some time window is defined as a task. The problem to solve is to allocate a set of tasks to a set of vehicles in real time under a stochastic environment, in which the travelling time and service time are considered as random variables. Information in this problem is usually distributed and therefore scalability becomes critical.

The rest of the paper is arranged as follows. Related work is presented in Section 5.2 and a detailed problem definition is given in Section 5.3. Section 5.4 provides details on the estimation of marginal cost in a stochastic environment. In Section 5.5, approaches based on DAMM are shown. Numerical results are given in Section 5.6, followed by discussion and analysis in Section 5.7. Finally, conclusions and future work are presented in Section 5.8.

## 5.2 Related work

In this section, related work concerning distributed negotiation protocols and their applications to the logistics procurement problem are discussed. Special attention is paid to market-based negotiation mechanism due to its close relationship with DAMM.

Negotiation is the process in which two or more agents move from contradictory decisions toward agreement through compromise or searching for new alternatives [Pruitt, 1981]. In the context of procurement problems, in general, the progress from an infeasible task allocation to a feasible one is a process of negotiation. In specific, negotiation starts from a feasible task allocation and more precisely refers to reallocation and replanning. Whatever state the negotiation starts at, it is a continuous process among a set of agents who agree to interact based on a common protocol.

Kraus [1997] suggests that in such a negotiation it is beneficial to combine AI techniques with techniques from other areas such as game theory, operations research, physics and philosophy. Recently, game theory has acted as a theoretical framework for many negotiation processes. For instance, auction theories [Klemperer, 1999; Wurman et al., 1998] and market programming [Walsh and Wellman, 1998] are two important negotiation mechanisms which are grounded in game theory. Especially, market programming exhibits the spirit of multi-player interactions in game theory.

In a multi-agent system that supports both local decision making and asynchronous operation, markets provide some answers to the motivation and direction for the actions of those individual agents to evolve toward a global goal [Wellman, 1993]. Although New York Stock Exchange and the major Chicago Exchanges held real trade for

homogeneous goods in the trade market 100 years ago [Friedman, 1993], the study of applicability toward a society of software agents did not start until 1993 in the WALRAS system [Wellman, 1993; Cheng and Wellman, 1998], which is an approach with a framework of decentralized problem solving based on market principles. WALRAS is a virtual simulated trading system organized by multiple computational agents. The producers supply prices for different goods. The consumers in turn, use these prices to create demand. The consumers' demands lead to another price quotation by the producers. The bidding cycle continues till an equilibrium, which is reached at the state where no producers want to sell and no consumers want to buy. It has been shown by Wellman [Wellman, 1992] that equilibrium exists under certain assumptions. This model was also applied to solve a transportation problem [Wellman, 1993], in which the vehicles were modelled as consumers and the transportation links were modelled as producers. Based on the selling and buying offers in a virtual trading market, finding the agreement of negotiation is an NP-complete problem [Bachem et al., 1993], but the distributed multi-agent model and the concept of equilibrium tracks the invisible hand in the market so that the computational efforts can be reduced.

In the recent past, researchers attempted to solve vehicle routing problems (VRP) using market programming. Fischer et al. [1995] modelled transportation vehicles and companies in a multi-agent system by using the CNP as the task allocation protocol; however, the final stage of the problem was solved using a centralized algorithm based on the idea of simulated trading. Kohout and Erol [1999] modelled vehicles and transportation orders as agents, and the solution was improved based on the repeated announcement of the allocated orders in the auction mechanism.

## 5.3 Problem definition

Given the same quality of products in a supply chain, the problem of providing the products requested by the customer becomes one of selecting the optimal means of transportation. More formally, given the availability at $s$ of a product within a time window and requirement at $d$ within a time window, how can the products be transported most efficiently? This is defined as a supply chain procurement problem. This is also termed as an e-procurement problem because the Internet plays a key role. The ontology of this problem is presented in the following.

A **transportation order** is defined as a request of picking up a set of products from a location and delivering them to another location with the restriction of time windows at both the locations. Let a tuple $o =< V, (p, d), [e_p, l_p], [e_d, l_d] >$ represent a transportation order, where $V$ denotes the set of parcels (each parcel can contain multiple products) to transport, $p$ the pickup location, $d$ the delivery location, $[e_p, l_p]$ the time window at the pickup location $p$, and $[e_d, l_d]$ the time window at the delivery location $d$. In the set $V = \{v_1, v_2, ..., v_c\}$, each element $v_i$ $(i \in [1, c])$ denotes the amount of volume of an indivisible **parcel** and $c$ denotes the total number of parcels in $V$. A parcel is a collection of products that are packaged to be shipped together due to either the customer requirements or supplier preferences. For the potential efficiency of transportation, an order is decomposed as tasks as granular as possible.

A **transportation task**, which is derived from a transportation order, defines the transportation activity with one parcel to be picked up and then delivered. Let a tuple $t =< v, (p, d), [e_p, l_p], [e_d, l_d] >$ denote a transportation task, where $v$ is the volume

amount of a parcel and other notations share the meanings with its corresponding order. Different transportation tasks belonging to the same transportation order share the basic properties.

The process of generating tasks from an order is called **task decomposition**. Based on the above definitions, this is a splitting process, in which each parcel with the other properties of its order organizes one individual task. To illustrate, a transportation order $o =< V, (p, d), [e_p, l_p], [e_d, l_d] >$ can be decomposed into $c$ transportation tasks as $t_1 =< v_1, (p, d), [e_p, l_p], [e_d, l_d] >$, $t_2 =< v_2, (p, d), [e_p, l_p], [e_d, l_d] >$, $\cdots$ and $t_c =< v_c, (p, d), [e_p, l_p], [e_d, l_d] >$. The decomposition of an order into a set of tasks provides more flexibility for transportation, so that the parcels in one order are not restricted to be shipped together.

Intuitively, the transportation company is modeled as an agent who collects the transportation tasks from customers and then decompose them as tasks. Vehicles are modeled as agents who simulate virtual execution of transportation plans. In most of the cases, a company agent serves as a master and vehicle agents as slaves [Satapathy, 1999]. No negotiations occurs between a master and its slaves, but slaves are forced to accept what its master commands. In this research, we model a master agent and its vehicle agents in the same level uniquely. They have the same market roles in DAMM and are called task agents.

A **task agent** is either a transportation company agent or a transportation vehicle agent. Let $\{A_i, i = 0, ..., n\}$ denote these $n + 1$ task agents, where $A_0$ designates the transportation company agent and each of $\{A_i, i = 1, ..., n\}$ a transportation vehicle agent. Each agent $A_i$ holds a set of tasks $T_i = \{t_i^1, ..., t_i^{m_i}\}$, where $m_i$ denotes the

number of tasks agent $A_i$ holds. A vehicle agent $A_i, i = 1, ..., n$ has to be able to execute the set $T_i$ of tasks by satisfying all the customer requirements such as the constraints of time windows, capacities, and reliability. However, the company agent $A_0$ has no ability to execute the set $T_0$ of tasks, which must be allocated to its vehicles. If the vehicles in $\{A_i, i = 1, ..., n\}$ have no abilities to execute these tasks, the tasks that are not allocated will be delegated to other transportation companies.

In the context of a supply chain procurement problem, some important questions are:

- How are the tasks allocated to the vehicle agents efficiently in a distributed information system where the task agents are connected through the Internet?
- How do the agents exchange the task commitment relationships virtually to improve the transportation efficiency?
- What is the criteria of Pareto Optimality if heuristics are applied?
- How are the realtime requirements handled?
- How are the contingency plans taken care of?
- How is a large-scale problem manipulated?

In this paper we attempt to address these questions. *Based on the algorithm of evaluating marginal costs in each task agent, a distributed negotiation protocol DAMM is developed to solve a large-scale e-procurement problems. This distributed negotiation protocol provides an efficient and flexible way to deal with the real-time and contingent requirements.* We start discussion with marginal cost evaluation.

## 5.4 Marginal cost evaluation

The efficiency of task allocation among a set of task agents is evaluated based on their transportation costs. Lesser the cost, higher the efficiency. On the other hand, the total cost associated with a set of tasks in DAMM is not so interesting as cost increases or decreases when a task is inserted into or deleted from the set of tasks. The former cost is referred to a static cost and the latter a dynamic one, which is also termed as **marginal cost** [Sandholm, 1996]. Two types of marginal cost, which designate cost increase and decrease respectively, along with additional definitions, cost functions, reliability are defined in this section.

### 5.4.1 Additional definitions

A **task node** $k$ denotes a mandatory location to visit to execute a task. In general, the attributes corresponding to a task node $k$ are a tuple $< (k^p, k^d), [e_k, l_k], \Delta q_k, s_k >$, where $k^p$ denotes its pickup location, $k^d$ its delivery location, $[e_k, l_k]$ its time window, $\Delta q_k$ its update on volume, and $s_k$ its service time. If $k$ is a pickup node, then $k^p = 0$, $\Delta q_k$ is positive and $s_k$ is the pickup service time. On the other hand, if $k$ is a delivery node, then $k^d = 0$, $\Delta q_k$ is negative and $s_k$ is the delivery service time. A task consists of two task nodes, one for pickup and the other for delivery. Thus, a task $t =< v, (k^p, k^d), [e_{k^p}, l_{k^p}], [e_{k^d}, l_{k^d}] >$ can be decomposed into two task nodes $k^p$ and $k^d$ with corresponding attributes $< (0, k^d), [e_{k^p}, l_{k^p}], v, s_{k^p} >$ and $< (k^p, 0), [e_{k^d}, l_{k^d}], -v, s_{k^d} >$.

A **schedule**, corresponding to a task node, is a piece of routing information that tells where, when, and how a vehicle visits a task node. Let $\pi =< k, a, w, q >$ denote a

schedule, where $k$ denotes the task node to visit, $a$ the arrival time, $w$ the waiting time, and $q$ the total volume occupied in the vehicle. A **plan** is a set of consecutive schedules. Let $\Pi_i = \{\pi_0, \pi_1, ..., \pi_e\}$ denote a plan, where $\pi_u =< k_u, a_{k_u}, w_{k_u}, q_{k_u} >$ $(u \in [0, e])$ is the $u$th schedule, and specially $k_0 = k_e = 0$, which denotes the depot.

The **transportation cost** results from travelling distances, and service (pickup or delivery) and/or travelling times. Suppose that a vehicle arrives at a task node $k$ at time $a_k$, hitting its time window, then it spends some time $s_k$ picking up or delivering a parcel, and then arrives at next task node $h$ at time

$$a_h^{new} = a_k + s_k + \tau_{kh} \tag{5.1}$$

, where $\tau_{kh}$ is the travelling time between the node $k$ and $h$. Ideally, $a_h = a_h^{new}$ if $a_h^{new}$ hits the time window at the node $h$. However, due to its constraint of time window, the planned arrival time $a_h$ is accommodated in Equation (5.2).

$$a_h = \begin{cases} a_h^{new}, & a_h^{new} \in [e_h, l_h]; \\ e_h, & a_h^{new} < e_h; \\ \text{infeasible} & \text{otherwise.} \end{cases} \tag{5.2}$$

With only the feasible arrival time considered, the associated cost during this process can be expressed as $c_{kh} = \mu d_{kh} + \nu(a_h - a_k)$, where $d_{kh}$ is the distance from node $k$ to $h$, $\mu$ and $\nu$ are the cost coefficients of distances and times respectively. If $\mu = 0$, only the time-related cost is considered; if $\nu = 0$, only the distance-related cost is

considered. In real practice, two of them are meaningful (the mileage is a type of cost and the working time including waiting time needs to be paid).

The total cost for the transportation plan of the task agent $A_i(i > 0)$ is expressed as $C(\Pi_i) = \sum_{r=0}^{e} c_{k_r k_{r+1}}$ and the total cost for all the vehicle agents is expressed as $C^v = \sum_{i=1}^{m} C(\Pi_i)$, where $m$ is the number of vehicles. The cost calculation in the company agent needs a special consideration because the routing plan for its holding tasks does not exist.

Let $c_t^0$ denote the initial cost of the task $t$. $c_t^0$ is determined by the transportation cost for a vehicle to execute the task alone,

$$t =< v, (k^p, k^d), [e_{k^p}, l_{k^p}], [e_{k^d}, l_{k^d}] > \tag{5.3}$$

i.e., the transportation cost caused by the operations in which the vehicle starts travelling at the depot, picks the parcel with the volume $v$ at the pickup task node $k^p$, delivers the parcel to the delivery task node $k^d$ and returns to the depot finally. $c_t^0$ is expressed as

$$\begin{aligned} c_t^0 &= \mu(d_{0k^p} + \delta_{k^p k^d} + \delta_{k^d 0}) \\ &+ \nu(\tau_{0k^p} + a_{k^d} - a_{k^p} + \tau_{k^d 0}) \end{aligned} \tag{5.4}$$

, where $\tau_{0k^p}$ and $\tau_{k^d 0}$ are the travelling times. Notice that the extra waiting times are eliminated at both the ends of the route. Though a vehicle can start from the depot at the time $e_0$ and return to the depot at the time $l_0$ without violating time windows, the vehicle is forced to arrive at the pickup node $p$ as late as possible and return to the depot

as early as possible, for the purpose of saving unnecessary waiting times. Consequently, the total cost of $A_0$ can be calculated as $C^0 = \sum_{t \in T_0} c_t^0$, and the total cost of all the transportation tasks is the summation of $c^v$ and $c^0$.

Although the total transportation cost for either all the tasks or the tasks allocated to each task agent can be explicitly calculated, DAMM needs the calculation of marginal costs, which drive the virtual market. The **insertion cost** defines the cost increase if a task is inserted into a set of tasks, while **deletion cost** defines the cost decrease if a task is deleted from a set of tasks. In both cases, the transportation tasks are decomposed into task nodes, whose sequences are heuristically arranged. Let

$$c_{T_i}^{\oplus}(t) = C(T_i \cup \{t\}) - C(T_i) \tag{5.5}$$

denote the insertion cost of adding a task $t$ into the set $T_i$ of tasks in $A_i$, where $C(T_i)$ is the cost for $A_i$ to carry out the set $T_i$ of tasks. Let

$$c_{T_i}^{\ominus}(t_i^j) = C(T_i) - C(T_i \setminus \{t_i^j\}) \tag{5.6}$$

denote the deletion cost of removing a task $t_i^j$ from the set $T_i$ of tasks in $A_i$, where $T_i \setminus \{t_i^j\}$ is the task set after the task $t_i^j$ is removed from the task set $T_i$.

The insertion cost and deletion cost for a vehicle agent are calculated based on heuristic algorithms, which are introduced in Section 5.4.2 and 5.4.3. Considered to be a special case, the calculation of marginal cost in the company agent does not need these heuristic algorithms. This is because the task agent representing the transportation

company, i.e. the company agent, only holds a set of transportation tasks temporarily and their insertion costs do not fluctuate. Specifically, the insertion and the deletion costs of a task takes the same value of the initial cost, i.e., $c^{\oplus}_{T_0}(t) = c^0_t$ and $c^{\ominus}_{T_0}(t^j_0) = c^0_{t^j_0}$, where $T_0$ denotes the set of tasks held by the company agent.

### 5.4.2 Insertion cost

To obtain an optimal routing plan of only *one* vehicle for a set of transportation tasks is an *NP-hard* problem [Dumas et al., 1986]; the complexity grows if multiple vehicles seek the optimal solutions repeatedly, which is required in DAMM. Therefore, instead of an exact optimal solution which enumerates all the possibilities and requires exponential complexity, a heuristic solution is proposed and developed based on time feasibility [Solomon, 1987a] first and then capacity feasibility.

#### 5.4.2.1 Time and capacity feasibility

In a feasible transportation plan $\Pi_i$, the constraints of time windows and vehicle capacities must be satisfied at the inserted nodes and all their consecutive nodes, when a new task, which consists of two task nodes for pickup and delivery respectively, is to be inserted into $\Pi_i$. Constraint satisfaction defines feasible conditions. Based on [Solomon, 1987a], we can derive the necessary and sufficient conditions for time feasibility when considering a task node alone; however, we need to consider two task nodes in the task as a whole when deriving capacity feasibility.

Let $g$ denote a task node to be inserted into $\Pi_i$ between the nodes $k_{u-1}$ and $k_u$ ($u \in [1, e]$), which is represented as the relationship of $k_{u-1} \rightarrow g \rightarrow k_u$. Substituting $g$

for $h$ and $k_{u-1}$ to $k$ in Equation (5.1), we get $a_g^{new} = a_{k_{u-1}} + s_{k_{u-1}} + \tau_{k_{u-1}g}$. Then, we can derive $a_g$ based on Equation (5.2). Furthermore, we can derive $a_{k_u}^{new}$ as the arrival time at the node $k_u$ without considering the time window $[e_{k_u}, l_{k_u}]$. If the triangle rule $\tau_{k_{u-1}g} + \tau_{gk_u} \geq \tau_{k_{u-1}k_u}$ is satisfied, $\gamma_{k_u} = a_{k_u}^{new} - a_{k_u} \geq 0$ is true, where $\gamma_{k_u}$ denotes the push back at the node $k_u$. This push back can be propagated to the consecutive nodes of $k_u$ as

$$\gamma_{k_{r+1}} = \max(0, \gamma_{k_r} - w_{k_{r+1}}),\ u \leq r \leq e-1. \tag{5.7}$$

until it becomes zero. Therefore, the necessary and sufficient conditions for time feasibility of $k_{u-1} \rightarrow g \rightarrow k_u$ are:

$$a_g \leq l_g \tag{5.8}$$

$$a_{k_r} + \gamma_{k_r} \leq l_{k_r},\ u \leq r \leq e. \tag{5.9}$$

Time feasibility for a task $t$ can be derived from Equation (5.8) and (5.9). At the same time, the capacity feasibility is to be checked for the nodes between $k^p$ and $k^d$ in the new plan. Let $k_{u^p-1} \rightarrow k^p \rightarrow k_{u^p}$ and $k_{u^d-1} \rightarrow k^d \rightarrow k_{u^d}$ prescribe the insertion relationships in $\Pi_i$. The conditions for time feasibility when inserting $k^p$ are:

$$a_{k^p} \leq l_{k^p} \tag{5.10}$$

$$a_{k_r} + \gamma_{k_r} \leq l_{k_r},\ u^p \leq r \leq e. \tag{5.11}$$

Let $a'_{k_r}$ and $w'_{k_r}$ $(u^d \leq r \leq e)$ denote the arrival and waiting times of the task node $k_r$ after the node $k^p$ is inserted. The conditions for time feasibility when inserting $k^d$ are:

$$a'_{k^d} \leq l_{k^d} \tag{5.12}$$

$$a'_{k_r} + \gamma'_{k_r} \leq l_{k_r},\ u^d \leq r \leq e \tag{5.13}$$

, where $\gamma'_{k_r}$ is the push forward based on $a'_{k_r}$ and $w'_{k_r}$. The conditions capacity feasibility are:

$$q_{k^{u^p-1}} + \Delta q_{k_{u^p}} \leq Q_i \tag{5.14}$$

$$q_{k^r} + \Delta q_{k_{u^p}} \leq Q_i \tag{5.15}$$

$$u^p \leq r \leq u^d - 1$$

, where $Q_i$ is the capacity of agent $A_i (i > 0)$.

#### 5.4.2.2 Heuristic to evaluate insertion cost

The necessary and sufficient conditions derived for feasibility when inserting a task into a plan in the above result in a heuristic search, which evaluates the insertion cost of the task. Besides the steps that use these conditions directly, other necessary steps are introduced, including those of collecting possible insertion positions for both the pickup and delivery task nodes, and those of calculating the cost update after the task is inserted.

The possible insertion positions for a task node $g$ are found out by relaxing the feasibility condition of the arrival time at the task node $g$. Since $e_g \leq a_g \leq a_{k_u}$ for $g$ to be inserted before $k_u$ and $a_{k_u} \leq l_{k_u}$ hold true,

$$e_g \leq l_{k_u} \tag{5.16}$$

must be true. So, instead of checking the feasibility of inserting $g$ using the complete necessary and sufficient conditions in Equation (5.8) and (5.9), we can use Equation (5.16) to filter out some infeasible nodes. The complete conditions will be checked eventually. Thus, the set of possible positions (node sequences), before whose nodes $k^p$ is to be inserted, is defined as

$$\begin{aligned} U_i^p &= \{r | k_{r-1} \rightarrow k^p \rightarrow k_r, \\ &\qquad e_{k^p} \leq l_{k_r}, 1 \leq r \leq e\} \end{aligned} \tag{5.17}$$

and the set of possible positions, before whose nodes for $k^d$ to be inserted after $u^p (u^p \in U_i^p)$ is defined as

$$\begin{aligned} U_i^d(u^p) &= \{r | k_{r-1} \rightarrow k^d \rightarrow k_r, \\ &\qquad e_{k^d} \leq l_{k_r},\ u^p \leq r \leq e\},\ \forall u^p \in U_i^p. \end{aligned} \tag{5.18}$$

This relaxation makes it easy to obtain the insertion pairs for a task without considering the time feasibility conditions.

Assuming that all the feasibility conditions are checked to be true for a pair of task nodes, we can derive the insertion cost by comparing the updated parts of the transportation plans before and after insertion. The distance update is calculated as

$$\begin{aligned} \Delta d &= d_{k_{u^p-1}k^p} + d_{k^p k_u^p} - d_{k_{u^p-1}k_u^p} \\ &+ d_{k_{u^d-1}k^d} + d_{k^d k_u^d} - d_{k_{u^d-1}k_u^d} \end{aligned} \tag{5.19}$$

, where the first line is the distance update for the pickup node and the second line is that for the delivery node, and $d_{k_{u^*-1}k^*} + d_{k^*k_{u^*}}$ ($*$ can be $p$ or $d$) is in the new plan while $d_{k_{u^*-1}k_{u^*}}$ is in the old plan. The time update is calculated as

$$\begin{aligned} \Delta\tau &= a_{k'_{e+1}} + s_{k'_{e+1}} + \tau_{k'_{e+1}k'_{e+2}} \\ &- (a'_{k_1} - \tau_{0k'_1}) \\ &- \{a_{k_{e-1}} + s_{k_{e-1}} + \tau_{k_{e-1}k'_e} \\ &- (a_{k_1} - \tau_{0k_1})\} \end{aligned} \tag{5.20}$$

, where the first line is the total time in the new plan and the second line is that in the old plan, and in each line the first component before the symbol minus is the time without *squeezing* the waiting time of the first task node, while the second component is the suppressed time of the first task node.

For each pair of insertion positions, we obtain an insertion cost. The evaluation process requires that the one with the minimum value is returned as the insertion cost for the task.

The heuristic to find the insertion cost for a transportation task is summarized in Algorithm 1. At the beginning, the current transportation plan is copied so that any insertion and update during this evaluating process will not affect the original plan. At the same time, the insertion cost is set at a positive infinity. Then, the possible positions before whose nodes the pickup node is to be inserted, are collected in $U_i^p$ based on Equation (5.17). For each possible position $u^p$ in $U_i^p$, the algorithm checks the conditions for time feasibility according to Equation (5.10) and (5.11) when the pickup node is inserted before $k_{u^p}$. If these conditions are not satisfied, the algorithm returns with the insertion cost of positive infinity; otherwise, the pickup task node is inserted and the possible positions before whose nodes the delivery node is to be inserted are collected in $U_i^d(u^p)$ based on Equation (5.18). For each possible position $u^d$ in $U_i^d(u^p)$, the conditions for time feasibility are checked in the same way as those for its pickup node. For the successful pair in time feasibility, the algorithm calculates the distance and service time updates and further the cost update. This cost update is compared with the current insertion cost and the minimum one is set as the insertion cost. Notice that two interrelated loops exist in this algorithm to guarantee the minimum cost update.

### 5.4.3 Deletion cost

Compared to the calculation of insertion cost, that of deletion cost becomes less complicated because the new plan without the removed task nodes is still feasible. Generally, it is assumed that $k_{u^p-1} \rightarrow k^p \rightarrow k_{u^p}$ and $k_{u^d-1} \rightarrow k^d \rightarrow k_{u^d}$ prescribe the positions of the pickup and delivery nodes respectively. If $k_{u^p} = k^d$ and $k_{u^d-1} = k^p$ is

**Algorithm 5.1**: The algorithm to find the insertion cost for a task

```
begin
    Copy the plan Π_i^copy = Π_i and set c^⊕(t) = +∞;
    Find U_i^p in Π_i^copy;
    for each u^p ∈ U_i^p do
        Check time feasibility conditions in Equation (5.10) and (5.11) are
        false;
        Insert k^p between k_{u^p-1} and k_{u^p} in Π_i^copy;
        Find U_i^d(u^p) in Π_i^copy;
        for each u^d ∈ U_i^d(u^p) do
            Check time feasibility conditions in Equation (5.12) and (5.13);
            Check capacity feasibility conditions in Equation (5.14) and
            (5.15);
            Calculate Δd and Δτ based on Equations (5.19) and (5.20);
            Calculate the insertion cost Δc = μΔd + νΔτ;
            Compare Δc with c^⊕(t) to set the minimum as c^⊕(t);
        end
    end
    return c^⊕(t);
end
```

true, i.e. no other nodes exist between $k^p$ and $k^d$, the updated distance is calculated as

$$\Delta d_1 = \{d_{k_{u^p-1}k^p} + d_{k^p k^d} + d_{k^d k_u^d}\} - d_{k_{u^p-1}k_u^d} \tag{5.21}$$

, where the first part before the symbol minus is the total distance from the node $k_{u^p-1}$ to the node $k_u^d$ including the nodes $k^p$ and $k^d$ while the second part is the distance from $k_{u^p-1}$ to $k_u^d$ excluding those two nodes. Otherwise, the updated distance is calculated as

$$\begin{aligned} \Delta d_2 &= d_{k_{u^p-1}k^p} + d_{k^p k_u^p} - d_{k_{u^p-1}k_u^p} \\ &+ d_{k_{u^d-1}k^d} + d_{k^d k_u^d} - d_{k_{u^d-1}k_u^d} \end{aligned} \tag{5.22}$$

, which is the same as Equation (5.19).

If the ending node in the plan $\Pi$ is $k_e$, then the ending node in the deleted plan will be $k_{e-2}$ (a task occupies two positions in the transportation plan). The update of service time is still calculated as the difference of service time between the plans before and after deletion. It is calculated as

$$\begin{aligned} \Delta\tau^D &= a_{k_{e-1}} + s_{k_{e-1}} + \tau_{k_{e-1}k'_e} \\ &- (a_{k_1} - \tau_{0k_1}) \\ &- \{a_{k'_{e-3}} + s_{k'_{e-3}} + \tau_{k'_{e-3}k'_{e-2}} \\ &- (a'_{k_1} - \tau_{0k'_1})\} \end{aligned} \tag{5.23}$$

, where $k'_*$s denote the nodes after deletion.

### 5.4.4 Reliability

Since the travelling time along each segment of a transportation route and service time at each location are stochastic, it is not safe to plan the routing schedules of a vehicle according to the mean service and travelling times. To measure the safety of a plan, transportation **reliability** is defined on every consecutive pairs of adjacent schedules in a plan. A pair is called a segment. Suppose that a vehicle arrives at location $i$ at time $\tau_i$ and at location $j$ at time $\tau_j$. The service (delivery plus pickup) time $\Delta\tau_s$ at location $i$ is a random variable with pdf (probability density function) $f(\Delta\tau)$, and the travelling time $\Delta\tau_t$ between $i$ and $j$ is another random variable with pdf $g(\Delta\tau)$. The reliability of finishing this task, i.e. the probability of delivering a parcel and picking up another parcel at location $i$ and travelling from $i$ to $j$, is computed as given in Equation (5.24).

$$R_{ij} = Prob(\Delta\tau_s + \Delta\tau_t \leq \tau_j - \tau_i)$$
$$= \int_{-\infty}^{-\infty} g(\Delta\tau) f(\tau_j - \tau_i - \tau) d\tau \tag{5.24}$$

Along a planned route, each segment travelled requires high reliability. Otherwise, the products to be transported will probably fail to be delivered to the customers as their time windows entail. This requirement is represented as a constraint, saying that the reliability for each segment must be no lower than a threshold value. Let $\alpha$ denote this **reliability threshold** and the constraint is shown in Equation (5.25).

$$R_{ij} = Prob(\Delta\tau_s + \Delta\tau_t \leq \tau_j - \tau_i) \leq \alpha \tag{5.25}$$

In order for the routing plan to be scheduled in the stochastic environment, the arrival time $\tau_j$ is calculated by solving a non-linear equation $R_{ij} = \alpha$. Since $R_{ij}(\tau_j)$ increases monotonically with $\tau_j$'s increase (shown in Fig. 5.4.4) based on its integral effect, $\tau_j$ is calculated according to the inverse function of $R_{ij}$, represented in Equation (5.26), where $F^{NL}$ represents a non-linear mapping from $(\alpha, \tau_i, g, f)$ to $\tau_j$.

$$\tau_j = F^{NL}(\alpha, \tau_i, g, f) \tag{5.26}$$

If the pdfs $f(\Delta\tau)$ and $g(\Delta\tau)$ are just general functions, it is expensive to find $\tau_j$ in Equation (5.26) due to the complexity of convolution in Equation (5.24). The worse situation is that three time variables — the delivery service time, the pickup service time,

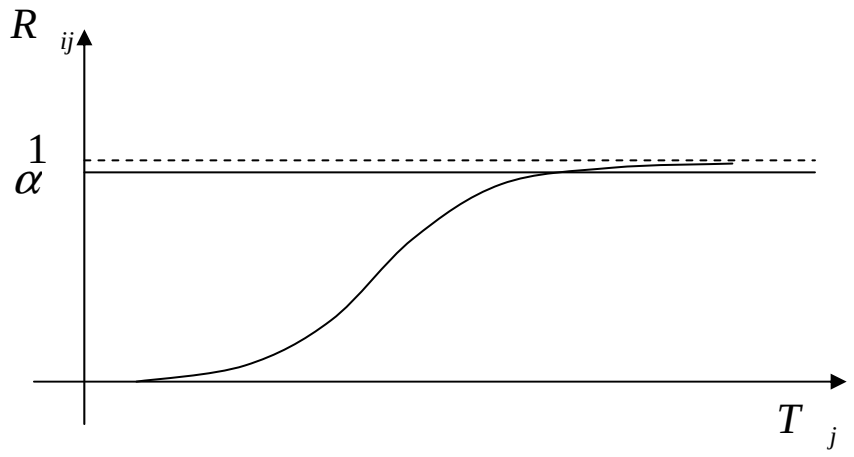


Fig. 5.1. The relationship between the function of $R_{ij}(T_j)$ and the reliability threshold $\alpha$

the travelling time — need to be convoluted to calculate the transportation reliability for a single segment. One convolutional variable adds one dimension of convolutional computation. Since the computational complexity increase exponentially with the number of variables and the sample size for each variable, it is extremely expensive to handle the computation without simplified assumptions.

The PDFs are assumed to be normal distributions, i.e, the delivery service, pickup service, and travelling times are all normally distributed. According to the properties of normal distribution, the summation of three normally distributed variables results in another normal distribution whose parameters can be easily derived from the given parameters. Let $\Delta\tau_d$ be a period of delivery service time $\Delta\tau_p$ be a period of pickup service time, and $\Delta\tau_t$ be a period of travelling time.

$$\begin{aligned} \Delta\tau_d &\sim N(\mu_d, \sigma_d^2) \\ \Delta\tau_p &\sim N(\mu_p, \sigma_p^2) \\ \Delta\tau_t &\sim N(\mu_t, \sigma_t^2) \end{aligned} \tag{5.27}$$

Let $\Delta\tau = \Delta\tau_d + \Delta\tau_p + \Delta\tau_t$ be the summation of these three variables, then

$$\Delta\tau \sim N(\mu_d + \mu_p + \mu_t, \sigma_d^2 + \sigma_p^2 + \sigma_t^2). \tag{5.28}$$

Let $\mu = \mu_d + \mu_p + \mu_t$ and $\sigma = \sqrt{\sigma_d^2 + \sigma_p^2 + \sigma_t^2}$. The Z-score is defined as

$$Z = \frac{\tau - \mu}{\sigma} \tag{5.29}$$

where $Z$ is a standard normal distribution. A normal integral table defining the relationship between $z$ and $\Phi(z)$ exists:

$$P(Z \leq z) = \Phi(z) = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-\frac{v^2}{2}} dv \tag{5.30}$$

The problem is solved reversely. In another word, what is the $\tau$ value, given a $\Phi(z)$ (the predefined reliability)?

It is assumed that the reliability of each link needs to be guaranteed with a enough high value. If we fix this value, e.g. 99%, the above reverse problem can be solved by looking for the "Integral Table of Normal Distribution". Then we can get a value of

$Z$ for this fixed reliability, e.g. it is $z = 2.33$ for $\Phi(z) = 99\%$. With the reliability guaranteed at 99%, we can use $z = 2.33$ to find the corresponding $\Delta\tau$. $\Delta\tau$ can therefore be calculated as follows:

$$\Delta\tau = \mu + 2.33\sigma \tag{5.31}$$

From Equation (5.31), the relationship between the period of mean service time $\mu$ and the period of real service time $\Delta\tau$ to satisfy the reliability threshold is a linear function. In a stochastic environment, the total time required for each segment of a route needs to be scheduled longer than that in a deterministic environment. This is the cost caused by stochastic environment.

## 5.5 Double auction market mechanism

In an economic market, many sellers and buyers exchange commodities for their own interests. Over a long period of time, the market evolves to be stable due to these continuous exchanges. The relationship between sellers and buyers can be one-to-one (bargaining), one-to-many (auction), and many-to-many, e.g. double auction. In specific, we are interested in the double auction market mechanism, which designates the relationship between many sellers and many buyers in a trading institution. With this mechanism, buyers and sellers submit their proposals at the same time and the market specialist coordinates the interaction such that suitable pairs of one seller and one buyer are chosen to trade. The double auction mechanism proved to be successful in global goods and stock markets [Friedman, 1993].

By imitating the invisible hand behind the double auction market system, researchers attempted to build virtual markets with computer software to solve engineering problems [Walsh and Wellman, 1999; Wellman, 1992], which cover a wide range of discrete (combinatory) optimization problems, especially those of resource and task allocation. In these problems, some entities emulate traders (decision makers), some emulate products and money. For example, in a resource allocation problem, resource requesters are modelled as decision makers, resources are modelled as products and the value of resources are modelled as money. Through the continuous exchange of products and money among the traders, the solution to the optimization problem is improved and finally stays at a level of Pareto Optimality.

A virtual market can be implemented in a distributed information systems — a Multi-Agent System (MAS), in which the traders are modelled as distributed software agents and they communicate via computer networks. Why? First of all, it is a natural way to employ a MAS to implement the virtual market due to the clear definition of distributed interactions among the traders. Additionally, some good features such as reducing complexity, improving computational efficiency, and being adaptive to a scalable and dynamically changing environment can be achieved with cheap communication costs in MAS. According to the literature, this virtual market model with a distributed information system has been used in several applications of resource and service allocation [Walsh and Wellman, 1998; Rajan et al., 1997].

In the scenario of distributed supply chain procurement, double auction can be the computational market mechanism in which a group of selling agents and another group of buying agents attempt to exchange transportation tasks, virtually based on the

designed market rule. This rule benefits both the sellers and buyers in an institution, where a selling agent with his highest ask (the selling request with the highest deletion cost) and a buying agent with his lowest bid (the buying request with the lowest insertion cost) are suggested to exchange their proposed tasks. According to the suggestions, the selling and buying agents update their commitment relationships with those proposed tasks.

Besides the two types of task agents (a transportation company agent and vehicle agents), another agent called *monitor agent* is introduced to the market system. Let $M$ denote this agent, which is a virtual agent that does not correspond to any physical facility. The monitor agent applies the market rule to the collected asks and bids.

Double auction market mechanism regulates a series of interactions, round by round. The interactions between the different rounds are similar, and the interactive process continues till convergence. During one round of interaction, the following are sequentially executed: (1) the task agents propose asks and bids simultaneously, (2) the monitor agent suggests agreements based on the market rule and (3) the task agents update their plan according to the suggested agreements. In order for the market mechanism to work properly, the market is cleaned up at the beginning of every round such that the new round of interactions is not affected by the previous interaction. In the market cleanup, all the asks and bids proposed by task agents are deleted at the beginning of a new round. At the same time, other state variables such as the round benefits which influence the transaction will be initialized. At the end of each round, the converging condition is checked to see whether the iteration needs to be terminated or not.

In a multi-agent distributed system, the procedures above will be strictly followed. However, without a global control, a systematic coordination of these interactions becomes a necessity. There are two possible coordination methods: synchronous and asynchronous coordination. If each agent has the knowledge of the number of agents in the community, he can wait to make his own decision until all the other agents have made their decisions. This coordination is deterministic and the interactions among the agents will definitely be realized as planned. Since each agent is guaranteed to work cooperatively with other agents, logical synchronization exists among them. On the other hand, the communication among these agents is not guaranteed if the number of agents in the community is not known or it is impossible to be known. Some waiting time between adjacent procedures has to be designed with the expectation that the necessary information for the next step of decision will be collected during this waiting period. If slow processors or slow network connections exist, agents will have to make decisions without complete information. This could bring inaccurate computational results. The interactions relying on the designed waiting time is termed as asynchronous interactions. In this paper, only synchronous coordination is considered.

### 5.5.1 Propose asks and bids

Each task agent $A_i$ needs to make the following decisions sequentially to propose a ask or bid:(1) whether to sell, or buy a task or keep silent, (2) which task to sell or buy from a list of tasks,(3) the cost estimation as an ask or bid to be submitted. These three decisions have to be made sequentially and each previous decision is a prior to its subsequent decision. Because each agent does not have enough information to build a

deterministic model to make Decisions (1) and (2), stochastic models with probability parameters are developed. For decision (3), there is an approximate and deterministic decision model, i.e. the evaluation of marginal costs (see Section 5.4). These costs can be explicitly calculated by calling the algorithms "tryInsertionCost" and "tryDeletionCost" based on the information of the previous two decisions. This true valuation is according to the principles of auction *mechanism design* in which the true values are risk-neutral to bidders [Varian, 1995].

Three possible actions, *sell, buy, and silent*, are predefined with a discrete uniform distribution, in which each action corresponds to a probability of $\frac{1}{3}$. The decision on whether to sell, buy or keep silent is made through generating a *random action* based on this probability distribution.

After the action decision is made, a task needs to be chosen to be sold or bought. A stochastic model is developed to realize this. By calculating the marginal costs of a list of relevant tasks, a probability representing the incentive to sell or buy is assigned to each task. The calculation of each probability is based on the relative values of the marginal costs. The decision about which task to be sold or bought is made through generating a random variable based on this probability distribution. The decision process of selling a task is different from that of buying a task.

The decision to sell a task is the process of selecting a proper task to sell from a set $T_i$ of tasks. During this process, the deletion cost of each individual task in $T_i$ is first calculated, and then the probability distribution is calculated according to Equation (5.32), where $p^{\ominus}_{T_i}(t_i^h)$ is the probability of selling a task $t_i^h$ from the task set $T_i$.

$$p^{\ominus}_{T_i}(t^h_i) = \begin{cases} \frac{1}{m_i}, & \text{if } C^{\ominus}_{T_i}(t^h_i) = 0 \\ \frac{C^{\ominus}_{T_i}(t^h_i)}{\sum_{j=1}^{m_i} C^{\ominus}_{T_i}(t^j_i)}, & \text{otherwise} \end{cases} \quad \forall h = 1, ..., m_i. \tag{5.32}$$

The buying decision is not be made until all the selling decisions are made and the monitor agent informs all the buying agents about the selling list of tasks. Let $T^{sell} = \{t^h_k \mid k \in [1, n], h \in [1, m_k]\}$ denote the selling list. In this decision process, the feasibility of inserting each task in $T^{sell}$ is checked at first. Let $T^{buy}_i$ denote the set of tasks that pass this feasibility checking (see Equation (5.33)). Then the benefit (the difference between the selling marginal cost and the buying marginal cost) of each task in $T^{buy}_i$ is calculated and compared to find the minimal benefit $\eta$ (see Equation (5.34)). The buying probability distribution on $T^{buy}_i$ is shown in Equation (5.35), where $p^{\oplus}_{T^{buy}_i}(t^h_k)$ is the probability of choosing a task $t^h_i$ to buy from the task set $T_i$.

$$\begin{aligned} T^{buy}_i &= \{C^{\ominus}_{T_k}(t^h_k) \geq C^{\oplus}_{T_i}(t^h_k), \\ &\quad \forall t^h_k \mid t^h_k \in T^{sell}\} && (5.33) \\ \eta &= \min_{\forall t^h_k \in T^{buy}_i} \{C^{\ominus}_{T_k}(t^h_k) - C^{\oplus}_{T_i}(t^h_k)\} && (5.34) \\ p^{\oplus}_{T^{buy}_i}(t^h_k) &= \frac{A}{B}, \ \forall t^h_k \in T^{buy}_i, \text{ where} && (5.35) \\ A &= C^{\ominus}_{T_k}(t^h_k) - C^{\oplus}_{T_i}(t^h_k) - \eta \\ B &= \sum_{\forall t^h_k \in T^{buy}_i} \{C^{\ominus}_{T_k}(t^h_k) - C^{\oplus}_{T_i}(t^h_k) - \eta\} \end{aligned}$$

### 5.5.2 Suggest transactions

After collecting all the proposals submitted by all the task agents in one round, the monitor agent needs to find suitable matches that imply the transactions between sellers and buyers. This is not a decision process because it does not provoke any marginal cost. However, it is a necessary procedure because all the task agents need the suggestion to make the final decision: whether to commit to their proposals or not. The whole process of suggesting includes three steps. Although these three steps are implemented in an iterative way, they are expressed as a series of aggregate operations. Briefly, these three steps are: (1) to cluster all the proposals according to the order ID (Different tasks may share the same order ID if they are ordered by the same customer); (2) to sort the selling proposals in a descending order and buying proposals in an ascending order both by the values of marginal costs respectively for each type of transportation task (the same starting and ending locations, and the same time windows), (3) to choose the top proposals from both the selling and buying proposal list to generate a suggested transaction for each cluster.

### 5.5.3 Update commitments

Although it is true that a seller or a buyer could discard the suggestion by the monitor agent in a dynamically changing environment, the task agents in this mechanism are designed to follow the suggested transactions strictly. The details are shown in Section 5.4.2 and 5.4.3.

### 5.5.4 Convergence criteria

In an ideal market where each trader has complete information, the convergence criteria can be set as the state where no-one has the inclination to buy or sell. Although the model is made up of incomplete information and employs stochastic processes to propose asks and bids, the convergence criteria based on the spirit of the ideal market can be derived. As a matter of fact, it was found that the interests of the task agents reflected in the monitor agent indicates the convergence criteria.

The state where the buying agents in the market have no inclination to buy any task proposed by the selling agents is a hint that the market is stable because no transactions will happen in this round. In order to avoid the influence of the selling agents proposing wrong tasks, the hints indicating stability need to observed for a period of time. Thus, a threshold regulating this period needs to be predefined in order for the convergence criteria to be checked.

## 5.6 Computational experiments

Our computational experiments are carried out using Solomon's [Solomon, 1987b] vehicle routing problems and their extensions, with the variation that transportation tasks come sequentially. The sample problems are formulated based on some reasonable randomization, where the pickup and delivery locations spread in a geographical network, the time window of each task node spanning within a certain time range, the number of parcels in an order are chosen from integer numbers, and the volume of each parcel from a set of real numbers. With the assumed normal distribution on the service time and

travelling time, the stochastic part of the problem is transformed into a deterministic problem with the idea of reliability threshold. The distance between any two locations is approximated by their Euclidean distance. Since the notification of a transportation order is released gradually, the dynamics of the distributed problem cannot be easily handled in a traditional vehicle routing solver, which is based on integer programming or other heuristic algorithms. Therefore, the results in this paper will not be compared with the results of the vehicle routing solvers.

The auction protocol is a preliminary dynamic and distributed algorithm, without the ability for re-planning. Once a new transportation task arrives, it will be allocated to the vehicle which offers the least cost and then this allocation is fixed. This process is characterized as an auction, more precisely, the Reverse English Auction, in a virtual market. In another aspect, auction is a negotiation protocol based on CNP, which in this paper we call as Auction Market Mechanism (AMM), which is also the method used in Satapathy [1999].

DAMM can be incorporated in two different ways. First, DAMM is introduced after all the transportation tasks are allocated to the vehicle agents (assuming enough vehicle capacities). This incorporation is identified as the incomplete DAMM. On the other hand, an iteration of auction and another iteration of double auction run alternatively. Since an auction can be considered as a special double auction, this incorporation is identified as the complete DAMM. In the computational experiments, we mainly identify these three distributed negotiation protocols: AMM, incomplete DAMM and complete DAMM. We examine the sensitivities of these three algorithms to the variations

of the task sequences, and the scalability on how DAMM can be adaptive to the large number of transportation tasks.

DAMM is implemented in Java with JDK1.4.1. Instead of building a distributed system where agents communicate through networks, a virtual platform where the agents communicate through memory, is developed for the purpose of validation of the mechanism. The logics, algorithms and components in the virtual system can be easily wrapped as software agents and deployed in an internet/intranet based multi-agent platform.

### 5.6.1 Test problem

The test problem has 25 vehicles, each with the capacity of 200 units of volume, and 53 transportation orders, each with one parcel that occupies 10, 20, 30, 40 or 50 units of volume. Thus, 53 transportation tasks are to be allocated. Fig. 5.2 shows the distribution of these tasks in an artificial geographical area. The center dot with coordinates $(40, 50)$ represents the depot location, where each vehicle starts and ends its routine. Other dots are the pickup or delivery locations extracted from the requests of the customers. The transportation orders are already split into the transportation tasks, which are represented by arrows, each of whose tails designate a pickup location and each of whose heads a delivery location. If a dot is not connected with an arrow, the pickup and delivery locations in the same task coincide. Time windows are not displayed explicitly, but each pickup and delivery task node is attached with a time window.

To gain intuitive understanding, we first present the results of task allocation and evolutionary history for a specified task sequence. Fig. 5.3 shows the results of task

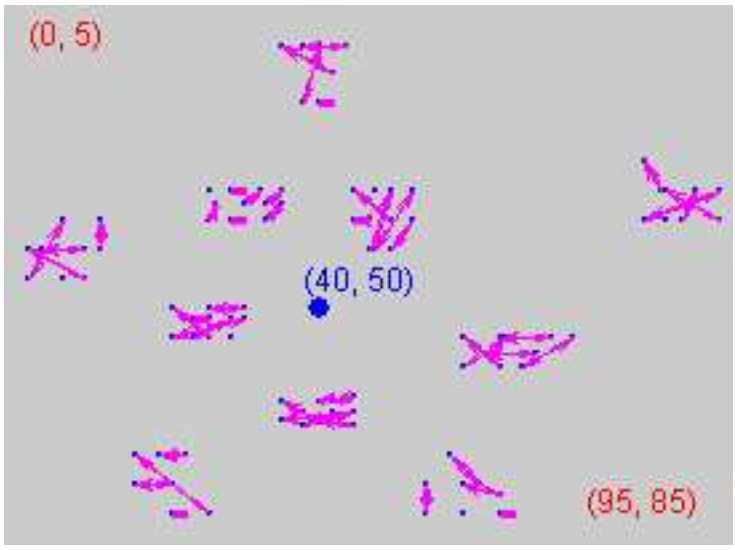


Fig. 5.2. A sample supply chain procurement problem with 25 vehicles and 53 transportation orders

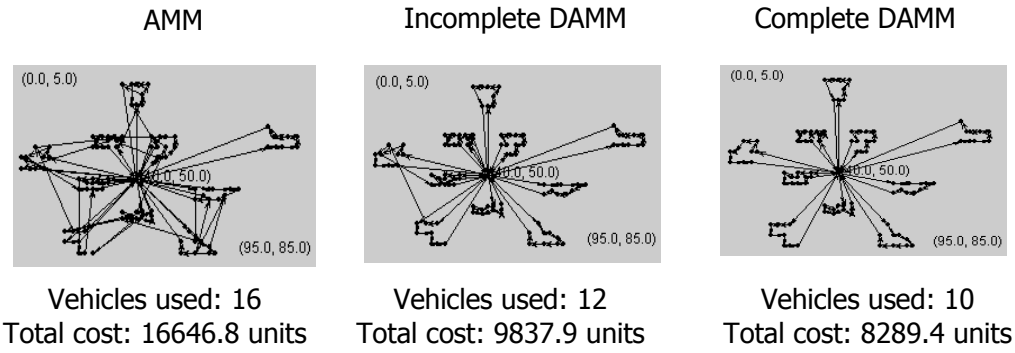


Fig. 5.3. The results of task allocations with routing plans of the special sequence for three distributed algorithms:AMM, incomplete DAMM, and complete DAMM

allocations, respectively for each distributed algorithm. The results of vehicle utilizations (the number of the vehicles scheduled to execute the tasks), routing plans (the geographical distribution of the tasks for each vehicle), and the value of objective function (the final cost level) are presented for the purpose of comparison. Furthermore, the evolutionary histories of the total cost are shown in Fig. 5.4.

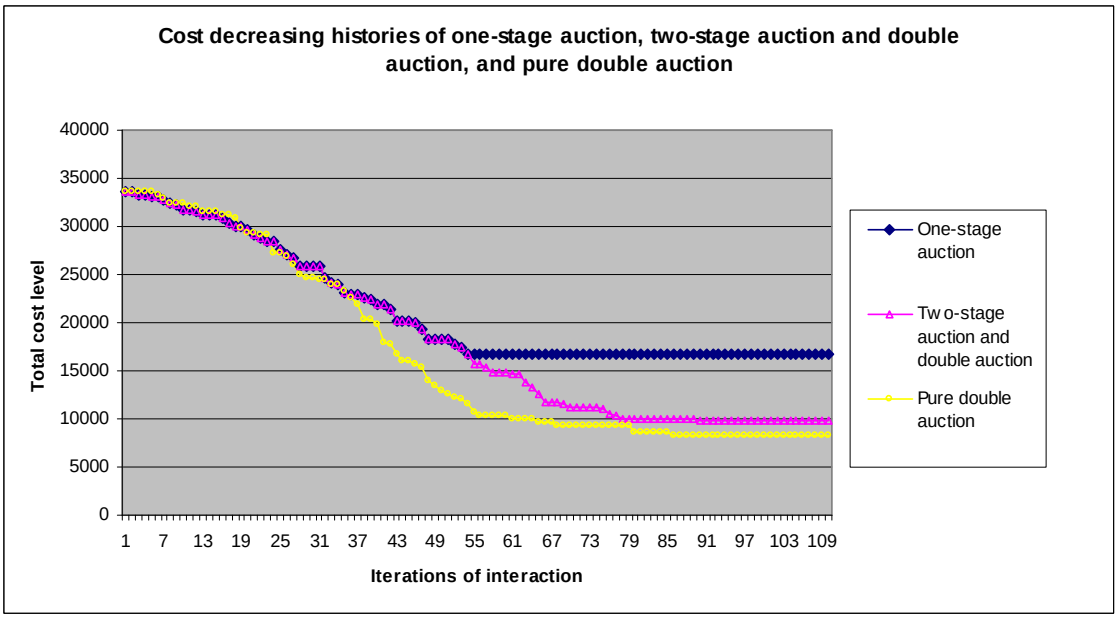


Fig. 5.4. The convergence history of the total cost for data set 25 for market mechanisms of AAM and DAMM

### 5.6.2 Sensitivity analysis

We design a set of computational experiments to study sensitivities of the three algorithms to the variations of task sequences. Through shuffling a set of transportation tasks, we obtain a series of testing problems, with the same set of tasks but different

sequences. Let $N_t$ denote the number of the testing problems. To address the stochastic heuristic search in DAMM, we run each algorithm on each testing problem 10 times, with totally $30 * N_t$ computational experiments. We collect the total transportation cost for each experiment. For the purpose of comparison, we either choose the average or minimum among the 10 results with the same algorithm and task sequences. Fig. 5.5 presents comparison for the average total transportation cost and Fig. 5.6 for the minimum. Table 5.1 summarize the means and variances of the numerical values for both cases.

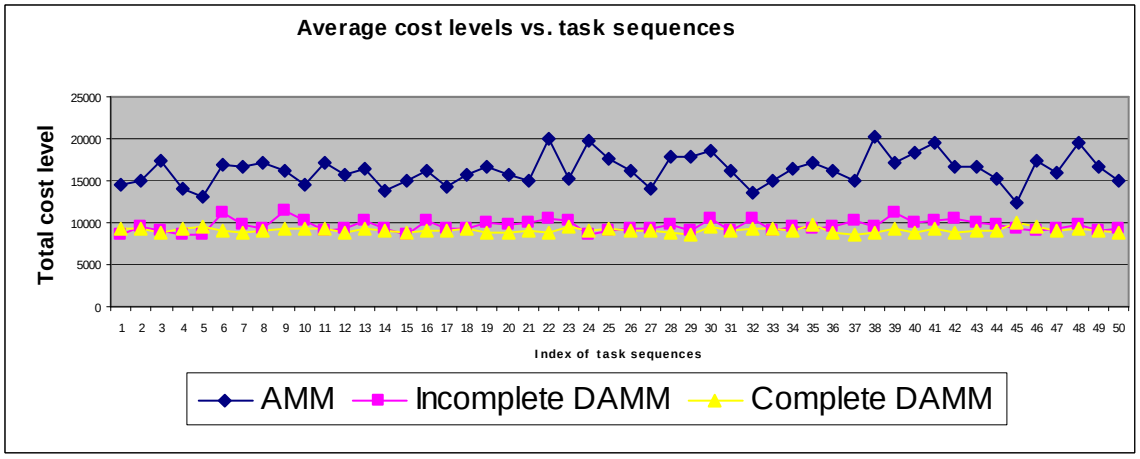


Fig. 5.5. The sensitivity of average cost levels to task sequences for the chosen experiments

To measure how much improvement that DAMM offers, we define the measurement called **improvement ratio**, which is the fraction of the reduced cost over the cost of the AMM. Let $c_i^{AMM}$ denote the cost level of data set $i$ calculated using the AMM, $c_i^{DAMM}$ the corresponding cost level using the DAMM. The improved ratio is calculated

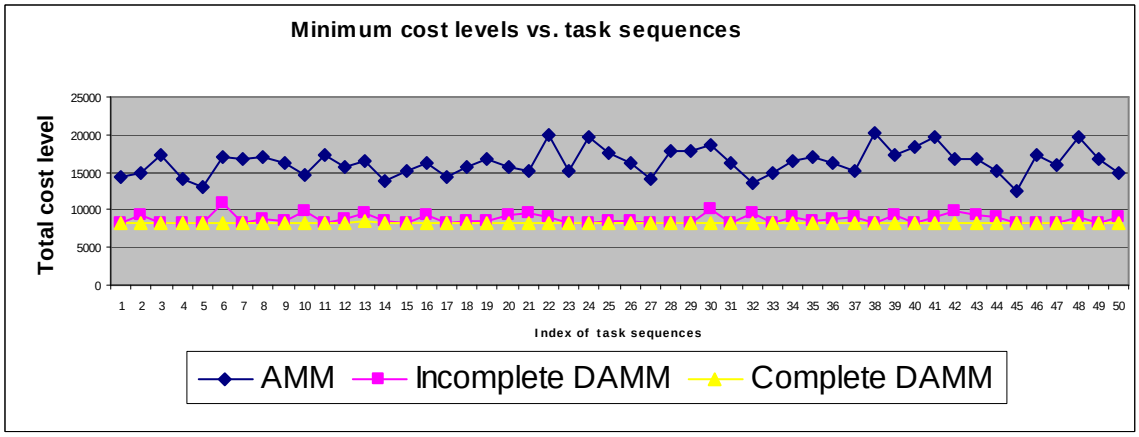


Fig. 5.6. The sensitivity of minimum cost levels to task sequences for the chosen experiments

Table 5.1. Statistic comparison of different algorithms on the cost level for 53 tasks

| Algorithms | AMM | Average cost of DAMM | | Minimum cost of DAMM | |
|---|---|---|---|---|---|
| | | Incomplete | Complete | Incomplete | Complete |
| Mean | 16300.8 | 9625.5 | 9093.0 | 8793.9 | 8297.1 |
| Variance | 1788.6 | 695.9 | 278.1 | 591.5 | 47.2 |

as:

$$\rho_i^{DAMM} = \frac{c_i^{AMM} - c_i^{DAMM}}{c_i^{AMM}}$$

Since we know the optimal solution for the test problem, we measure how much DAMM fills the gap between AMM and optimal cost levels. Thus we define the measurement called **optimality ratio**, which is defined as the fraction of the cost reduction offered by DAMM over the possible maximum cost reduction. Let $c_{opt}$ denote the optimal cost for the sample problem. The optimality gap the DAMM is calculated as:

$$\varphi_i^{DAMM} = \frac{c_i^{AMM} - c_i^{DAMM}}{c_i^{AMM} - c_{opt}}$$

Fig. 5.7 represents the comparison of these improvement and optimality ratios, and Table 5.2 summarizes the means, variances, and extremes of those ratios, corresponding to the average cost. Likewise, Fig. 5.8 and Table 5.3 shows the ratios corresponding to the minimum cost.

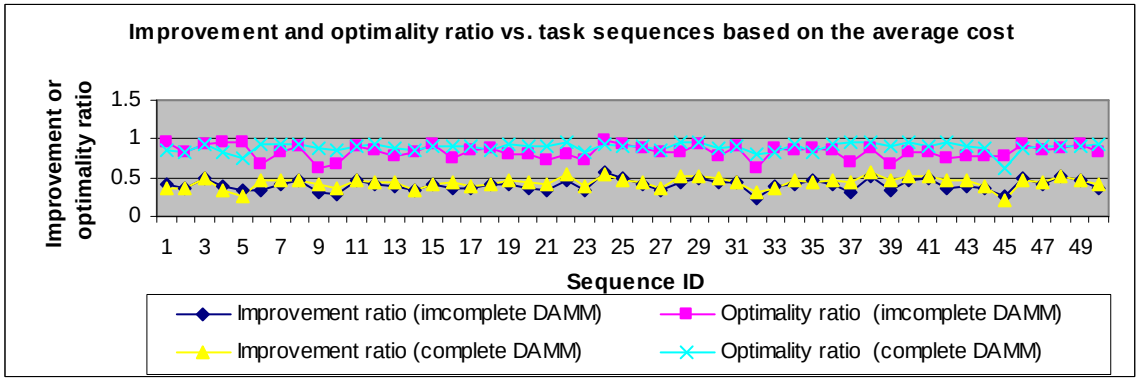


Fig. 5.7. The sensitivity of improvement and optimality ratio to task sequences for the chosen experiments based on the average cost level

Table 5.2. Comparison of different algorithms on improvement and optimality ratios based on the average cost level

| Algorithms | Incomplete DAMM | | Complete DAMM | |
|---|---|---|---|---|
| Ratio types | Improvement | Optimality | Improvement | Optimality |
| Mean | 0.40 | 0.83 | 0.43 | 0.89 |
| Variance | 0.07 | 0.09 | 0.07 | 0.06 |
| Minimum | 0.24 | 0.61 | 0.20 | 0.61 |
| Maximum | 0.57 | 0.98 | 0.57 | 0.96 |

Table 5.3. Comparison of different algorithms on improvement and optimality ratios based on the minimum cost level

| Algorithms | Incomplete DAMM | | Complete DAMM | |
|---|---|---|---|---|
| Ratio types | Improvement | Optimality | Improvement | Optimality |
| Mean | 0.45 | 0.94 | 0.48 | 1.00 |
| Variance | 0.07 | 0.08 | 0.06 | 0.01 |
| Minimum | 0.30 | 0.71 | 0.33 | 0.96 |
| Maximum | 0.59 | 1.00 | 0.59 | 1.00 |

### 5.6.3 Large-scale Experiments

We also design the experiments for large-scale problems, where the number of tasks are increased to 100, 200, 300, 400 and 500, with the increase of vehicle availability accordingly. The results with the AMM and complete DAMM are compared, and their comparisons are summarized in Table 5.4.

### 5.6.4 Summary

By observing the geographical distribution of transportation plans and the evolutionary history of cost levels for a specified task sequence (Section 5.6.1), the sensitivities of DAMM to the task sequences on cost level, improvement ratio and optimality ratio

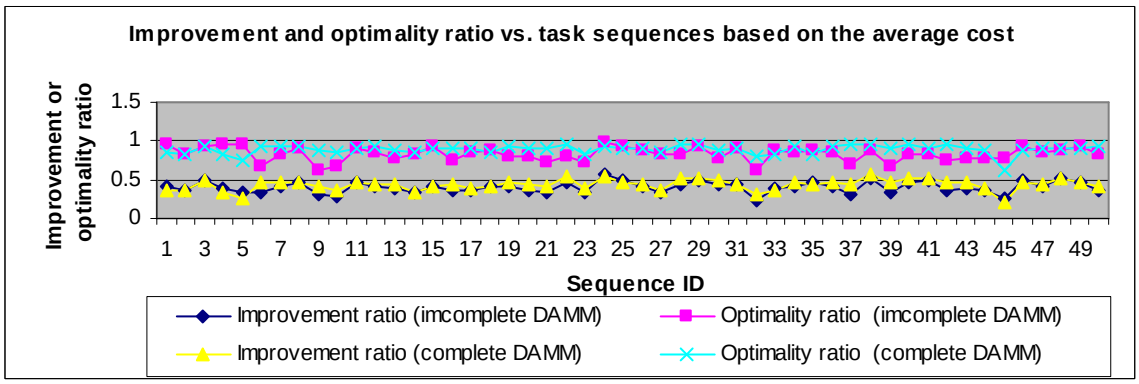


Fig. 5.8. The sensitivity of improvement and optimality ratio to task sequences for the chosen experiments based on the minimum cost level

Table 5.4. Comparison of cost reduction of DAMM for large-scale problems

| # of tasks | AMM | | DAMM | | Improved By (%) |
|---|---|---|---|---|---|
| | Cost | Vehicles | Cost | Vehicles | |
| 100 | 48890.2 | 29 | 27045.7 | 20 | 45 |
| 200 | 158547.6 | 62 | 77315.9 | 43 | 51 |
| 300 | 316287.8 | 98 | 148754.9 | 64 | 53 |
| 400 | 540634.2 | 130 | 286476.5 | 88 | 47 |
| 500 | 924905.3 | 171 | 452657.7 | 107 | 49 |

(Section 5.6.2), and the improvement offered by DAMM to solve large scale problems (Section 5.6.3), it can be concluded that the double auction mechanism is an efficient distributed negotiation protocol for supply chain procurement problems in a dynamic and unpredictable environment. At the same time, it can concluded that the complete DAMM outperforms the incomplete to some extent.

Firstly, the instinctive observations from Figs 5.3 and 5.4 show that the DAMM heuristic clusters the transportation tasks in the manner of saving transportation costs to a large extent. In Fig. 5.3, it is observed that the DAMM improves the solution by reorganizing the locations to visit almost in the most convenient way. In Fig. 5.4, it is observed that the DAMM pushes the total transportation cost of the company lower and lower and finally makes it converge to a fixed value. However, the convergence property has not been shown in the cost history using the AMM. Comparing the results of incomplete and complete DAMM, our computation demonstrates that the complete DAMM offers even better results than the incomplete DAMM.

Secondly, as shown in Figs. 5.5 and 5.6, and Table 5.1, DAMM produces lower mean and variance than AMM on the total transportation cost. This implies that DAMM does not only improve the efficiency of task allocations, but also is not so sensitive to task sequences as AMM. In accordance to the above instinctive observation, the performance of the complete DAMM is still better than that of the incomplete DAMM. As shown in Fig. 5.7 and 5.8, and Table 5.2 and 5.3, the computation of improvement and optimality ratios demonstrate the same results. For most task sequences, their improvement ratios are above 40% and their optimality ratios lie between 90% and 100%. Since the task reallocation occurs frequently in a dynamic environment due to the arrival of new tasks or

contingent tasks because of failure, this property demonstrates that the DAMM provides anytime capability in solving dynamic problems for a distributed and dynamic supply chain procurement system.

Finally, the cost reduction for large-scale problems (Table 5.4) demonstrate the ability of the performance improvement of DAMM. Compared to AMM, DAMM reduces the total cost of AMM by at least 45% and possibly more than 50%. This compares favorably with the results for smaller problems shown in Table 5.2 and 5.3. Therefore we can induce that DAMM is a good candidate for solving large-scale distributed and dynamic problems.

## 5.7 Analysis

The above empirical results show the efficiency of the DAMM for solving distributed supply chain procurement problems. A theoretical proof is beyond the scope of this paper. Therefore some explanations are provided on the working mechanisms of DAMM in this section. Furthermore, the characteristics of this mechanism are summarized.

### 5.7.1 Working mechanism

Intuitively transaction rules and random optimization may be reasons for DAMM's results.

#### 5.7.1.1 Transaction rules

From the point of view of the transactions rules in Section 5.5.2 and 5.5.3, the task agents benefit from *exchanging tasks sequentially* during the negotiation process.

In a round, the task $t$ in agent $A_1$ is expensive to execute, but agent $A_2$ can do it cheaply. Then it is beneficial for both agents if $A_1$ breaks its commitment with $t$, and $A_2$ commits to the task $t$. In such a case, both $A_1$ and $A_2$ obtain the benefits of half of the cost saving. When this process continues, sequential transactions can be executed to make the total cost of the transportation company lower and lower. This is the basic exchanging scenario in DAMM.

#### 5.7.1.2 Random optimization

Although the optimization solution is sought when the double auction negotiation protocol is applied to solve the supply chain procurement problem, it is very difficult to build an analytical model to evaluate how closely the double auction mechanism approximates the optimal solution. In fact, the double auction mechanism is an efficient heuristic based on random optimization. In this section, the theory of random optimization [Pflug, 1996] is employed to show the convergence of this double auction (DA) heuristic.

A vector $\mathbf{x}$ is defined to represent the state of the task allocation in the problem. The length of $\mathbf{x}$ is the the number of tasks to be allocated. The index of an entry in $\mathbf{x}$ is defined as a task ID. The content of an entry of $\mathbf{x}$ is filled with the ID of the vehicle which is committed to executing the task. In other words, the vector $\mathbf{x}$ denotes the task allocation information among a set of vehicles. Any optimization process working on the problem needs to find a vector $\mathbf{x}_{\hat{opt}}$ such that all the constraints are satisfied and the disturbance to $\mathbf{x}_{\hat{opt}}$ worsens the quality of solution.

The DA heuristic provides a mechanism with which to update the vector $\mathbf{x}$ thus improving the solution. In the double auction mechanism market, each task agent wants to maximize his payoff by proposing asks or bids stochastically without caring about the proposals of others. However, the monitor agent conservatively only allows the beneficial proposals to be transacted. The repetition of the process will not stop until transactions are no longer possible in the market. This exactly pushes the whole market migrate toward an optimal solution $\mathbf{x}_{opt}$, which is assumed to be available through strict mathematical programming.

On the other hand, the whole DA heuristic can be considered as a process of random optimization, where the randomness is defined in the process of proposing tasks to sell or buy as shown in section 5.5.1. The convergence of random search is proved with an infinite number of trials, based on the fact that an I.I.D (Identical Independent Distribution) random sequence hits eventually every set of positive measures [Pflug, 1996]. Although the process can not run infinitely in real computation, the global cost becomes smaller and smaller and finally fixed on a value as the market interactions continue.

### 5.7.2 Characteristics

The DA heuristic exhibits several important characteristics such as scalability, anytime capability, distributed deployment ability, and adaptability to discrete optimization problems, which adds more value to this methodology applied to distributed optimization.

#### 5.7.2.1 Scalability

When the DA heuristic is applied in solving the supply chain procurement problem, the numbers of both attending vehicles and transportation tasks are flexible. Any vehicle, which needs to be modelled as a software agent, can join the market at any time and any vehicle can also recede from the market if he has sold out all of his tasks. New transportation tasks decomposed from new customer orders can be immediately brought to the market without any changes in the algorithms. Due to the flexibility of the heuristic, the system built with the DA heuristic is scalable with the number of attending vehicles and transportation tasks. This enables the DA heuristic to solve large-scale problems.

#### 5.7.2.2 Anytime capability

Due to the reason that DA heuristic can be employed to improve an existing allocation and the next round solution is better than its previous one, this heuristic creates a very important property: *anytime*. In a real-time environment, where a quick response is important, the allocation result can be executed at anytime; at the same time, the allocation result excluding the executed tasks can be improved if any computational time is available. With the support of information systems, the DA heuristic can be effectively applied to real-time problems.

On the other hand, the anytime DA heuristic can take care of the contingencies caused by any failure of the transportation system. The failed transportation task can be prompted to the DA market immediately and the negotiation protocol directs this task to a cheap vehicle's routing plan. This process is based on the current routing plan

and it is not necessary to rerun the whole allocation process. The anytime DA heuristic can help the transportation system deal with some critical and unexpected situations.

#### 5.7.2.3 Distributed deployment

Since the DA heuristic was developed to solve the procurement problem among a group of physically distributed vehicles in real time, it can easily be deployed in a distributed information system, especially a multi-agent system. Also, it can be effectively deployed in parallel computational environments such as PC clusters. Consequently, the DA heuristic provides an efficient way to solve a large scale problem by exploiting the computational capacity via LAN (Local Area Networks) and the Internet.

#### 5.7.2.4 Adaptability

The DA heuristic can be a general heuristic method for solving discrete optimization problems. If it can be determined that who are buyers and sellers, what goods and money or cost are, the DA heuristic can be applied in a straightforward manner. Discrete optimization problems, especially resource allocation and task allocation problems, are easily modelled using this heuristic because the actual facilities can easily correspond to the sellers and buyers. For example, in an application that a group of workers work on different types of jobs, we can model it in two different ways using double auction heuristic. In the first choice, the workers can be modelled as traders, the jobs as goods, and the time spent on the jobs and their urgency as cost. In the second choice, the jobs are modelled as traders, and the time slots of the workers as goods. Which way to go depends on the characteristics of the problem. Usually we hope the traders to be fixed

and goods to be dynamic. If we have a fixed groups of workers but a large number of jobs, we choose the first one. If we have a fixed set of jobs but many workers with their time slots, we choose the second one.

## 5.8 Conclusions and future work

In this paper, a double auction market mechanism is developed to enable the negotiation process of exchanging transportation tasks among a group of distributed transportation facilities in a multi-agent distributed information system.

Firstly, a supply chain procurement problem is successfully mapped to a market-based negotiation problem. In this mapping, vehicles and transportation companies were modelled as task agents and a virtual monitor agent was introduced as a market specialist into the market; transportation tasks are modelled as goods and saved marginal cost as money. This mapping leads to the development of efficient heuristics to solve the NP-hard distributed transportation procurement problems. This also paves a new path to solving resource allocation and task allocation problems.

Secondly, a negotiation protocol based on the double auction mechanism, which is also called the DA heuristic, is developed. In this protocol, the task agents can repeatedly exchange tasks based on a stochastic process in order for a better allocation to be obtained. The DAMM is developed to handle different situations. In the DAMM, the transportation company without the ability to execute tasks joins the market so that the double auction market mechanism can start from an infeasible solution. This provides a new view on the heuristic optimization.

Finally, a series of computational experiments are carried out on a test problem and DAMM was further applied to large-scale problems. Our computational results demonstrates that DAMM greatly improves the solution of AMM either from an existing allocation or not. At the same time, the gap between the objective values obtained by AMM and the optimal solution is shortened when DAMM applied.

For future work, the following research directions are promising. Firstly, a learning method based on the DA heuristic can improve the allocation. While the DA heuristic can improve a solution immensely starting at either an infeasible solution or a feasible solution, the task agents do not have occasions to regret the decisions they made. This is because the total cost function is monotonically decreasing during the process of the market interaction. To make the task agents be regretful for their decisions, a simulation system needs to be designed so that the DA heuristic can be repeated starting from any initial allocation. In addition, a machine learning method needs to be developed to collect and train using the simulation results. Once the convergence patterns are found, optimal decisions can easily be sought within the trained knowledge. Reinforcement learning may be a good candidate. Secondly, a real multi-agent system can accelerate the computation by utilizing its powerful paralleling computational ability. JADE may be a candidate platform. Finally, it would be profitable to develop different proposing strategies for each task agent. In the current system, we designed the simplest mechanism. In the future, we would like to consider developing more complicated mechanisms. For instance, each proposal could contain more than one task; the task agents could propose using different strategies; the task agents could play a fictitious play. A comprehensive analysis would be a necessity to evaluate these alternatives.

## Bibliography

A. Bachem, W. Hochstattler, and M. Malich. The simulated trading heuristic for solving vehicle routing problems. Technical Report 93.139, Mathematisches Institut, Universitat zu Koln, Weyertal 80, 50931 Koln, Germany, 1993.

D. Brugali, G. Menga, and S. Galarraga. Intercompany supply chains integration via mobile agents. Jacucci, G. (ed.), Globalization of Manufacturing in the Digital Communications Era of the 21st Century: Innovation, Agility and the Virtual Enterprise, Kluwer Academic Publisher, 1998.

J. Q. Cheng and M. P. Wellman. The walras algorithm: A convergent distributed implementation of general equilibrium outcomes. *Computational Economics*, 12:1–24, 1998.

C. Cook and K. Hamlin. Manufacturing a supply chain. *EAI Journal*, May 2000.

R. Davis and R. G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20:63–109, 1983.

J. Dumas, Y. Desrosier, and F. Soumis. A dynamic programming solution of a large scale single-vehicle dial-a-ride problem with time windows. *American Journal of Mathematical and Management Science*, 6:301–325, 1986.

K. Fischer, J. P. Muller, M. Pischel, and D. Schier. A model for cooperative transportation scheduling. *Proceedings of the First International Conference on MultiAgent Systems*, pages 109–116, 1995. AAAI Press, MIT Press, San Francisco, California.

D. Friedman. *The Double Auction Market Institution: A Survey*, volume XIV, chapter 1, pages 3–25. Addison-Wesley, SFI Studies in the Science of Complexity, the double auction market, eds. d. friedman and j. rust edition, 1993.

R. Hull, A. Kumar, and J. Simeon. Smart supply web: An application of web-based data and workflow mediation. *Workshop on Technologies for E-Services, Hotel Le Meridien Pyramids, Cairo, Egypt*, Sep. 2000.

R. Kalakota. Inter-enterprise fusion: The future of supply chains. *EAI Journal*, May 2000.

P. Klemperer. Auction thoery: A guide to the literature. *Journal of Economic Surveys*, 13(3):227–286, May 1999.

R. Kohout and K. Erol. In-time agent-based vehicle routing with a stochastic improvement heuristic. *Eleventh Conference on Innovative Applications of Artificial Intelligence*, 1999. Orlando, FL.

S. Kraus. Negotiation and cooperation in multi-agent environments. *Artificial Intelligence*, 94(1-2):79–97, 1997.

G. Olsen. An overview of b2b integration. *EAI Journal*, May 2000.

G. Ch. Pflug. *Optimization of Stochastic Models –The Interface Between Simulation and Optimization –*. Kluwer Academic Publishers, 1996. ISBN 0-7923-9780-0.

D. G. Pruitt. *Negotiation Behaviors*. Academic Press, 1981.

V. Rajan, J. R. Slagle, J. Dickhaut, and A. Mukherji. Decentralized problem solving using the double auction market institution. *Expert System with Applications*, 12(1): 1–10, 1997.

T. W. Sandholm. *Negotiation among Self-interested computationally limited agents*. PhD thesis, MIT, 1996.

G. Satapathy. *Distributed and Collaborative Logistics Planning and Replanning Under Uncertainty: A Multiagent based approach*. PhD thesis, Industrial and Manufacturing Engineering Department, PA, December 1999.

J. A. Sauter and H. V. D. Parunak. Ants in the supply chain. *Proceedings of the Workshop on Agent based Decision Support for Managing the InternetEnabled Supply Chain*, May 1999. Seattle.

M. Solomon. Algorithms for vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35:254–265, 1987a.

M.M Solomon. The vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987b.

B.O. Szuprowicz. *Supply Chain Management for E-business Infrastructures*. Computer Technology Research Corp., first edition, 2000.

R. H. Varian. Economic mechanism design for computerised agents. *USENIX Workshop on Electronic Commerce*, July 1995. New York.

W. Walsh and M. Wellman. A market protocol for decentralized task allocation. *In Third International Conference on Multi-Agent Systems*, pages 325–332, 1998.

W. E. Walsh and M. P. Wellman. Modeling supply chain formation in multiagent systems. *IJCAI-99 Workshop on Agent Mediated Electronic Commerce*, 1999.

M. P. Wellman. A general-equilibrium approach to distributed transportation planning. *Proceedings of National Conference on Artificial Intelligence, AAAI-92, San Joe, CA*, pages 282–289, 1992.

M. P. Wellman. A market-oriented programming environment and its application to distributed multicommondity flow problem. *Journal of Artificial Intelligence*, 1:1–23, 1993.

P. R. Wurman, W. E. Walsh, and M. P. Wellman. Flexible double auctions for electronic commerce: Theory and implementation. *Decision Support Systems*, 24:17–27, 1998.

## Chapter 6

# Using Reinforcement Learning to Refine the Distributed Supply Chain Procurement Plans

**Abstract:** In this paper, we propose and develop the method of Coordinated Reinforcement Learning (CRL) to refine the distributed transportation plans generated by a group of agents using Double Auction Market Mechanism (DAMM) in the context of supply chain procurement problem. Through analyzing the drawbacks of the distributed double auction heuristic based on DAMM due to its strict market rules, we enable the task agents in the virtual market to return to their previous stages and reconsider their market trading actions. This is realized in the framework of RL. To be adaptive to the architecture of RL, the task allocation derived from the transportation plans is defined as a *state*; the collective proposals of the task agents are defined as an *action*; the cost saving during one round of market interaction is defined as a *reward*; the update of transportation plans resulting from updating task allocation is defined as a *state transition*. Numerical results demonstrate that CRL with tabular representations effectively refines the distributed transportation plans generated by the DAMM.

**Keywords:** reinforcement learning, supply chain procurement, double auction heuristic, market mechanism, multi-agent systems

## 6.1 Introduction

In a futuristic environment of supply chain procurement, transportation facilities, warehouses and headquarters are distributed but connected through computer networks. Due to the requirements of rapidness and accuracy upon the business processes, distributed task allocation, negotiation and delegation in real time have begun to play a more and more important role [Tang and Kumara, 2004]. With the help of multi-agent systems (MAS), the decision entities in those processes can easily be modeled as software agents. From the perspective of economics, these software agents can play the role of traders and be organized into a virtual market. In the market, these agents exchange their transportation tasks through double auction market mechanism (DAMM), which drive the transportation efficiency to be improved, i.e. the transportation cost to be reduced. The relationships between the systems of transportation planning and market trading are: software agents are modeled as traders; transportation tasks as goods; and cost saving as money or profit. Similar to continuous transactions in the real trading system, the task exchange process repeated many times. This repetitive process is termed as *double auction heuristic* and it updates the commitment relationship virtually to reduce the transportation cost during planning stage.

Reinforcement learning (RL) solves the problem how an autonomous agent that senses the change of its environment after taking actions updates its decisions in order to achieve an optimal goal. This learning method was first applied in game playing [Samuel, 1959], and later much in robot control [Kaelbling et al., 1996], where the states, actions, rewards and state transitions were straightforwardly formulated based on the

movement of the physical entities. Recently, RL, especially combined with neural networks, has been used in the Operations Research (OR) [Bertsekas and Tsitsiklis, 1996]. In these applications, a simulation model that emulates the decision making process of the corresponding optimization problem needs to be developed such that states, actions, rewards and state transitions are defined. When applied in OR, RL is especially good at solving stochastic discrete optimization problems [Secomandi, 2000], which are difficult to be solved with the traditional mathematical programming.

Borrowing the basic structure from the dynamical programming [Mitchell, 1997], RL takes a multi-stage tree-like paradigm for learning and searching. In a double auction heuristic aimed to solve a supply chain procurement problem [Tang and Kumara, 2004], the multi-stage process is a series of continuous iterations based on DAMM. Although DAMM can tremendously reduce the total transportation cost, its potential for cost reduction is not fully explored during one process of the double auction heuristic. If each agent is allowed to turn back to make their decisions again, all the double auction interactions made in multiple processes thus form a multi-stage tree-like paradigm. This paradigm should provide a solid foundation for the agents to make correct decisions toward the reduction of the total cost.

In this paper, we apply RL to refining the distributed transportation plans generated by the DAMM, by connecting the task allocation among the agents with a state, the collective proposals during each round of the DAMM with an action, how to determine the cost saving by the market rule with a reward function, and how to update the task allocation after a transaction with a state transition. As the maximum Q value[1]

[1]Q learning, which is also called model-free RL, is a special kind of RL methods.

in the top hierarchy converges after a number of double auction heuristic processes, the learning process stops at an optimal task allocation, whose total cost cannot be further reduced.

The rest of this paper is arranged as follows. Section 6.2 summarizes the work related to this research. Section 6.3 explains the necessities, possibilities and requirements of introducing RL. Section 6.4, which is the main part, presents the coordinated multi-agent RL that is applied to refine the distributed transportation plans generated from the DAMM. Section 6.5 discusses numerical results. Finally, Section 6.6 summarizes the contribution of the paper and points out some future work.

## 6.2 Related research

This research is related to the areas of heuristic search, distributed simulation, and RL applied in Operations Research (OR). Although the DAMM originates from the negotiation protocol in a MAS, it can be identified as another heuristic search method from the perspective of optimization. Each round of market interaction in the DAMM can be described as a distributed simulation system, which is employed to evaluate the task allocation. At the same time, this simulation model replaces the components of sensing and reacting in the RL. The coordinated RL (CRL) is closely related to the OR applications that are solved with the RL.

For many NP-hard optimization problems, the search space is too huge to be exhaustively searched by enumerating all the possibilities or by always following derivative directions [Rayward-Smith et al., 1996]. Thus, it is widely accepted to develop heuristics to solve these problems. In the heuristic search, the absolute optimality is

replaced by the Pareto optimality with the assumption that many possibilities may not be reached during this incomplete searching process. For the search to be performed at each step, a possible movement that may draw the following state closer to the goal state is conjectured and then evaluated. Based on the evaluation, this conjectured movement may be accepted or rejected with some probability distributions. In the literature [Rayward-Smith et al., 1996], three heuristics – genetic algorithms, simulated annealing, and tabu search – are popular. They are also called meta-heuristics due to their adaptability to many discrete and continuous optimization problems. Usually, only the current state during the search process is remembered by the search process due to its limited memory, and each state has much freedom to move to another state so that the search process can reach as many states as possible. However, the lack of exploitation of the search history may result in visiting the same state repetitively meanwhile little restriction on the movement of the states may easily direct the searching to the wrong direction. In a recent development in exploiting the search history [Laguna, 2002], Artificial Neural Networks (ANN) is applied to remembering the state-value function of the past explorations. A new conjectured movement will be judged by this approximate function before being evaluated. For this reason, the whole search time will be reduced.

A simulation model usually aims to answer "what-if" questions so that user can find the problems of the current system and then propose new solutions to improve the old system [Law et al., 1999]. Recently, researchers and simulation software companies have been attempting to develop the interface between the optimization techniques and the simulations [Fu, 2002] so that the above process can be automatically conducted. A event-driven simulation system is usually implemented in a single computer system

[Banks et al., 2000]. But with the development of the Multi-Agent System, a distributed simulation model can be easily built with the help of a multi-agent platform [Bellifemine et al., 2000; Horling et al., 2003]. Compared to the single computer system, a MAS has several advantages for simulation. First, simulation analysts can match a real system to a distributed system naturally so that the modeling complexity is reduced. Second, since the multi-agent simulation can be deployed in a distributed computer system, the expense of time-consuming simulation can be deducted. Third, since software agents works almost independently with little communications, simulation developers can avoid many details of interacting among the agents.

The theoretic foundation of the delayed RL lies in the Markov Decision Process (MDP) [Sutton et al., 1991]. A MDP is a multi-stage decision process which includes a set of Markov states, their possible actions, their transition probabilities and the rewards associated with all the state-action pairs [Puterman, 1994]. The objective of a MDP is to seek an optimal policy, which is a mapping from the MDP states to actions that maximizes the expected accumulated reward obtained by following a track of MDP states. Each state-action pair is associated with a value that tells the maximum accumulated reward if this pair is chosen. However, it is not a necessity to model the transition probabilities explicitly in the RL. In fact, typical RL algorithms, e.g. SARSA or Q-learning are model free. The iterative updates without the probabilities model prove to converge to the optimal state-action values. More details about RL can be found in [Kaelbling et al., 1996].

A great many applications of RL exist in the literature of machine learning [Mitchell, 1997], especially the adaptive control area [Sutton et al., 1991]. The RL is

especially good at modeling an environment without an explicit model. The RL is also a powerful tool for online learning [Barto et al., 1993] because the knowledge is updated after its physical system has taken actions. On the other hand, almost simultaneously, the RL was applied to solve optimization problems [Bellman, 1956]. Under the name of dynamical programming and its variation neuro-dynamical programming [Bertsekas and Tsitsiklis, 1996], the RL can find the optimal state following the right state-action path starting an initial state. Researchers have successfully solved an elevator scheduling problem [Crites and Barto, 1996], a job-shop scheduling problem [Zhang and Dietterich, 1996], and a channel allocation problem [Singh and Bertsekas, 1996]. The difficult parts in these problems, such as a large state-action space, indeterministic state transition and reward values, dynamic and online optimization, etc., are tackled well through RL.

RL can be combined with a heuristic search method in order to seek global optimization [Miagkikh and III, 1999] for a combinatory optimization problem. In this combination, a vector of free variables is modeled as a state, a heuristic movement as an action, an immediate improvement in the objective value as a reward and the migration to another vector of the variables as a state transition. The history of heuristic search becomes useful, and the optimal and feasible solution can be searched in the RL knowledge-base. Still, compared to the optimization problems with finite possible actions explicitly expressed for each state, to represent this state-action relationship is a challenge.

## 6.3 Learning in double auction heuristic

Although DAMM provides a computationally inexpensive way of allocating transportation tasks in the distributed and dynamic environment [Tang and Kumara, 2004], it can produce low-quality solutions and miss other possibilities to improve the allocation efficiency due to its strict rules of market interactions. To better the solution quality, we introduce a learning model. In the following, the necessities, possibilities and requirements of introducing learning are discussed.

### 6.3.1 Necessities of introducing learning

The DAMM guarantees that the market interactions converge [Tang and Kumara, 2004] at the state when no buying proposals are available to bid the selling proposals. However, the solution converged may not be the global optimal solution because the original problem is intrinsically NP-hard and approximation exists in the heuristic search. Through analyzing the market based heuristic search, we discovered that the non-optimal convergence may result from the following aspects.

- **Stochastic decision rule**

  During the double auction heuristic process, each agent makes decisions based on a set of stochastic rules. No deterministic rules are available due to the lack of necessary information to build a global optimization model. This stochastic process may lead to a direction that will never reach the optimal solution.

- **No prediction**

  Each agent makes decisions solely based on the partial information of the current

stage. However, the correctness of the decisions depends on the accumulative effects over its consecutive stages. A prediction model is needed, but difficult to be developed due to the complexity of the stochastic decision rules and the lack of necessary information. Since the objective value of each agent is the summation of the rewards over all the stages, the model of making decisions based on current stage might make the decision process fail to achieve the optimal goal.

- **No reselection**

  Because the marginal cost of a task will always has the tendency to become increasingly lower during the double auction heuristic process, the market rule does not allow an agent to buy or sell the same task repeatedly. Due to this restriction, the agents are not able to return to a previous stage to change their previous decisions. This might cause the agents deviate from the path toward optimality and never come back.

The design of RL that learns the interactions in the DAMM aims to overcome the drawbacks of stochastic decision rules by means of providing a predictive model and reselection mechanism.

### 6.3.2 Possibilities of introducing learning

Several factors make it possible to incorporate learning into double auction heuristic, and these factors are: the availability of the virtual planning and re-planning system, test-bed of simulation, and a relatively long period of decision time compared.

Firstly, the existing double auction heuristic has the ability of virtual planning and re-planning. At each interactive stage of double auction heuristic, the commitment relationship between a task agent and its tasks is not attached to the physical entities, but subject to adjustment until the repetitive market interactions are stopped. This process of adjusting transportation plans virtually and dynamically can be extended so that learning is achieved because all the plans explored in the learning process are also virtual. However, instead of updating the plans directly in DAMM, the solving process using learning has two phases: the training process and the process of making decisions based on the learnt knowledge.

Secondly, a simulation model for market interactions has been developed so that the training process of a learning model can use it directly. Compared to the learning system that has to sense and react to the physical environment, these two behaviors are easily enabled via software. More remarkably, the DAMM simulator filters out numerous infeasible states and thus saves knowledge storage and the training time significantly.

Finally, decision making has enough time compared to the response of the physical system. There is always a period between the moment that transportation tasks are collected and the moment that physical transportation facilities start to execute them. In the practical scenario of supply chain procurement planning, the task execution can be hours or days later than the time when the customers place their orders. It is also acceptable to respond in minutes or an hour under contingency situation. In the world of computation, minutes, hours and even days are relatively a long time. The long period provides enough time for decision makers to calculate and recalculate. Thus it is possible to accumulate the knowledge of market transactions before the agents make

real decisions to execute tasks. Of course, the high speed of a distributed system is a plus for efficient learning.

### 6.3.3 Requirements on the learning component

In order for the learning component to be integrated with the existing double auction heuristic seamlessly, several requirements must be met. First of all, the efficiency of the task allocations after learning has to be higher than or at least equal to that before learning. At the same time, the anytime characteristic [Tang and Kumara, 2004] of the double auction heuristic algorithm make the transportation planning system flexible. Thus, the learning algorithm needs also to be anytime as well, that is, if the training is stopped at an intermediate stage, the current best transportation plans explored are still useful enough.

## 6.4 Coordinated multi-agent reinforcement learning

CRL is the marriage between machine learning and heuristic search. In this marriage, the double auction reward evaluation in each round corresponds to the simulation in the RL and each step of heuristic search corresponds to the action exploration and exploitation. The RL provides the knowledge repository and the ability to update and to reason about the knowledge. This marriage takes the full usage of the history of the heuristic search.

### 6.4.1 Overview of the DAMM

The details of the DAMM heuristic developed for the supply chain procurement problem can be found in [Tang and Kumara, 2004]. In this section, only a brief version of DAMM is presented. Based on the outline, the interface between the RL and the DAMM heuristic is developed.

A supply chain procurement problem dictated by its transportation cost is modeled and solved with a multi-agent system and the double auction market. In the multi-agent system, transportation vehicles and their transportation companies are defined as task agents, while a virtual broker, who acts as a market specialist, is defined as a monitor agent. Each task agent holds a set of transportation tasks and has the ability to plan these tasks based on their requirements in the way of minimizing the transportation cost. In the double auction market, a task agent emulates a trader and a monitor agent corresponds to a market specialist in a real market (e.g. stock market). The monitor agent implements the double auction market rule based on the collected asks and bids from the task agents.

One round of double auction transaction in DAMM consists of a series of consecutive and distributed computational steps. At the beginning of the transaction, the market is cleared. Then the task agents decide their own trading actions, which can be one of the actions of selling, buying or keeping silent, according to a uniform discrete probability distribution. Later, each selling agent proposes one task to sell (an ask) based on a discrete probability distribution, which is calculated on line according to the relative values of the deletion costs of its tasks to sell. The monitor agent collects

these asks, and then informs the buying agents. With the set of tasks proposed to sell, each buying agent propose one of them to buy (a bid) based on another discrete probability distribution, which is calculated also in line according to the relative difference between their insertion costs and deletion costs. After the monitor agent collects all these bids, finally, the monitor agent implements the double auction market rule to suggest the transactions for the task agents. According to the suggested transactions, the task agents update the commitment relationship with their transportation tasks for the next round of transaction. These interactions are repeated until the convergence criteria are met or the planning time expires.

In DAMM, although the task agents are modeled as individual decision makers, they are not self-interested because they belong to the same company. At some times, they have to sacrifice their own interests for the sake of the whole company. This sacrifice is modeled in the double auction market such that the objective of each individual task agent is in concordance with that of the whole transportation company. For this reason, the Pareto optimality [Sandholm, 1996] is defined as the convergence criteria for the distributed optimization in the DAMM. When the RL is applied to refining its solution, the shared interest among all the task agents is addressed again and this global optimization method is employed to control the heuristic directions. At the same time, RL takes the full usage of DAMM. The process of suggesting transactions in DAMM corresponds to one step of simulation in RL. The decision process of deciding trading actions, asks and bids in DAMM corresponds to the action exploration and exploitation in RL.

A super agent which implements RL is defined based on the goal of the global optimization problem. In fact, this super agent models the collective behaviors of those distributed task agents, and learns the optimal policy in a stochastic transportation network. Let $\mathcal{A}$ denote this super agent, which aims to refine its decisions with the distributed simulation. Its state, action, reward function and state transition are defined as follows.

### 6.4.2 State and action

In a multi-agent system designed for modeling a distributed transportation planning system, we define a set of $n+2$ agents $\mathbf{A} = \{A_0, A_1, ..., A_n, M\}$, where $A_0$ denotes a transportation company, $A_i$ ($i \in [1, n]$) denotes one of the vehicles working for $A_0$ and $M$ a monitor agent required by DAMM. Based on their roles in the virtual market, these agents are classified into two categories: task agents and monitor agents, though different physical facilities in the real system may correspond to the same category[2]. The agents in the set $\mathbf{A}^t = \{A_0, A_1, ..., A_n\}$ are categorized as task agents because each of them holds a set of transportation tasks and is committed to carry out these tasks with a transportation cost. The agent $M$ is categorized as a monitor agent because it monitors and coordinates the double auction market. Only one monitor agent is designed for this application.

The task agent $A_i$ ($i \in [0, n]$) holds a set of transportation tasks $\mathbf{t}_i = \mathbf{t}_i^f \bigcup \mathbf{t}_i^c$, where $\mathbf{t}_i^f$ denotes the set of tasks that are fixed[3] in $A_i$'s routing plan, and $\mathbf{t}_i^c$ denotes the

[2] A transportation company which accepts transportation orders from customers and its transportation vehicles are both modeled as task agents.

[3] These tasks either have been carried out or are being carried on.

set of tasks that are virtually scheduled in $A_i$'s routing plan. The set $\mathbf{t}_i^f$ of fixed tasks is not considered in the double auction heuristic because their commitment relationships with $A_i$ are not subject to change. But the set $\mathbf{t}_i^c$ of scheduled tasks is considered for further allocation. Let $\mathbf{t} = \bigcup_{i=1}^{n} \mathbf{t}_i^c = \{t_1, t_2, ..., t_m\}$ denote the set of virtually scheduled tasks throughout the task agents.

Since the super agent $\mathcal{A}$ models the collective behaviors of all its task agents, the state and action of $\mathcal{A}$ consist of their collective states and actions. Let $s_{\mathcal{A}} = \{\xi_1, \xi_2, ...\xi_m\}$ denote a super state, where $\xi_j (j \in [1, m])$ is the identification of the agent who holds the task $t_j$. $s_{\mathcal{A}}$ memorizes the task allocation among the agents in the set $\mathbf{A}^t$ and can be converted to and from the task set $\mathbf{t}$, which is further decomposed into each task set $\mathbf{t}_i^c$ of each agent $A_i$. On the other hand, the set $\mathbf{t}_i^c$ can be converted to and from its transportation plan. Let $\mathbf{s}_{\mathcal{A}}$ denote the set of all the super states.

The action of $\mathcal{A}$ is the union of the elementary actions of all the task agents. Let $a_{\mathcal{A}} = \bigcup_{i=0}^{n} a_i^e$ denote an action of $\mathcal{A}$, where $a_i^e$ is an elementary action of agent $A_i$. In a double auction market, an elementary action is a tuple including a trading action and a task identification. Let $a^e = (\delta, \rho)$ be an action, where $\delta \in \{sell, buy, silent\}$[4] denotes a trading action, and $\rho$ denotes the identification of the task $t_\rho$[5]. During each round (stage) of interaction, each agent is restricted to submit an ask or bid related to a single task.

[4]With the assumption that the communication cost is zero, a task agent do not need to consider the trading action "silent" during stage of proposing asks and bids. But the agent will learn soon that she should submit a "silent" action if her action brings no reward.

[5]In DAMM, the marginal cost is submitted in a proposal (ask or bid). Since it is calculated based on the interactions with other agents for a reward to be evaluated, it is not explicitly modeled in an action.

The representation of an action also have variations depending on the complexity of the double auction mechanism. For a general case, a task agent can submit $L$ proposals during one round; each proposal has $B$ tasks. Therefore, the action of an agent is denoted by

$$a^e = (< \delta_1, \{\rho_1^1, \cdots, \rho_1^B\} >, \cdots, < \delta_L, \{\rho_L^1, \cdots, \rho_L^B\} >).$$

This is a string with the length of $L \times (1 + B)$. If some tasks or trading actions are not available, their trading actions are filled with *silent* and their task identifications are filled with $-1$.

Each state $s_{\mathcal{A}}$ is associated with a set of actions, which is denoted by $\mathbf{a}_{\mathcal{A}}(s_{\mathcal{A}})$. The union of all these action sets organize the set of all the actions, which is shown in Equation (6.1).

$$\mathbf{a}_{\mathcal{A}} = \bigcup_{\forall s_{\mathcal{A}} \in \mathbf{s}_{\mathcal{A}}} \mathbf{a}_{\mathcal{A}}(s_{\mathcal{A}}) \tag{6.1}$$

### 6.4.3 Reward function and state transition

The reward function and the state transition of the super agent $\mathcal{A}$ are determined algorithmically by the market rules, which are implemented in the monitor agent. The realization of the market rules is a set of matching algorithm to search the transactions such that the total reward based on the proposing agents at current stage is maximized. The complexity of the matching algorithm varies with the complexity of elementary actions. In general, all these matching problems are NP-complete. However, the matching

process can be quick due to the scarcity of the matching graph. The reward value of the super agent $\mathcal{A}$ is the summation of all the individual rewards. The next state is determined by the result of the task exchange after the task agents transact those matched proposals. The reward function and the state transition are summarized in Algorithm 6.1.

**Algorithm 6.1**: The algorithm to determine the reward function and state transition

**for** *each proposed elementary action* $a^e$ **do**
  *Insert* $a^e$ *into a match graph* $G^m$
**end**
*Solve the maximal-benefit matching problem*;
*Modify the non-beneficial actions as silent actions*;
*Send beneficial actions to the corresponding task agents*;
*Wait for execution of the beneficial actions by the task agents*;
*Transfer the allocation of the task allocations to the next state*;

In this paper, we consider a simple double auction market, where at most one task is sold or bought during one round of interaction. After the task agents submit their proposals, the number of the double auction institutions collected by the monitor agent is $I$. Each institution $k \in [1, I]$ is connected with multiple asks and bids. Let $i(k)$ denote the task identification in the institution $k$. The asks are sorted in a descending order. Let $\alpha_g(k)$ denote the ask at the $g^{\text{th}}$ order for the institution $k$ and its corresponding identification of the task agent is $j_g^{\text{S}}(k)$, where $g \in [1, G]$. The bids are sorted in an ascending order. Let $\beta_h(k)$ denote the bid at the $h^{\text{th}}$ order for the institution $k$ and its corresponding identification of the task agent is $j_h^{\text{B}}(k)$, where $h \in [1, H]$. In this double auction institution $I$, the relationships of these asks and bids are restricted in Equations from (6.2) to (6.5), where Equation (6.2) shows the descending asks, Equation (6.3)

the ascending bids, Equation (6.4) how the buying agents bid the selling proposals, and Equation (6.5) how the selling agents bid the buying proposals.

$$\alpha_g(k) \geq \alpha_{g+1}(k)\ \forall k \in [1, I] \text{ and } g \in [1, G] \tag{6.2}$$

$$\beta_h(k) \leq \beta_{h+1}(k)\ \forall k \in [1, I] \text{ and } h \in [1, H] \tag{6.3}$$

$$\beta_H(k) \leq \alpha_1(k)\ \forall k \in [1, I] \tag{6.4}$$

$$\alpha_G(k) \geq \beta_1(k)\ \forall k \in [1, I] \tag{6.5}$$

Thus, the reward, the summation of the market benefits throughout the institutions is calculated in Equation (6.6) and The next state is updated in Equation (6.6).

$$r_{\mathcal{A}} = \sum_{k=1}^{I} (\alpha_1(k) - \beta_1(k)) \tag{6.6}$$

$$\xi_{i(k)} = j_1^{\mathrm{B}}(k)\ \forall k \in [1, I] \tag{6.7}$$

Generally, the reward function and can be expressed in Equation (6.8), where $r_{\mathcal{A}}$ denote the round reward, and its corresponding state transition can be expressed in Equation (6.9).

$$r_{\mathcal{A}} = r_{\mathcal{A}}(s_{\mathcal{A}}, a_{\mathcal{A}}^{u}) \tag{6.8}$$

$$s'_{\mathcal{A}} = f_{\mathcal{A}}(s_{\mathcal{A}}, a^u_{\mathcal{A}}) \tag{6.9}$$

### 6.4.4 Updating Q Values

With the states of the super agent connected by those actions, which are associated with rewards, a RL tree is formed. At each state, the optimal policy maximize the summation of the rewards along a series of states and actions, starting from this state and ending with a convergence state. When a Q value, what is to be learned in the Q-learning, reformulates this maximization problem recursively, the learning process will be simplified based on the update of Q values.

Starting from the state $s_{\mathcal{A}}$ and the action $a_{\mathcal{A}}$, the reward summation at the stage $\tau$ under the policy $\pi$ can be expressed in Equation (6.10), where $\tau + n_e$ is the ending stage. Usually, there exists a discounted factor (less than 1) in calculating the reward summation, and this factor enables the RL to deal with the problem with infinite state transitions in a series. However, it is omitted, for finite states are considered in this paper.

$$\begin{aligned} V^{\tau}_{\pi}(s_{\mathcal{A}}, a_{\mathcal{A}}) &= r^{\tau}_{\mathcal{A}} + r^{\tau+1}_{\mathcal{A}} + \cdots + r^{\tau+n_e}_{\mathcal{A}} \\ &= \sum_{i=0}^{n_e} r^{\tau+i}_{\mathcal{A}}, \ \forall s_{\mathcal{A}} \in \mathbf{s}_{\mathcal{A}}, a_{\mathcal{A}} \in \mathbf{a}_{\mathcal{A}}(s_{\mathcal{A}}) \end{aligned} \tag{6.10}$$

The optimization problem is expressed in Equation (6.11), where $s'_{\mathcal{A}}$ is the result of state transition of $(s_{\mathcal{A}}, a_{\mathcal{A}})$, and $a'_{\mathcal{A}}$ is one of the actions associated with the state $s'_{\mathcal{A}}$.

$$\begin{aligned}\pi^*(s_{\mathcal{A}},\tau) &= arg \max_{a_{\mathcal{A}}\in \mathbf{a}_{\mathcal{A}}(s_{\mathcal{A}})} V_{\pi}^{\tau}(s_{\mathcal{A}},a_{\mathcal{A}}),\ \forall s_{\mathcal{A}}\in \mathbf{s}_{\mathcal{A}} \\ &= arg \max_{a_{\mathcal{A}}\in \mathbf{a}_{\mathcal{A}}(s_{\mathcal{A}})} \{r_{\mathcal{A}}^{\tau}(s_{\mathcal{A}},a_{\mathcal{A}}) + \max_{a'_{\mathcal{A}}\in \mathbf{a}_{\mathcal{A}}(s'_{\mathcal{A}})} V_{\pi^*}^{\tau+1}(s'_{\mathcal{A}},a'_{\mathcal{A}})\}\end{aligned} \tag{6.11}$$

Let $q^{\tau}(s_{\mathcal{A}},a_{\mathcal{A}})$ in Equation (6.12) define a Q value.

$$q^{\tau}(s_{\mathcal{A}},a_{\mathcal{A}}) = r_{\mathcal{A}}^{\tau}(s_{\mathcal{A}},a_{\mathcal{A}}) + \max_{a'_{\mathcal{A}}\in \mathbf{a}_{\mathcal{A}}(s'_{\mathcal{A}})} V_{\pi^*}^{\tau+1}(s'_{\mathcal{A}},a'_{\mathcal{A}}) \tag{6.12}$$

Then a Q value can be defined recursively in Equation (6.13).

$$q^{\tau}(s_{\mathcal{A}},a_{\mathcal{A}}) = r_{\mathcal{A}}^{\tau}(s_{\mathcal{A}},a_{\mathcal{A}}) + \max_{a'_{\mathcal{A}}\in \mathbf{a}_{\mathcal{A}}(s'_{\mathcal{A}})} q^{\tau+1}(s'_{\mathcal{A}},a'_{\mathcal{A}}) \tag{6.13}$$

The optimization problem can be expressed in Equation (6.14).

$$\pi^*(s_{\mathcal{A}},\tau) = arg \max_{a_{\mathcal{A}}\in \mathbf{a}_{\mathcal{A}}(s_{\mathcal{A}})} q^{\tau}(s_{\mathcal{A}},a_{\mathcal{A}}) \tag{6.14}$$

In this paper, the updates of Q values are based on Equation (6.15). This update form is also known as a *full backup* [Kaelbling et al., 1996], which uses up the information from all the successor states. Usually, the updates in the Q learning take the form in Equation (6.16) [Singh, 1993] to deal with a stochastic environment to learn a free model. Since the learning rate $\alpha$ decreases slowly, the learning agent has enough opportunities to explore the new actions during the early phase of learning. However, when applied to refining the DAMM, the Q learning is dealing with a deterministic environment. Furthermore, the action exploration are considered by the DAMM heuristic. Therefore,

taking the form of full backups does not only make no influence on the process of action exploration, a key component of the RL, but also this type of form speeds up the learning process.

$$q^{\tau}(s_{\mathcal{A}}, a_{\mathcal{A}}) \leftarrow r^{\tau}_{\mathcal{A}}(s_{\mathcal{A}}, a_{\mathcal{A}}) + \max_{a'_{\mathcal{A}} \in \mathbf{a}_{\mathcal{A}}(s'_{\mathcal{A}})} q^{\tau+1}(s'_{\mathcal{A}}, a'_{\mathcal{A}}) \tag{6.15}$$

$$\begin{aligned} q^{\tau}(s_{\mathcal{A}}, a_{\mathcal{A}}) \quad \leftarrow \quad & (1-\alpha) q^{\tau}(s_{\mathcal{A}}, a_{\mathcal{A}}) \\ & + \alpha \{ r^{\tau}_{\mathcal{A}}(s_{\mathcal{A}}, a_{\mathcal{A}}) + \max_{a'_{\mathcal{A}} \in \mathbf{a}_{\mathcal{A}}(s'_{\mathcal{A}})} q^{\tau+1}(s'_{\mathcal{A}}, a'_{\mathcal{A}}) \} \end{aligned} \tag{6.16}$$

### 6.4.5 Coordinated learning

The DAMM provides an efficient way to provide the state transition and reward evaluation in a distributed multi-agent simulation system; the RL learns to refine the decisions of these task agents in a global manner. In this section, it is presented how the RL incorporates the distributed simulation.

Currently, the Q values are explicitly stored in a table without utilizing any approximation method. In fact, this Q table does not necessarily store all the Q values. Instead, only those Q values whose states and actions are explored and produce positive rewards are stored. This saves a large amount of memory.

In an ordinary RL, action exploration and exploitation are interwoven to provide information to update the Q values based on the probability distribution calculated according to these Q values relatively. In our coordinated learning architecture, these

two tasks are carried out by the double auction market mechanism. This strategy of action exploration and exploitation has high efficiency because it considers the context of the problem.

The RL is usually claimed as a delayed update because the simulation in one step only updates the Q value for one state transition and never updates the upstream states and actions. This design is mainly made for the approximation architecture. Since we are using an explicit tabular architecture, the upstream states and actions can be updated explicitly to accelerate the process of updating Q values. Of course, the delayed update can also be employed.

Generally, the RL is categorized into TD($\lambda$) (Temporary Difference) methods, that is, the simulation for one step of learning continues for $\lambda$ steps and the update takes the advantage of these $\lambda$ steps. In our coordinated learning, the method of TD($\infty$), where the simulation for one state continues until an ending state, is applied for updating. When this learning strategy is combined with the explicit upstream update, each simulation will update the Q values starting from the initial state, that is, the existing task allocation to be improved.

Let $s_{\mathcal{A}}$ be the chosen state for the current step of learning. Starting from the state $s_{\mathcal{A}}$, the track to an ending state is expressed in Equation (6.17), where $e$ is number of state transitions.

$$s_{\mathcal{A}}(a_{\mathcal{A}}) \rightarrow s^1_{\mathcal{A}}(a^1_{\mathcal{A}}) \rightarrow s^2_{\mathcal{A}}(a^2_{\mathcal{A}}) \rightarrow \cdots s^{e-1}_{\mathcal{A}}(a^{e-1}_{\mathcal{A}}) \rightarrow s^e_{\mathcal{A}} \tag{6.17}$$

Let $s_o$ denote the starting state and the track from $s_o$ to $s$ is expressed in Equation (6.18). All the information in this track has been simulated and stored in memory.

$$s_{\mathcal{A}}^{o}(a_{\mathcal{A}}^{o}) \rightarrow s_{\mathcal{A}}^{o+1}(a_{\mathcal{A}}^{o+1}) \rightarrow \cdots s_{\mathcal{A}}^{-1}(a_{\mathcal{A}}^{-1}) \rightarrow s_{\mathcal{A}} \tag{6.18}$$

The updating process can be expressed in Equation (6.19). It first utilizes the fresh simulation results. This update starts from the state-action pair $s_{\mathcal{A}}^{e-1}, a_{\mathcal{A}}^{e-1}$, whose Q value is assigned with the reward of this state-action pair. The upstream Q values are updated with the summation of a corresponding new reward and the maximum Q value starting from the next state. This update based on the fresh simulation results continues until the chose state for this step of learning is reached. Then this process continues to update Q values based on the existing state transitions, until the initial state is reached. Such strategy of updating makes convenient the detection of convergence condition for learning, which just considers the maximum Q values originated from the initial state.

$$
\begin{aligned}
q(s_{\mathcal{A}}^{e-1}, a_{\mathcal{A}}^{e-1}) &= r_{\mathcal{A}}(s_{\mathcal{A}}^{e-1}, a_{\mathcal{A}}^{e-1}) \\
q(s_{\mathcal{A}}^{e-2}, a_{\mathcal{A}}^{e-2}) &= r_{\mathcal{A}}(s_{\mathcal{A}}^{e-2}, a_{\mathcal{A}}^{e-2}) + \max_{a_{\mathcal{A}}^{e-1} \in \mathbf{a}_{\mathcal{A}}(s_{\mathcal{A}}^{e-1})} q(s_{\mathcal{A}}^{e-1}, a_{\mathcal{A}}^{e-1}) \\
&\vdots \\
q(s_{\mathcal{A}}, a_{\mathcal{A}}) &= r_{\mathcal{A}}(s_{\mathcal{A}}, a_{\mathcal{A}}) + \max_{a_{\mathcal{A}}^{1} \in \mathbf{a}_{\mathcal{A}}(s_{\mathcal{A}}^{1})} q(s_{\mathcal{A}}^{1}, a_{\mathcal{A}}^{1}) \\
q(s_{\mathcal{A}}^{-1}, a_{\mathcal{A}}^{-1}) &= r_{\mathcal{A}}(s_{\mathcal{A}}^{-1}, a_{\mathcal{A}}^{-1}) + \max_{a_{\mathcal{A}} \in \mathbf{a}_{\mathcal{A}}(s_{\mathcal{A}})} q(s_{\mathcal{A}}, a_{\mathcal{A}}) \\
&\vdots \\
q(s_{\mathcal{A}}^{o}, a_{\mathcal{A}}^{o}) &= r_{\mathcal{A}}(s_{\mathcal{A}}^{o}, a_{\mathcal{A}}^{o}) + \max_{a_{\mathcal{A}}^{o+1} \in \mathbf{a}_{\mathcal{A}}(s_{\mathcal{A}}^{o+1})} q(s_{\mathcal{A}}^{o+1}, a_{\mathcal{A}}^{o+1})
\end{aligned}
\tag{6.19}
$$

Algorithm 6.2 summarizes the above learning process, which consists of two phrases: simulation and updating. In the simulation phrase, the state transitions are explored until a ending state is reached. In the updating phrase, the Q values starting from this ending state are updated until the initial state is reached. These two phrases are wrapped in one learning step and the whole learning algorithm repeats this learning step until the stopping criteria is stratified.

The simulation phrase starts from an arbitrarily chosen state, which is transformed into the transportation plans among a set of task agents. Then the DAMM is invoked and the market interactions occurs based on the Algorithm 6.1. During each interaction, the reward and state transition results are detected and registered in the Q table. If a convergence state is reached based on the convergence criteria in the DAMM, the simulation phrase will stop and the next phrase, the updating phrase begins.

**Algorithm 6.2**: The coordinated reinforcement learning algorithm to refine the distributed double auction market mechanism

*Initialize the starting state from the initial allocations*;
**while** *stopping criteria of learning not satisfied* **do**
  *Select a state* $s_{\mathcal{A}}$ *arbitrarily*;
  **while** $s_{\mathcal{A}}$ *is not an ending state* **do**
    *Transfer state* $s_{\mathcal{A}}$ *to the transportation plans among the task agents*;
    *Find the reward* $r_{\mathcal{A}}$*, and next state* $s'_{\mathcal{A}}$ *by calling Algorithm 6.1*;
    **if** $r_{\mathcal{A}} > 0$ **then**
      *Adding* $s'_{\mathcal{A}}$ *to the state table*;
      $s_{\mathcal{A}} \leftarrow s'_{\mathcal{A}}$;
    **end**
  **end**
  **while** $s_{\mathcal{A}}$ *is not the initial state* **do**
    *Update* $q(s_{\mathcal{A}}^{-1}, a_{\mathcal{A}}^{-1}) \leftarrow r_{\mathcal{A}}^{-1} + \max_{a_{\mathcal{A}} \in \mathbf{a}_{\mathcal{A}}(s_{\mathcal{A}})} q(s_{\mathcal{A}}, a_{\mathcal{A}})$;
    $s_{\mathcal{A}} \leftarrow s_{\mathcal{A}}^{-1}$;
  **end**
**end**

Compared to the simulation phrase, the updating phrase visit the trace of states reversely. Starting from an ending state, all its upstream state-action pairs are updated according to Equation (6.19). The reason why this update process works lies in the explicit Q tables, which memorize the precedent state-action pairs of all the states. It can be proved that the Q value of any state-action pair will be monotonically increasing because of the maximization operator in each Q update equation.

The stopping criteria of learning is based on two conditions. The first one is the maximum number of times of learning. The second one is based on the convergence property of the maximum Q value starting from the initial state. Similar to the design of the convergence condition of the DAMM, this convergence condition also prescribe a strong signal, i.e., that maximum Q value needs to be stable for multiple consecutive learning steps.

Let $q^M(l)$ in Equation (6.20) denote the maximum Q value starting from the initial state $s^o_{\mathcal{A}}$ for $l$ times of learning.

$$q^M(l) = \max_{a^o_{\mathcal{A}} \in \mathbf{a}_{\mathcal{A}}(s^o_{\mathcal{A}})} q(s^o_{\mathcal{A}}, a^o_{\mathcal{A}})|l \tag{6.20}$$

Let $\epsilon$ denote the convergence precision allowed, then the convergence condition for $l$ times of learning is expressed in Equation (6.21), where $q^M(0)$ is defined as an arbitrary small number.

$$\frac{|q^M(l) - q^M(l-1)|}{q^M(l-1)} \leq \epsilon \tag{6.21}$$

Let $c$ denote the times of learning, where the value of $q^M(l)$ continuously converge. If the convergence condition in Equation (6.21) is satisfied, $c$ is incremented. Otherwise, $c$ is set to zero and the counting process starts over. Let $c^m$ denote the required minimum times of converging. If $c \geq c^m$ is checked, the learning process will terminate and the process of searching decisions will begin.

### 6.4.6 Searching decisions

After the Q learning converges, the Q values associated with all the state-action pairs provide the necessary knowledge for finding the best task allocation. A recursive searching process carries on this task. A general case, how to find the best task allocation starting from an arbitrary state, is considered at first. Its recursive procedures are shown in Algorithm 6.3, which is explained in the following. If the state $s_{\mathcal{A}}$ does not have any successor actions, the search process will be terminated and the current state is returned

as the best allocation. Otherwise, one of its successor actions with the maximum Q value is found by enumerating its action space. With this action, the current state is transited to another state $s'_{\mathcal{A}}$. Then the state $s'_{\mathcal{A}}$ is considered as another starting state and the above procedures are repeated. Based on this algorithm, the recursive calling will not stop until an ending state is reached. Since the whole searching process is based on enumerating the Q values at each state and the Q values have global meanings after the RL, this ending state is the best solution derived from the learned knowledge. If the initial state $s^o_{\mathcal{A}}$ is given as the starting point, this searching process can find the best global solution.

**Algorithm 6.3**: The search algorithm based on the learned Q values: searchDecision(State $s_{\mathcal{A}}$)

*Collect all the possible actions starting from* $s_{\mathcal{A}}$;
**if** *the collection is empty* **then**
  *Return* $s_{\mathcal{A}}$ *as the current best state*;
**end**
**else**
  *Find the action* $a^M_{\mathcal{A}}$ *with the maximum Q value from the collection*;
  *Move the state from* $s_{\mathcal{A}}$ *to* $s'_{\mathcal{A}}$ *along the action* $a^M_{\mathcal{A}}$;
  *Call searchDecision(*$s'_{\mathcal{A}}$*)*;
**end**

## 6.5 Numerical results

The coordinated RL algorithms are implemented in Java, together with DAMM, which is implemented in a virtual distributed system where communication between agents are implemented as message passing between Java objects. All the computations

run in a PC with 512M RAM, Pentium 4 CPU, and Windows XP OS. Under these experimental configurations, a set of sample problems are solved with coordinated RL.

We use the same sample set as in Tang and Kumara [2004] to test the method of coordinated RL. The sample set consists of a number of vehicle routing problems with time windows in stochastic environments; these problems are generated with reasonable randomization. For a sample problem, we compare the solutions generated with DAMM and the coordinated RL.

We compare the results of the methodologies of the DAMM and coordinated RL in various ways. First of all, the comparisons for a problem with 210 (medium size in the samples) transportation tasks are presented. We compare its task allocations with graphical illustrations, its evolutionary histories with curves, and its sensitivities to task sequences. Furthermore, we apply these two methods to the sample problems of different sizes.

### 6.5.1 The problem with 210 tasks

In this problem, 100 vehicles with the homogenous capacity of 200 units of volume are available for 210 transportation tasks. Fig. 6.1 shows the distribution of these tasks in an artificial geographical area. The center dot with coordinates $(100, 100)$ serves as the depot, at which each vehicle starts and ends its routine. Other dots are the pickup or delivery locations given by the customers. A transportation task is represented by an arrow, whose tail designates a pickup location and whose head designates a delivery location. If a dot is not connected with an arrow, the pickup and delivery location in

the same task coincides. Time windows are not displayed explicitly, but each task node is attached with a time window.

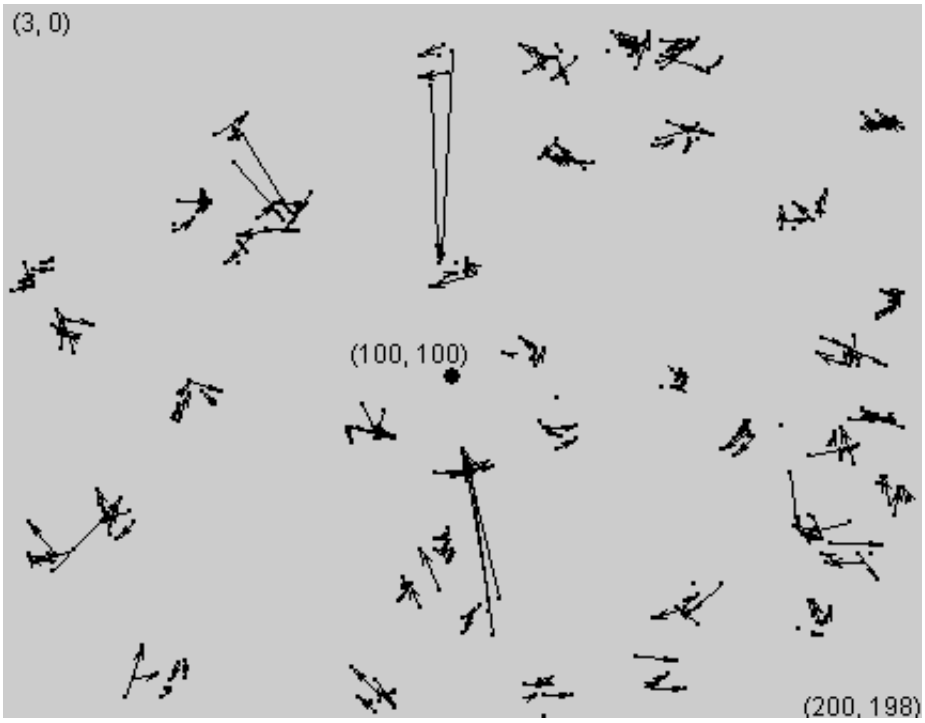


Fig. 6.1. The graphical distribution of 210 tasks in a vehicle routing problem

#### 6.5.1.1 Routines comparison

Fig. 6.2 and 6.3 draws the routines generated by DAMM and coordinated RL respectively for a problem with 210 transportation tasks. Both solutions consist multiple routines, corresponding to each vehicle. Each routine starts from the depot, visits series of task nodes sequentially, and finally return to the depot. In the figures, the travel between any two adjacent task nodes is represented by an arrow. We observes that the

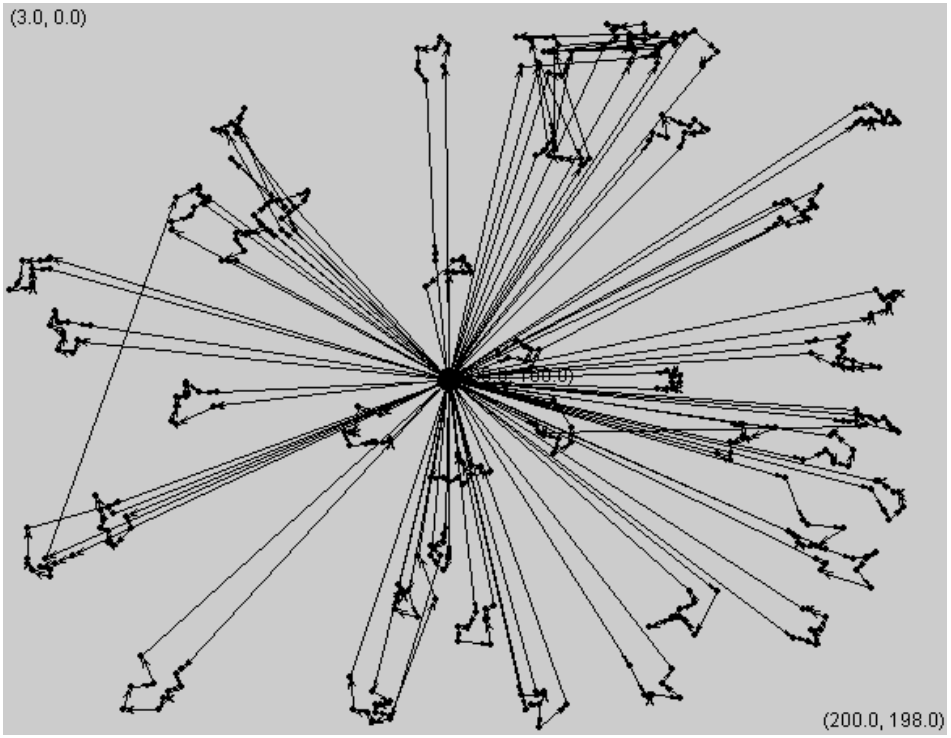


Fig. 6.2. The task allocation on the problem of 210 tasks using double auction heuristic with the transportation cost of 88952.0 units and 47 vehicles

routines in Fig. 6.2 is not so well clustered as those routines in Fig. 6.3. Also we notice that the allocation with DAMM takes more vehicles to finish all the tasks than that coordinated RL does.

#### 6.5.1.2 Evolutionary histories comparison

Fig. 6.4 shows different paths of cost reduction, as the number of market interactions increases. We observe that the path after the coordinated RL takes less iterations to converge than that with DAMM, and at the same time, that the former path reaches at a lower level that the later one. We notice that the coordinated RL makes the double auction market smart to choose an effective path of market transitions.

#### 6.5.1.3 Sequence sensitiveness comparison

In this experiment, we test how different sequences of the transportation tasks affect the coordinated RL and DAMM respectively. Since market transitions are conditional with market rules, the existing commitment relationships of the tasks may pose different market evolutionary scenarios. For this reason, the final cost for the same set of tasks may vary with their different sequences. The curves in Fig. 6.5 and 6.6 show how the sequence affects these two methods for the case of 53 and 210 tasks respectively. In Fig. 6.5, we observed that the final costs for all 35 different sequences after the coordinated RL converge to the optimal value, and thus no variance exists in this curve. In Fig. 6.6, we observed that the curve for the coordinated RL is much flatter than the one for DAMM. Table 6.1 shows the standard deviations of these four curves.

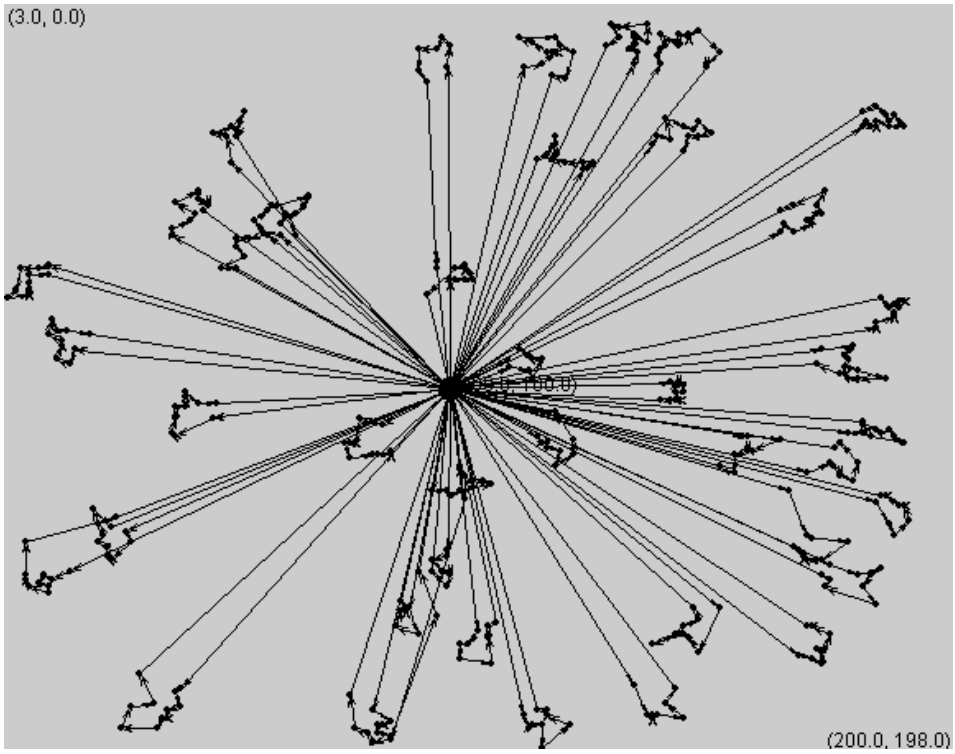


Fig. 6.3. The task allocation on the problem of 210 tasks using coordinated reinforcement learning in DAMM with the transportation cost of 73310.1 units and 41 vehicles

Table 6.1. Comparison of standard deviations between coordinated RL (CRL) and DAMM for 53 and 210 tasks respectively

| # of tasks | DAMM STD | CRL STD |
|---|---|---|
| 53 | 628.5 | 0.0 |
| 210 | 4038.1 | 1809.1 |

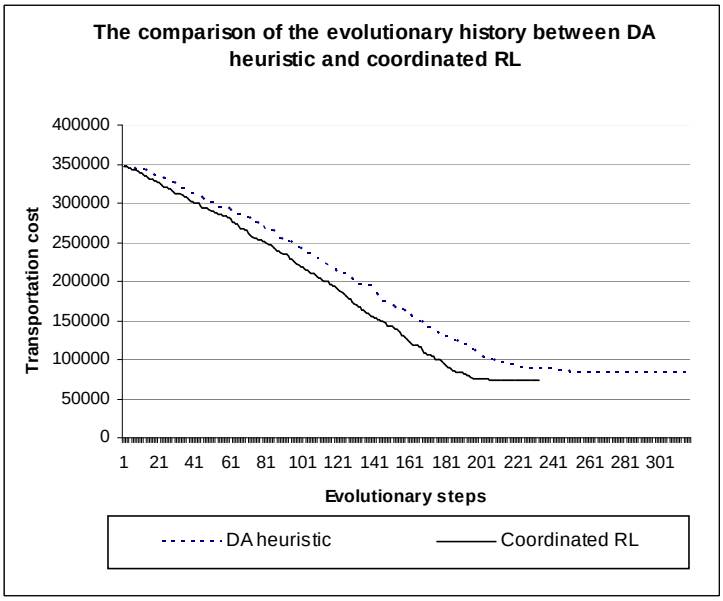


Fig. 6.4. The comparison of the evolutionary histories between DA heuristic and coordinated RL

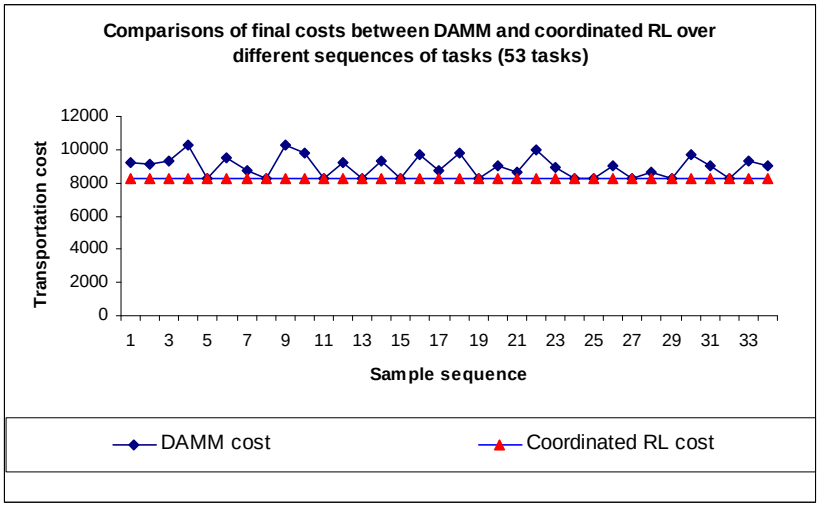


Fig. 6.5. The comparison of the final costs between DAMM and coordinated RL over different sequences of 53 tasks

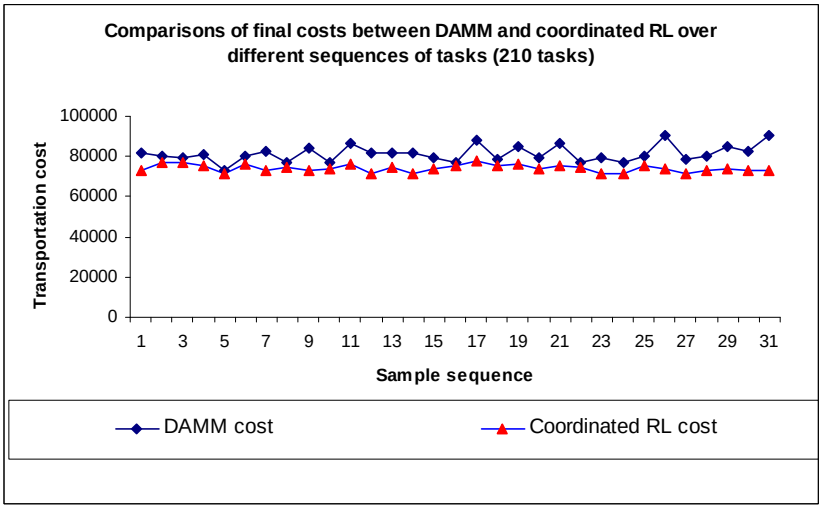


Fig. 6.6. The comparison of the final costs between DAMM and coordinated RL over different sequences of 210 tasks

We compare the coordinated RL with DAMM when they are applied to various problems with different numbers of tasks. Table 6.2 shows these comparisons and the percentage improved by coordinated RL over DAMM. We observed that the costs for all the cases are reduced to different extents, and especially a large amount of cost is reduced for the cases of 53, 105 and 210 tasks.

### 6.5.2 Discussion

Based on the comparisons between the coordinated RL and DAMM, we found that the coordinated RL continues to reduce the cost after the double auction heuristic converges. This does not only make the smart market takes a shortcut to the converged solution, but also the solutions generated by the coordinated RL is not so sensitive to the sequences of tasks as those generated by DAMM. This is because the coordinated RL directs the solution to a lower level and narrows the gap between the heuristic

Table 6.2. Comparison of cost reduction between coordinated RL (CRL) and DAMM for various sample problems

| # of tasks | DAMM | | CRL | | Reduced By (%) |
|---|---|---|---|---|---|
| | Cost | Vehicles | Cost | Vehicles | |
| 53 | 10271.4 | 13 | 8289.4 | 10 | 19.30 |
| 105 | 33307.1 | 24 | 27045.7 | 20 | 18.80 |
| 210 | 90664.4 | 49 | 72933.2 | 41 | 19.56 |
| 314 | 161840.4 | 68 | 145434.2 | 62 | 10.14 |
| 419 | 296187.6 | 92 | 271948.2 | 84 | 8.08 |
| 526 | 490192.4 | 115 | 447559.9 | 105 | 8.70 |

solution and the optimal solution. These improvements are due to the data structure and updating mechanism of the RL.

The tree-like data structure of RL tracks possible heuristic traits and enables the task agents to reconsider their trading choices made previously; the updating mechanism refines the Q value of each arc so that it can be improved, as more market interactions are transacted. Fig. 6.7 shows this situation. Let the trait from $s_{\mathcal{A}}^0$ to $s_{\mathcal{A}}^1$ ending with $s_{\mathcal{A}}^2$ is a path of the double auction heuristic, and the trait from $s_{\mathcal{A}}^0$ to $s_{\mathcal{A}}^1$ ending with $s_{\mathcal{A}}^4$ another path. This situation could be ascribed to the fact that RL chooses $s_{\mathcal{A}}^1$ as one of its starting state for learning. If $q_1^1 > q_1^0$ is true, the second trait is chosen as a better solution and the searching process will automatically enable it.

From another aspect of view, the double auction heuristic is a special case of the RL and the RL is the extension of the double auction heuristic. In the learning algorithm, the double auction heuristic is repeatedly started at every possible state until the ending state is met. Every repetition is a double auction heuristic. The decisions are refined after more valuable samples are collected.

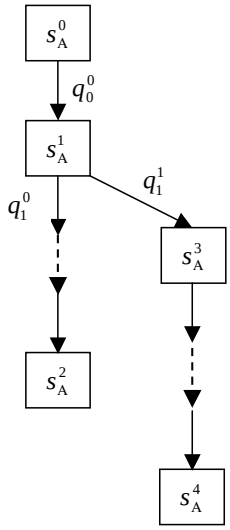


Fig. 6.7. A tree-like memory structure in the reinforcement learning

## 6.6 Conclusion

In this paper, we introduce the coordinated RL to refine the task allocation generated by the double auction heuristic with the help of a distributed MAS. Numerical comparisons demonstrated that the coordinated RL can continuously reduce the cost after the double auction heuristic, given a durable period of time. For this reason, the double auction market with the coordinated RL performs overall better than the one without it. The development was considered as a marked improvement over the double auction heuristic and contribution to the distributed task allocation in a supply chain procurement scenario.

This research can be extended in several aspects. Firstly, only simple elementary trading actions are designed for the double auction heuristic in this paper, and we may try complicated elementary actions to improve the converging speed. Secondly, RL

in this paper selects a state with a uniform distribution over the explored states, and we may develop alternative deterministic or stochastic strategies to improve the learning efficiency. Thirdly, we need to develop approximation methods, for example using neural network, to solve even larger problems. We may extend the marriage between the market based heuristic and RL as a new heuristic to solve generic discrete optimization problems.

## Bibliography

J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol. *Discrete-Event System Simulation (3rd Edition)*. Prentice Hall, 2000. ISBN 0-130-88702-1.

A. G. Barto, S. J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. Technical Report UM-CS-1993-002, UM, 1993.

F. Bellifemine, A. Poggi, G. Rimassa, and Turci P. An object oriented framework to realize agent systems. *Proceedings of WOA 2000 Workshop*, pages 52–57, May 2000.

R. Bellman. A problem in sequential design of experiments. *Sakhuya*, 16:221–229, 1956.

P. D. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996. ISBN 1-886529-10-8.

R. Crites and A. Barto. Improving elevator performance using reinforcement learning. *Advances in Neural Information Processing Systems: Proc. of the 1999 Conf.*, pages 1017–1023, 1996.

M. C. Fu. Optimization for simulation: Theory vs. practice. *INFORMS Journal on Computing*, 14(3):192–215, 2002.

B. Horling, R. Mailler, and V. Lesser. Farm: A scalable environment for multi-agent development and evaluation. *Proceedings of the 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS 2003)*, May 2003.

L. P. Kaelbling, M. L. Littman, and A. P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

R. Laguna, M.and Marti. Neural network prediction in a system for optimizing simulations. *IIE Transactions*, 34:273–282, 2002.

A. M. Law, D. W. Kelton, W. D. Kelton, and D. M. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill Science/Engineering/Math, 1999. ISBN 0-070-59292-6.

Victor V. Miagkikh and William F. Punch III. Global search in combinatorial optimization using reinforcement learning algorithms. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 189–196, Mayflower Hotel, Washington D.C., USA, 6-9 1999. IEEE Press. ISBN 0-7803-5537-7 (Microfiche).

T.M. Mitchell. *Machine Learning*. McGra-Hill Companies, 1997.

M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc, 1994. ISBN 0-471-61977-9.

V. J. Rayward-Smith, I. H. Osman, C. R. Reeves, and G. D. Smith, editors. *Modern heuristic search methods*. John Wiley & Son Ltd, 1996. ISBN 0-471-96280-5.

A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3:211–229, 1959.

T. W. Sandholm. *Negotiation among Self-interested computationally limited agents*. PhD thesis, MIT, 1996.

N. Secomandi. Comparing neuro-dynamic programming algorithms for the vehicle routing problem with stochastic demands. *Computers and Operations Research*, 27(11): 1201–1225, Elsevier Science Ltd. 2000.

S. Singh and D. Bertsekas. Reinforcement learning for dynamic channel allocation in cellular telephone systems. *In Proc. of Advances in Neural Information Processing Systems*, pages 974–980, 1996.

S. P. Singh. *Learning to Solve Markovian Decision Processes*. PhD thesis, Department of Computer Science, University of Massachusetts, 1993. Also, CMPSCI Technical Report 93-77.

R. S. Sutton, A. G. Barto, and R. J. Williams. Reinforcement learning is direct adaptive optimal control. In *Proceedings of the 1991 American Control Conference*, 1991.

K. Tang and S. Kumara. Double auction market mechanism: A distributed negotiation protocol to model an e-procurement problem. *IEEE Automation Science and Engineering*, 2004. accepted.

W. Zhang and T. Dietterich. High performance job-shop scheduling with a time-delay td(l) network. *In Proc. of Advances in Neural Information Processing Systems*, pages 1024–1030, 1996.

## Chapter 7

# Meta-heuristic Enabled Multi-Agent System Optimization

**Abstract:** This paper introduces a meta-heuristic enabled multi-agent optimization architecture for dynamic transportation planning in the context of supply chain procurement planning. When multi-agent systems (MAS) are used for real-time dynamic optimization, agents seek the solution using distributed heuristics. However, distributed heuristics based on local information are prone to converge at local optimality. To escape from local optimality toward higher quality solution, we introduce meta-heuristics over agent interactions to advise agents' searching process. In this paper, we mainly propose variable neighborhood search meta-heuristic (VNS-MH) over distributed market based heuristic (DMBH), a distributed heuristic based on market interactions for transportation planning. The numerical results shows that VNS-MH does not only performs better on achieving optimality than DMBH, but also requires less memory and computational time than its predecessor – coordinated reinforcement learning (CRL).

**Keywords:** multi-agent Systems, meta-heuristic, variable neighborhood search, dynamic optimization, distributed heuristic

## 7.1 Introduction

Traditionally, transportation planning problems are solved using optimization methodologies such as mathematical programming or heuristic algorithms in a static and periodical manner. At the beginning of each period, an optimization problem is formulated for the purpose of planning the arrival orders. At the same time, this problem is solved with some optimization procedure, either a mathematical programming package or some heuristic algorithm. Then the solution of this problem is applied to the operational world for execution. The solution is not subject to change until the beginning of next period. This problem solving strategy is static and has two main drawbacks. First, the failure to consider new arrivals during execution periods may waste resources, thus degrade overall performance. At the same time, the ignorance of contingency conditions during execution periods may also worsen the overall performance.

As additional computing power, real-time data, and the ability to instantly communicate are becoming widely available with the advances in Information Technology (IT), it is more valuable to build the model of dynamic optimization to achieve overall better performance than the current situation with only static optimization [Tang et al., 2004]. In dynamic optimization architecture, Multi-Agent Systems (MAS) have been proposed as a promising approach to model problems in the distributed, complex, heterogeneous and dynamic domains [Weiss, 1999; Wooldridge, 2002]. To solve optimization problems, the agents in the same community cooperatively search the optimal solution through interactive protocols using distributed heuristics [Tang and Kumara, 2004a; Lee, 2002]. The optimal solution is later applied to operational execution. MAS seems to

demonstrate the advantages of integrating monitoring, controlling and optimization in a seamless manner so that the instantly available information is exploited for performance improvement.

MAS optimization is reported to be limited upon finding optimal solutions due to its strict and self-interested heuristic rules Miagkikh and Punch [1999]. During each step of a distributed heuristic, a neighboring state is formed and evaluated based on each agent's myopic view. Computationally, this heuristic can be very fast because each agent solves its own problem with reduced scale and less coupling, compared to the original optimization problem. However, rational and information limited agents may easily stop searching at local optimality, which cannot be further improved without new searching strategies.

The time gap between the readiness of planning solutions and the subsequent physical operations enable deeper search for better solution. Due to the high speed of computation and communication of MAS, the agents are usually idle before next iteration of distributed search. For an agent-based computer system, this time gap is usually a long period time, which ranges from several minutes to several days or even longer. The agent system may explore more alternatives to improve the solution quality. Observing the limitation of MAS optimization and time gap for further exploring alternatives, we ask this question: is it possible to utilize the time gap to improve the local optimal solution obtained from a distributed heuristic?

In optimization literature, the concept of meta-heuristics has been widely used to seek global optimal solutions [Blum and Roli, 2001]. One of the main branches of meta-heuristics, genetic algorithm has been applied in dynamic optimization [Branke, 2001].

In this paper, we propose and develop a meta-heuristics enabled MAS optimization architecture to exploit real-time information and the time gap between operations to improve overall performance. Specifically, we introduce and discuss Variable Neighborhood Search Meta-Heuristic (VNS-MH) over Distributed Market Based Heuristic (DMBH) for transportation planning.

The paper is organized as follows. Section 7.2 discusses literature related to this paper. Section 7.3 summarizes our previous work on DMBH, and discusses the weakness of DMBH and how it can be controlled by a higher level controller. Section 7.4 presents the proposed meta-heuristic enabled MAS optimization architecture. In this architecture, VNS-MH is developed to control DMBH to search the solutions with higher quality. Section 7.5 presents different numerical experiments and their corresponding results. Finally, Section 7.6 summarizes the main contributions of this paper and discusses some interesting research topics for the future.

## 7.2 Literature review

Dynamic and real-time transportation planning usually involves the interactions of transportation companies, who knows customer orders, and drivers, who have real-time routing information. In the recent past, encouraged by the autonomy and distributed intelligence of software agents, researchers in the area of transportation planning have come to propose MAS as a solution approach to model distributed decision support systems and enable optimization through the interactions among the agents. Fischer et al. [1995] modeled transportation vehicles and companies as software agents and used Contract Net Protocol (CNP) as the task decomposition and allocation protocol to obtain

an initial solution; however, the final stage of the problem was solved using a centralized algorithm based on the idea of simulated trading. Kohout and Erol [1999] modeled vehicles and transportation orders as agents in a dynamic vehicle routing scenario, and the solution was improved based on the repeated announcement of the allocated orders in the auction mechanism based on stochastic rules. Satapathy [1999] modeled transportation companies and vehicles as master/slave agents and applied game theory based protocol to coordinate the task allocation and reallocation. Tang and Kumara [2004a] modeled a transportation company and its vehicle in the same level of interaction and designed market based protocol based on stochastic rules to enable distributed search. Unfortunately, for the work listed, the searching process driven by agent interactions easily converge to some local optimality.

From the perspective of interactions in a complex world such as distributed transportation planning, self-interested protocols play important roles. There are three types of interaction protocols: negotiation for two players [Kraus, 1997], CNP or auction for one-to-many relationship [Klemperer, 1999], double auction or simultaneous auction for many-to-many relationship [Walsh and Wellman, 1998; Tang and Kumara, 2004a]. Although different protocols are applied in different scenarios, for a NP-hard scheduling and planning problems, the protocols that enable multiple agents to interact simultaneously are usually efficient. Thus, auction based protocol in the context of MAS, including normal auction, double auction and simultaneous auction, shows the highest interest. Auction algorithm was first proposed by [Bertsekas, 1992] to solve linear programming problems such as network flow and assignment problems, possibly with a distributed computation architecture, to avoid the degeneracy of the simplex method. It was proved

that the optimal solution can be obtained with finite number of iterations. Later, with some relaxation, auction algorithm was applied to solve various NP-hard integer programming problem in the domain of scheduling and planning (e.g. Cerulli et al. [1994]; Wang [1998]; Cerulli et al. [2001]; Parkes and Ungar [2001]; Lee [2002]). In the application domain, auction based distributed algorithms usually demonstrate the ability to solve large scale and heteronomous problems. On the other hand, it is well known that the solution quality cannot be guaranteed to be global optimal due to the intrinsic NP-hard property.

To avoid the local optimal solution of auction based distributed algorithm, efforts are made to improve the algorithm while keeping its scalability. This usually introduce stochastic rules and repeated auction because the sequence of the agents to hold auction is not clear. Kohout and Erol [1999] forces bidders to become auctioneers through stochastic selection to reallocate the tasks already allocated. Tang and Kumara [2004a] force the stochastic selection of askers and bidders in the framework of double auction, where the askers use stochastic selection to choose asks and the bidder use stochastic selection to choose bids. In the improvement strategy, the stochastic rules helps getting better solution luckily, but may also generate worse solution unluckily.

Meta-heuristics are designed to search optimal solutions for NP-hard combinatory optimization problems. Normally, meta-heuristics utilize some high level and global information to gain intelligence for efficient guidance of the searching process. For this reason, meta-heuristics are smarter than simple heuristics. On the other hand, since they work at the abstract level, they can be generalized toward almost any combinatory optimization problems. In the literature, three popular meta-heuristics, which are genetic

algorithms, tabu search and simulated annealing, are famous and have a lot of applications. Readers are referred to Blum and Roli [2001] for a comprehensive understanding of meta-heuristics. Remarkably, meta-heuristics can be adapted to dynamic optimization [Branke, 2001]. Usually, people view meta-heuristic as one indivisible approach, which searches optimal solutions in a heuristic way. However, we separate it into two parts – "meta" and "heuristic". We address the value of "meta" in this paper because we have already have an underlying heuristic for local search. However, "meta" is still connected with "heuristic" to provide a high level control, another kind of heuristic.

## 7.3 Distributed market based heuristic

This section summarizes our previous work in solving the transportation planning problems in an information-rich environment. Our approach is to develop multi-agent and market based distributed heuristic to automatically handle dynamic changes, through seamlessly integrating the processes of planning and execution. In this approach, we model a transportation company and its available vehicles as software agents, which together provide a distributed virtual planning platform. During the planning process, the company agents and vehicle agents exchange the commitment relationship to transportation tasks in an artificial and virtual market according to the double auction protocol [Tang and Kumara, 2004a]. The virtual market and its protocols are named after market based distributed heuristic from the algorithmic perspective. Later, we observed the problem of local optimality of the heuristic and developed an improvement algorithm based on reinforcement learning [Tang and Kumara, 2004b]. But this algorithm is prone to the curse of dimensionality and the state action space is difficult to be approximated.

### 7.3.1 MAS model

From the perspective of decision support system, MAS provides super computing power in the intermediary layer between sensors and human decision makers [Lin et al., 2002]. The agents in MAS usually model those decision makers, instead of the sensor objects, whose information can be collected easily by the agents. Besides the human interaction interface, the agents can do deeper scheduling and planning through interacting with other agents in the system. This is the greatest advantage of MAS. In the following, we first identify the interface and information flows in the agent system; the scheduling and planning algorithms are discussed in next section.

We design three types of agents: company agents, vehicle agents and monitor agents, which play different roles in the transportation planning system. The company agent represents the transportation company. Its behaviors include: (1) monitoring the arrivals of new transportation orders; (2) decomposing orders into transportation tasks in indivisible parcels; (3) monitoring the cancelation of orders; (4) joining the virtual market for task allocation and reallocation. A vehicle agent represents a truck during either the planning stage or the execution stage. Its main behaviors include: (1) monitoring the physical routing process; (2) reporting the location of the vehicle; (3) reporting contingency events; (4) joining the virtual market for distributed heuristic search. The monitor agent is designed for market interactions and represents a virtual market specialist that applies the market rules to coordinate the interactions.

### 7.3.2 Distributed market based heuristic

The distributed market based heuristic (DMBH) is based on a simultaneous double auction protocol. The details of the heuristic developed for the transportation planning, which is also called supply chain procurement problem, can be found in [Tang and Kumara, 2004a]. In this section, we outline the distributed heuristic algorithm.

The agents designed for transportation planning are mapped to a virtual market with double auction protocol. Distinguished from the monitor agent, vehicle agents and company agents are also called task agents because they hold a set of tasks and know how to evaluate their transportation cost. To distinguish between company agents and vehicle agents, each vehicle agent has the ability to plan its tasks to satisfy customers' requirements with the aim of minimizing the transportation cost. In the double auction market, a task agent emulates a trader and a monitor agent corresponds to a market specialist in a real market (e.g. stock market). The monitor agent implements the double auction market rule based on the collected asks and bids from the task agents.

One round of double auction transaction consists of a series of consecutive and distributed computational steps. At the beginning of the transaction, the market is cleared. In this step, the vehicle agents need to discard the proposals in last round. Then the task agents decide their own trading actions, which can be one of the actions of selling, buying or keeping silent, according to a predefined discrete probability distribution (usually a discrete uniform distribution without bias). Later, each selling agent proposes one task to sell (an ask) based on a discrete probability distribution, which is calculated according to the relative values of the deletion costs of its tasks. The monitor

agent collects these asks, and then inform the buying agents. With the set of tasks proposed to sell, each buying agent proposes one of them to buy (a bid) based on another discrete probability distribution, which is calculated according to the relative difference between their insertion costs and deletion costs. After the monitor agent collects all these bids, finally, the monitor agent implements the double auction market rule to suggest the transactions for the task agents. According to the suggested transactions, the task agents update the commitment relationship with their transportation tasks for the next round of transaction. These interactions repeat until the convergence criteria is met or the planning time expires. All these interactions are summarized in Algorithm 7.1.

**Algorithm 7.1**: The local search algorithm using market based distributed heuristic

**input** : a starting solution $x$
**output**: a converged solution $x'$

*transform the solution $x$ into the allocation among the task agents*;
**for** *not converged* **do**
  *the task agents decide whether to sell, buy or keep silent*;
  *the selling agents stochastically decide the tasks to sell*;
  *the monitor agent accepts all the asks and broadcasts to buying agents*;
  *the buying agents stochastically decide the tasks to buy*;
  *the monitor agent accepts all the bids*;
  *the monitor agent applies market rules to matching asks and bids*;
  *the task agents update their commitment relationships*;
**end**
*transfer the converged allocation to the solution $x'$*;
*return the solution $x'$*;

A gap possibly exists between the converged objective and the global optimal objective of the original transportation planning problem. When the transportation planning problem is solved with market based distributed heuristic, the converged solution is reached when all the agents agree not to sell or buy any of their transportation

tasks. The converged objective is calculated as the summation of the operation cost for all the planned vehicles. The converged solution is also called Pareto optimal solution, where each individual in a community refuses to move away from their respective plans. For a NP-hard problem, it is well-known that the Pareto optimality is different from global optimality. This makes the quality of converged solution for the transportation planning problem worse than the global optimal solution. In the real application for a distributed and large-scale problem, this Parto optimal solution has to be accepted. However, if more computational time is given and if it is possible to collect some global information, we are seeking some approaches that can improve the distributed heuristic search.

### 7.3.3 Improvement with reinforcement learning

Reinforcement learning is an improvement approach that we proposed initially (see Chapter 6). Considering that reinforcement learning can learn from simulation or real operations to update the searching knowledge, we proposed the integration of reinforcement learning with the market based distributed heuristic. For the adaption to the reinforcement learning, the assignment relationship between vehicles and tasks is defined as the state variable; the effective proposals during one round of interaction are defined as the action; the cost reduction for one round is defined as the reward; the state transition is defined as the update of the task commitment relationship. Thus, the market based distributed heuristic plays the role of simulation during the learning process to update the knowledge table of reinforcement learning. The learning process ended with the optimal knowledge table, where the global optimal solution can be found.

The details of the integration of reinforcement learning with distributed heuristic refer to Tang and Kumara [2004b].

The introduction of reinforcement learning indeed improve the solution quality of the market based distributed heuristic search. However, this approach is prone to the curse of dimensionality. Usually, the advantage of reinforcement learning is that the state action space can be approximated with neural networks. But at each state, the possible action space is still huge and explored with the heuristic rule by each task agent. Thus, the state action space has to be explicitly enumerated. Consequentially, the storage of the state action space becomes very difficult as the scale of the problem increases. This is the reason why we turn to meta-heuristic approaches.

## 7.4 Meta-heuristic enabled distributed heuristic

In the literature, meta-heuristic approaches have mainly been applied to solving difficult combinatory optimization problems. Compared with those straightforward heuristic approaches, namely local heuristics, which may easily converge to local optimality, meta-heuristics provide a high-level and strategic control over local heuristics. In other words, meta-heuristics direct local heuristics according to some global information and intelligence derived from the information collected. By observing the problem of local optimality of MAS, we propose an integrated architecture of meta-heuristic enabled MAS optimization. In this section, we first discuss the meta-heuristic from a strategic perspective. Based on the perspective, we present the MAS optimization architecture which is meta-heuristics enabled . Following the architecture, we specifically introduce

a meta-heuristic approach – Variable Neighborhood Search (VNS) – as an example to control DMBH for transportation planning.

### 7.4.1 Meta-heuristic strategy

Since almost two decades ago, meta-heuristics, which include Iterative Improvement, Simulated Annealing (SA), Tabu Search (TS), Evolutionary Computation (EC), Ant Colony Optimization (ACO), and etc., have been developed as an important branch of approximate optimization approaches for solving NP-hard combinatory problems [Blum and Roli, 2001]. The concept of meta-heuristic includes two important aspects: "meta" and "heuristic". The opposite of "heuristic" is "exact". An exact search finds an absolutely optimal solution, by either scanning through all the possible solutions or iterating in such a way that the optimal solution must be on the search path (e.g simplex methods). The original meaning of "meta" is "beyond, at an upper level". "Meta" must have the involvement of global information. The combined meaning of "meta" and "heuristic" is to search approximately in an upper level. This distinguishes it from the concept of local heuristics, where approximate search occurs in the low level without much involvement of global information. Due to the lack of global information, local heuristics seems to be very difficult to escape from sub-optimality. In other words, local heuristics are blind to the global optimization, though local heuristics converge well. In this paper, we especially address the integration of local heuristics and meta-heuristics because we are facing the situation that local heuristics exist and work efficiently in a distributed manner.

We strictly distinguish local heuristics and meta-heuristics, though they seem to be similar when used to solve optimization problems. In other words, low-level local heuristics and high-level meta-heuristics are in different hierarchies. In their relationship, meta-heuristics monitor and control over local heuristics; at the same time, local heuristics generate signals of the environment controlled and execute as controlled by meta-heuristics. Several well-known meta-heuristics such as simulated annealing, genetic algorithms, and tabu search already exhibit this type of hierarchical concept, though the underlying local heuristics are so simple as random searches. This hierarchical concept fits well in the situation of introducing meta-heuristics to MAS optimization, where many local heuristics work in a distributed manner but converge to suboptimal solutions efficiently. The introduction of meta-heuristic will enable a higher level control over the myopic and distributed heuristics.

Meta-heuristics can be classified into two categories: trajectory and population based searches. In the trajectory based meta-heuristic, very limited memory is required and the solutions visited during the searching process form a trajectory. Simulated annealing is a good example of this category. During its iterative process, only the current most update solution is kept, though a solution may be visited for multiple times. Its means of using memory is very similar to the basic search except that the trajectory based simulated annealing goes in a circular path for better solutions to be achieved. MAS optimization favor this type of meta-heuristic because no additional protocols are needed to update the global information. However, only limited intelligence can be achieved from the trajectory based meta-heuristic. In the population based meta-heuristic, the solutions visited in the history are remembered in a limited amount of memory. The

meta-control mechanism can have more information to advise the basic search. When this meta-heuristic is applied to MAS optimization, the global information related to multiple agents need to be modeled. At the same time, any dynamic change of the physical and operational environment will cause the update of the global information. Because more control can be taken in population based meta-heuristics, they are more favorable from the perspective of performance. In this paper, a population based Variable Neighborhood Search is proposed to control DMBH.

From a systematic perspective, the introduction of meta-heuristics needs to keep the good properties of MAS optimization. According to Davidsson et al. [2003], MAS optimization is a very flexible architecture and possesses many good properties, which include: (1) finding alternative solutions quickly; (2) supporting dynamic optimization; (3) lacking of global control. For meta-heuristic approaches to be incorporated into the MAS optimization, these good properties still need to be kept. We can design the corresponding protocols to update the global information so that the introduction of meta-heuristic will not damage the good properties of MAS optimization.

### 7.4.2 Meta-heuristic enabled MAS optimization architecture

Meta-heuristics need to be adapted in order to be incorporated into MAS optimization architecture. From the perspective of information systems, the functionalities of upper-level control can be wrapped as a software agent, which can work seamlessly with other existing agents via agent communications. This type of adaption is realized through designing a set of communication protocols. From the perspective of algorithm design, meta-heuristics are forced to use the existing distributed heuristic algorithm as

an embedded local heuristic. In such a scheme, the upper level meta-heuristic benefits considerably from the local heuristic because a lot of infeasible and low-quality solutions

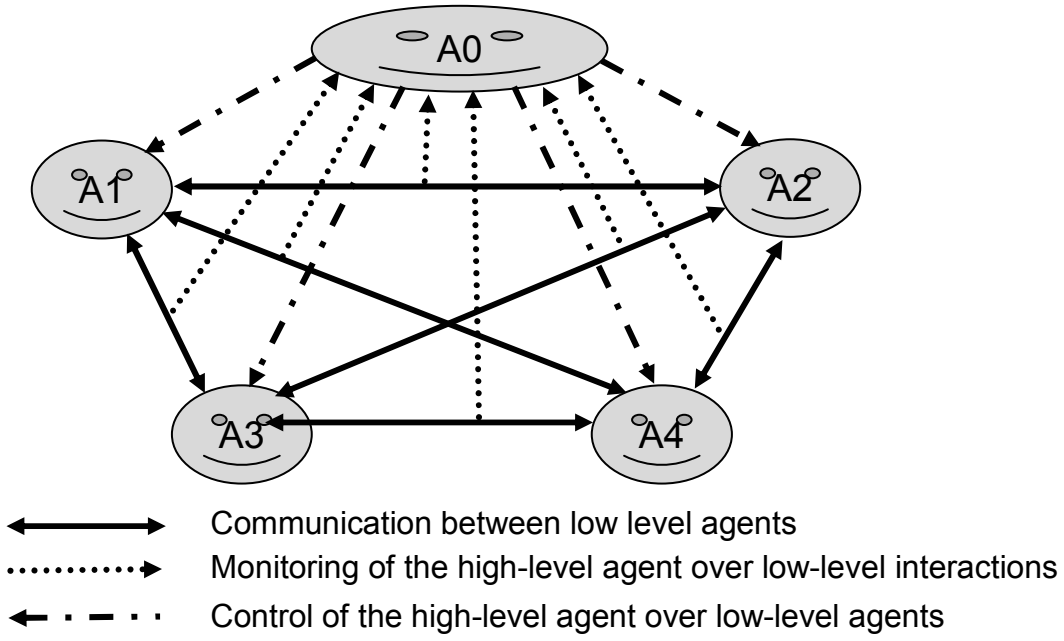


Fig. 7.1. Demonstration of MAS optimization with a meta-heuristic agent as a high-level controller

The relationship between the meta-heuristic agent and other agents who conduct distributed heuristic is shown in Fig. 7.1. This meta-heuristic agent is also called high-level agent, which is denoted by $A0$; other agents such as $A1$, $A2$, $A3$ and $A4$ are also called low-level agents. The local search is carried out through the interactions among the low level agents, which are represented with the solid double arrows. At the same time, the transaction data are sent to the high-level agent $A0$ for the purpose of monitoring.

Periodically or aperiodically caused by some events, the high-level agent sends control signals to those low level agents. Those signals are usually helpful to local search.

### 7.4.3 VNS enabled distributed search

Variable Neighborhood Search is a meta-heuristic [Hansen and N., 1999, 2001], which is based on systematic change of neighborhoods during the search. Different from local heuristics, which use only one neighborhood, VNS divides the search space into multiple neighborhoods based on a predefined structure. Starting from a neighborhood, VNS uses the local heuristic to repeatedly find a local optimality by switching neighborhoods. In other words, if the local search in one of the neighborhoods stops improving the solution, another starting point in the nearby neighborhood is chosen for the next round of local search. DMBH can be considered as a local heuristic. The role of VNS enabled MAS optimization is to repeatedly and strategically set up different starting points (solutions) for DMBH. To elaborate this approach, the following are presented: (1) the representation of a solution – state in the MAS optimization for transportation planning problems; (2) the neighborhood structure adaptive to the market based distributed heuristic; (3) the corresponding modified VNS algorithm.

#### 7.4.3.1 State

A state is an aggregate representation of the solution that needs to be stored concisely. In the double auction heuristic for transportation planning, each vehicle agent contains a set of transportation tasks, whose sequences designate the optimal routing plan for this vehicle agent. The routing plans across all the vehicle agents organize the

solution of transportation planning problem. Instead of directly defining the routing plans as a state, we define the task sets for all the vehicles as a state. To represent these task sets, we use a bipartite assignment relationship between the set of task agents and the set of transportation tasks to represent a state. This representation can be expressed as a vector with a fixed size. The detailed routing plan can be calculated based on a deterministic optimization algorithm [Tang and Kumara, 2004a].

The set of task agents consists of one transportation company agent and $n$ vehicle agents. The task agent $A_i$ ($i \in [0, n]$) holds a set of transportation tasks $\mathbf{t}_i = \mathbf{t}_i^f \bigcup \mathbf{t}_i^c$, where $\mathbf{t}_i^f$ denotes the set of tasks that are fixed (These tasks either have been carried out or are being carried on) in $A_i$'s routing plan, and $\mathbf{t}_i^c$ denotes the set of tasks that are virtually scheduled in $A_i$'s routing plan. The set $\mathbf{t}_i^f$ of fixed tasks is not considered in the market based heuristic because their commitment relationships with $A_i$ are not subject to change. But the set $\mathbf{t}_i^c$ of scheduled tasks is considered for further allocation. Let $\mathbf{t} = \bigcup_{i=1}^{n} \mathbf{t}_i^c = \{t_1, t_2, ..., t_m\}$ denote the set of virtually scheduled tasks throughout the task agents.

Let the vector $s$ with the size $n$ denote a state, where $s(j)(j \in [1, n])$ denotes the index (ranging from 0 to $m$) of the agent who holds the task $t_j$. According to the information in $s$, we can derive the task allocation among all the task agents, i.e. the set of tasks that each task agent holds. This can be realized in Algorithm 7.2. In order for a set of tasks to be converted to a routing plan, we use the heuristic algorithm to compute. On the other hand, a routing plan is first converted to a set of tasks, and then the set of tasks can be converted the state vector $s$ (see Algorithm 7.3).

**Algorithm 7.2**: The algorithm of converting the result of task allocation to a state vector

**input** : the task allocation among all the task agents: $\{\mathbf{t}_i^c | i \in [0, m]\}$
**output**: a state vector $s$

*initialize the state vector* $s$;
**for** *each task set* $\boldsymbol{t}_i^c$ *(*$i \in [0, m]$*)* **do**
  **for** *each task* $t \in \boldsymbol{t}_i^c$ **do**
    *let* $j = k$ *such that* $t = t_k$*, where* $t_k \in \boldsymbol{t}$ ;
    *set* $s(j) = i$;
  **end**
**end**
*return* $s$;

**Algorithm 7.3**: The algorithm of converting a state vector to the result of task allocation

**input** : a state vector $s$
**output**: the result of task allocation: $\{\mathbf{t}_i^c | i \in [0, m]\}$

**for** *each task set* $\boldsymbol{t}_i^c$ *(*$i \in [0, m]$*)* **do**
  *set* $\boldsymbol{t}_i^c = \emptyset$;
**end**
**for** *each element* $j$ *in the state vector* $s$ **do**
  *set* $\boldsymbol{t}_{s(j)}^c = \boldsymbol{t}_{s(j)}^c \bigcup \{j\}$;
  *plan* $\boldsymbol{t}_{s(j)}^c$ *optimally*;
**end**
*return the result of task allocation* $\{\boldsymbol{t}_i^c | i \in [0, m]\}$;

#### 7.4.3.2 Neighborhood structure

The neighborhood structure plays an important role in VNS. Almost all the local heuristics iterate through a sequence of states until a converged state, which is usually the local optimality, is met. Any two consecutive states during the searching path are neighbors, but a state has multiple neighboring states. The beauty of VNS is the systematic change of neighborhoods so that the search process will not be easily stuck in a local optimal state. To realize it, a neighborhood structure is required so that the next starting state is known if the local search process converges. With the starting state, the local heuristic search will probably find another local optimal state, which is probably better than the previous local optimality. From the general perspective, the neighborhood structure represents the whole search space as multiple islands, each of which has several neighborhoods.

The neighborhood structure for transportation planning is different from the standard VNS. In the standard VNS, a neighborhood structure is usually pre-defined analytically based on some simple rules or evaluation criteria, which can be either general or problem oriented. However, this structure is unclear for MAS based transportation planning, especially when the underlying distributed heuristic is modeled as a black box. To generalize our approach, we use a set of states to represent the whole search space and some simple criteria to classify those representative states. In other words, we artificially create a neighborhood structure.

The artificial neighborhood structure is organized in a two-column table with a fixed size. Let $T$ be the table, which has two columns: one for states and the other

for objective values. The capacity of $T$ is fixed and $|T|$ denotes it. All the rows in $T$ are sorted by the objective values. Thus, $T$ stores those representative states and their corresponding objective values. Let $k_{max}$ be the number of neighborhoods. The rows in $T$ are classified into $k_{max}$ categories continuously based on their objective values. If we fix the number of rows in each category, the neighborhood structure based on the table is fixed, though the states in each neighborhood could change during the searching process.

It is important for table $T$ to maintain a good representation of the search space. Since the capacity of the table is fixed, the storage of $T$ is not enough for all the states that the local distributed heuristic visits. If a new state is to be inserted into a full table, a row in the table needs to be replaced with this new state. An update policy needs to be defined with the objective to keep the state table as a best representation of the visited state space, and at the same time to give the recently visited state the highest priority. The problem of update policy is stated in the following.

Let $\{< s_i, v_i >\}$ $(i \in [1, |T|])$ denote the representative table $T$ such that

$$v_i < v_{i+1},\ i \in [1, |T| - 1] \tag{7.1}$$

In another word, the rows in table $T$ are sorted by their objective values. Let $< s^*, c^* >$ be the new state and its corresponding objective value. The update policy can be simply stated as the rule that an existing row in the table is to be updated with this new row. Basically, two problems need to be solved: first, which row is to be removed from the table; second, where is this new row to be inserted.

For the first problem, we need to find the row whose removal influences the overall representation least. Consider three consecutive rows $i-1$, $i$ and $i+1$ and their objective values satisfy the relationship $v_{i-1} < v_i < v_{i+1}$. Qualitatively, if the gap between $v_{i-1}$ and $v_{i+1}$ is very small, the existence of $v_i$ can be ignored. On the other hand, if the gap between $v_{i-1}$ and $v_{i+1}$ is large, $v_i$ will be an indispensable interpolating point. Therefore, we can use the gap between $v_{i-1}$ and $v_{i+1}$ to evaluate the removal of row $i$. Let $g(i) = v_{i+1} - v_{i-1}$ denote this gap. The row to be removed $i^R$ can be calculated as Equation (7.2).

$$i^R = arg \min_{i \in [2, |T|-1]} g(i) \tag{7.2}$$

Since there is no precedent row for first row and no subsequent row for the last row, $g(1)$ and $g(|T|)$ are positive infinite. Thus, they are not accounted during the optimization in Equation (7.2).

For the second problem, we need to insert the new row $< s^*, c^* >$ into the table $T$ without violating the sorted order. Since the table $T$ is still in order after the row $i^R$ is removed, we only need to find a right position to insert $< s^*, c^* >$. Let $i^I$ denote the position at which $< s^*, c^* >$ is going to be inserted and it can be calculated in Equation (7.3).

$$i^I = \begin{cases} i, & \text{if } v^* < v_i, i \in [1, |T|-1]; \\ |T|, & \text{otherwise.} \end{cases} \tag{7.3}$$

Based on the assumption that the table $T$ is full and sorted, the replacement policy can be summarized in Algorithm 7.4.

**Algorithm 7.4**: The algorithm of replacement policy

**input** : a state vector $s^*$ and its objective value $v^*$
**if** *$T$ contains $s^*$* **then**
  *return*;
**end**
*find $i^R$ according to Equation (7.2)*;
*remove row $i^R$*;
*find $i^I$ according to Equation (7.3)*;
*insert $< s^*, v^* >$ at row $i^I$*;

#### 7.4.3.3 Algorithm

The VNS algorithm utilizes the neighborhood structure explained in the last section to provide a high-level control over the distributed market based search. At the beginning, the representative table, which is also called state pool, is empty; an initial state, also called initial solution, is given. The whole meta-heuristic algorithm can be divided into two phases, though they will be seamlessly integrated during the implementation.

In the first phase, the empty state pool is filled with new explored states through running the distributed heuristic search (see Algorithm 7.5). The procedure starts with an initial state. This initial state is then fed into the distributed market based search as the starting state. The local search is monitored and any new state is used to fill the state pool. If the state pool is filled, the local search breaks and the whole procedure returns. If the state pool is not filled but the local search converges, another starting state will be drawn randomly from the existing states and the above steps are repeated. During all the processes, the best solution found so far is recorded and finally returned as the procedure ends. This best solution is still useful during the second phase of search. Due

to the incomplete and unsorted states in the pool, the rules of VNS are not recommended to be applied.

**Algorithm 7.5**: The algorithm of initializing the state pool for VNS

```
input  : an initial state s_0, the size of the pool L
output: the best solution with state s_min and value of the objective function
        v_min
insert state s_0 to the pool;
the state s_min with minimum objective v_min as the initial state and its
objective value;
while the pool is not full do
    pick a state s from the existing state pool randomly;
    initialize the starting state of local search with s;
    while local search is not converged do
        run one step of the local search with s and v;
        if v < v_min then
            set s_min = s and v_min = v;
        end
        insert s into the pool;
        if the pool is full then
            break this loop;
        end
    end
end
sort the pool by objective value;
return v_min and s_min;
```

In the second phase, the artificial neighborhood structure in the state pool provide the knowledge base for VNS. The details of the algorithm are presented in Algorithm 7.6. The algorithm calls Algorithm 7.5 to initialize the state pool and the best solution obtained from the initialization is recorded for further comparison. The algorithm uses two important parameters: the maximum number of rounds of searching $N_{max}$ and the number of neighborhoods $K_{max}$. These two parameters are the controlling variables for two interrelated loops and the searching process will be terminated if these two loops expire. The inner loop iterates on all the neighborhoods in a fixed sequence. At the beginning of each iteration, a state is drawn randomly from the corresponding

neighborhood. Then, this state is taken as the starting state of the local search and the local search runs until convergence. If a better solution is found in this iteration, the inner loop breaks and the next round of outer loop starts. The outer loop just runs until the number of searching rounds reaches the predefined maximum number.

**Algorithm 7.6**: The algorithm of variable neighborhood search using distributed market based heuristic as local search

**input** : an initial state $s_0$, the maximum number of searches $N_{max}$, the size of the pool $L$ and the number of neighborhoods $K_{max}$
**output**: the final state $s_{min}$ and its value of the objective function $v_{min}$

```
initialize the state pool (see Algorithm 7.5);
remember the state s_min with minimum objective v_min;
for n from 1 to N_max do
    for k from 1 to K_max do
        draw a state s randomly from the neighborhood k;
        initialize the starting state of local search with s;
        run the local search until convergence with state s_cvg and value v_cvg
        (see Algorithm 7.1);
        if v_cvg < v_min then
            v_min = v_cvg and s_min = s_cvg;
            break for the next round of search;
        end
    end
end
Return v_min and s_min;
```

### 7.4.4 Adaptive to the dynamic environment

Usually, the original MAS optimization architecture is designed to be adaptive to dynamic changes in the physical environment. In other words, the distributed market based heuristic can perform dynamic optimization subject to any of the following conditions: (1) an empty vehicle becomes available; (2) a vehicle filled with products starts its journey; (3) a new transportation task arrives into the system. These conditions will cover almost all the dynamics in the dynamic and contingency environment. The reason

for the flexibility of dynamic optimization originates from the local information located in each software agent and their instant market mechanism. Although the VNS enabled architecture maintains some global information, the flexibility of dynamic optimization needs to be kept. This can be enabled with the reflection of dynamic changes in the high-level agent, who update the state pool based on those dynamic changes. This section presents the protocols to deal with the conditions of (2) and (3). The condition (1) is not necessary to be dealt with because it does not affect the control of the high-level agent over the low-level agent.

These two protocols update the state vector accordingly. To respond to the condition that a new transportation task comes to the system (condition (2)), all the states in the pool will increase the size by one and this new slot is filled with the index of the transportation company. For the first time that this slot is filled with a vehicle index, all the states in the pool will be updated accordingly. To respond to the condition that a vehicle filled with products starts its journey, all the products in this vehicle will be taken away from the states. Then, the same columns across all the states will be deleted. Therefore, dependent on any dynamic changes of the physical environment, the state pool will be shrunken or expanded in the row dimension.

## 7.5 Numerical results

The VNS-MH algorithm is implemented in Java, same as the distributed market based heuristic (DMBH), which is implemented in a virtual distributed system where the agents communicate via memory. All the computations are carried out in a PC with 512M RAM, Pentium 4 CPU, and Windows XP OS. The same set of sample problems

in Tang and Kumara [2004a] are chosen for numerical experiments. The sample set consists of a number of pickup and delivery problems with time windows in stochastic environments; these problems are generated with reasonable randomization.

The numerical results, which are obtained from different approaches – distributed market based heuristic (DMBH), coordinated reinforcement learning (RL) and VNS-MH – are compared in different aspects. First of all, we choose a specific sample to demonstrate how the VNS-MH combined with DMBH works, by presenting the distribution of routines of the solution, the trajectory history of the meta-heuristic search and resource consumption. Secondly, we analyze how VNS-MH is sensitive to the task sequence of the specific sample. Lastly, we compare the results with larger numbers of transportation tasks.

### 7.5.1 The problem with 210 tasks

In this problem, 100 vehicles with the homogenous capacity of 200 units of volume are available for 210 transportation tasks. Fig. 7.2 shows the distribution of these tasks in an artificial geographical area. The center dot with coordinates $(100, 100)$ serves as the depot, at which each vehicle starts and ends its routine. Other dots are the pickup or delivery locations given by the customers. A transportation task is represented by an arrow, whose tail designates a pickup location and whose head designates a delivery location. If a dot is not connected with an arrow, the pickup and delivery location in the same task coincides. Time windows are not displayed explicitly, but each task node is attached with a time window.

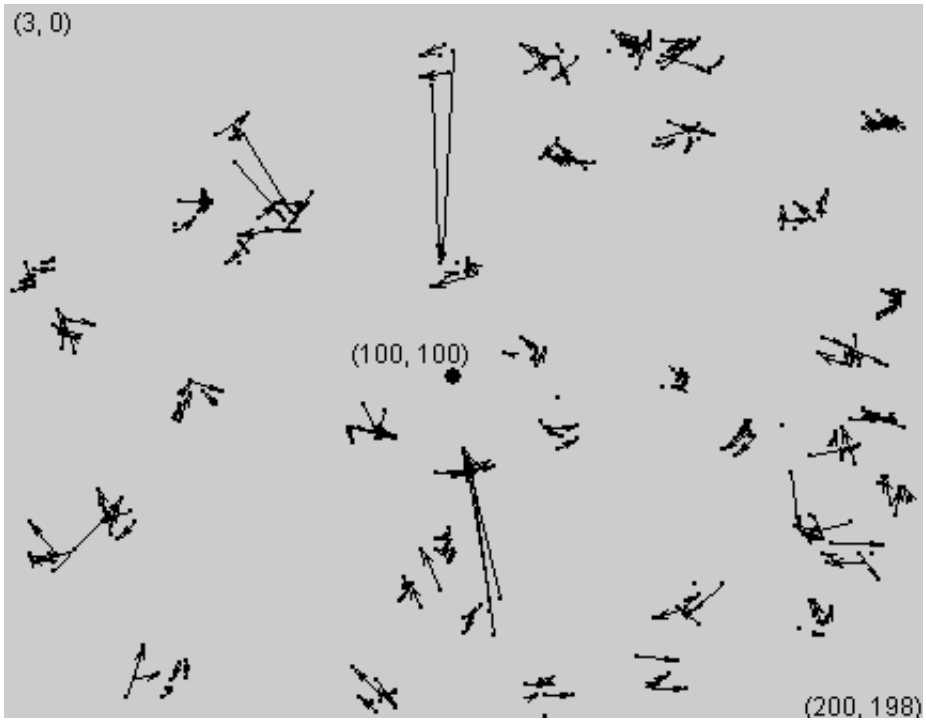


Fig. 7.2. The graphical distribution of 210 tasks in a pickup and delivery problem with time windows

#### 7.5.1.1 Routines comparison

Fig. 7.3 and 7.4 draws the routines generated by DMBH and VNS-MH for a problem with 210 transportation tasks. Both solutions consist of multiple routines, corresponding to each vehicle. Each routine starts from the depot, visits a series of task nodes sequentially, and finally returns to the depot. In these figures, the travel between any two adjacent task nodes is represented by an arrow. We observe that the routines in Fig. 7.3 are not so well clustered as those routines in Fig. 7.4. Also we notice that the allocation with distributed market based heuristic takes more vehicles to finish all the tasks than that VNS meta-heuristic does.

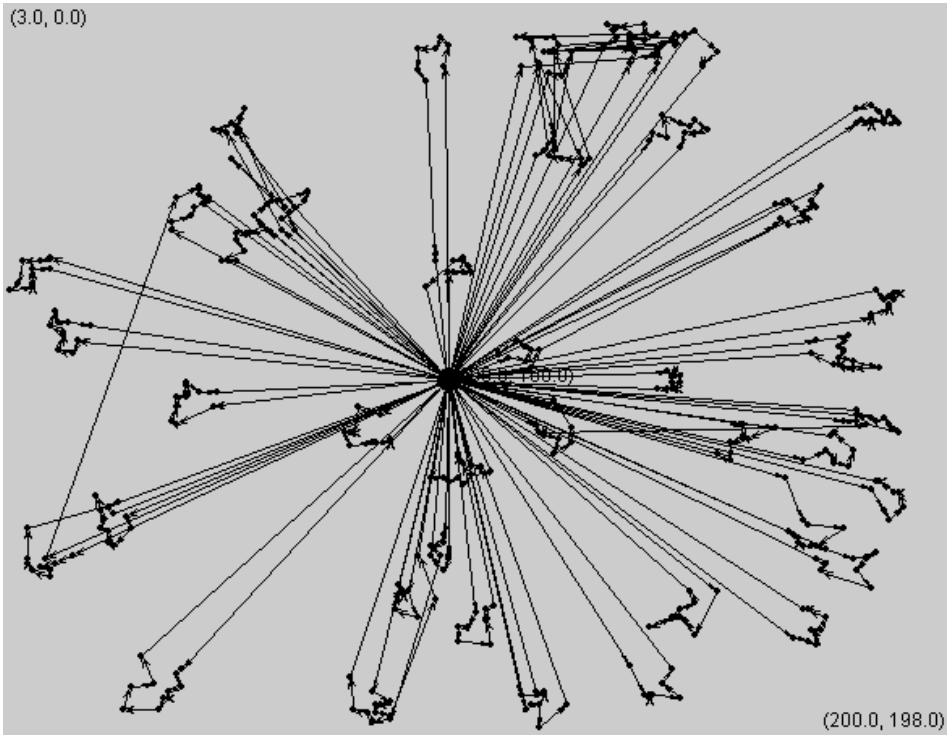


Fig. 7.3. The task allocation on the problem of 210 tasks using double auction heuristic with the transportation cost of 88952.0 units and 47 vehicles

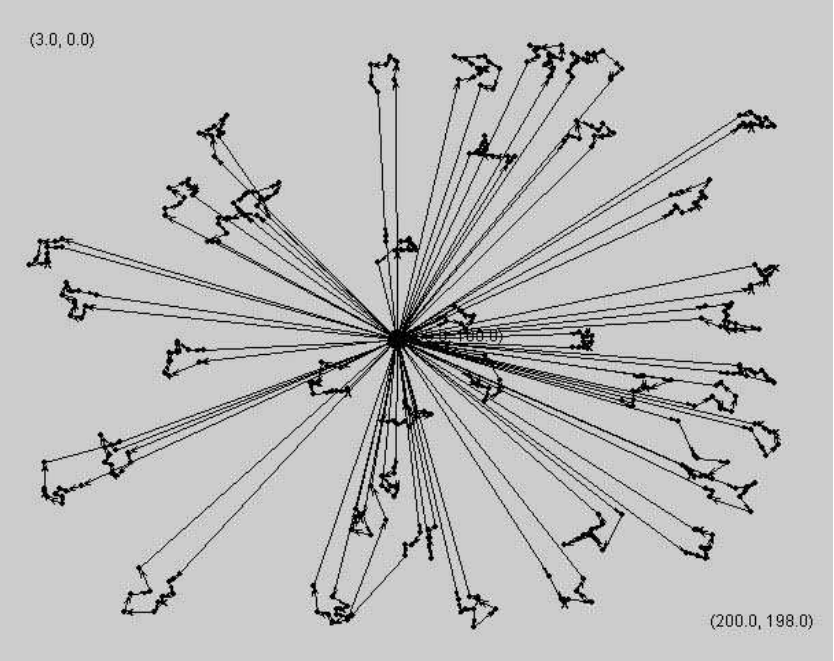


Fig. 7.4. The task allocation on the problem of 210 tasks using VNS meta-heuristic to control the distributed market based heuristic with the transportation cost of 73307.2 units and 41 vehicles

#### 7.5.1.2 Searching history

Fig. 7.5 shows the searching history of the VNS-MH. The horizontal axis represents the search steps, each of which contains a round of market interaction. The vertical axis represents the cost level, which is used to evaluate the quality of a solution. From this observation, we notice that the first round of local search converges at the optimal point $A$, but the optimal solution of the whole VNS meta-heuristic search is located at point $B$. The figure shows an example how VNS meta-heuristic improves the

solution of the distributed market based search in the systematic way of changing the neighborhoods.

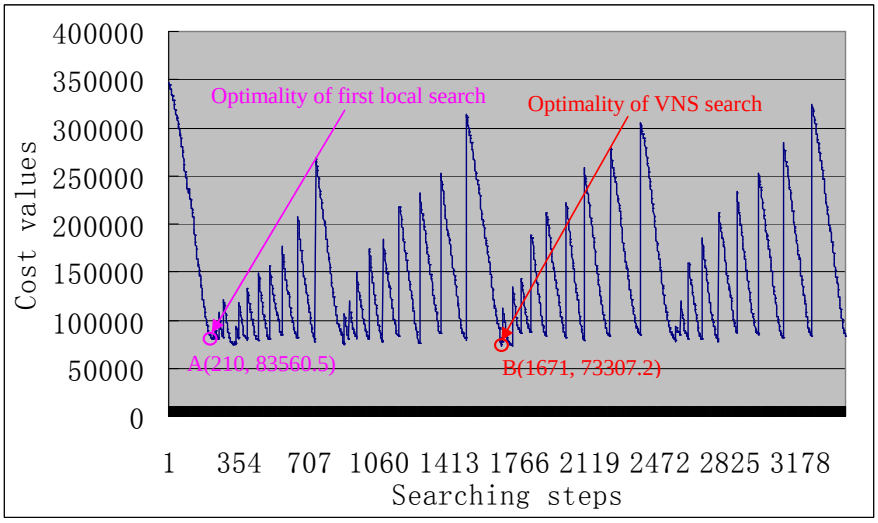


Fig. 7.5. Search history of VNS enabled distributed market based heuristic

#### 7.5.1.3 Comparison with coordinated reinforcement learning

The coordinated Rl is also used to refine the solution of distributed market based heuristic [Tang and Kumara, 2004b]. Table 7.1 summarizes the differences of optimization performance and resource consumption of those three approaches: DMBH, coordinated RL, and VNS-MH. We have the following observations. First, both the VNS-MH and coordinated RL improve the optimal solution over the DMBH to a large extent. From the perspective of optimization performance, VNS-MH almost achieves the same

result as the coordinated RL. However, from the perspective of resource utilization, VNS-MH used a much less amount of memory and less computational time than coordinated RL.

Table 7.1. Performance comparison of three different MAS based heuristic

| Approaches | Vehicle utilization | Optimal cost | Memory | Computation time |
|---|---|---|---|---|
| DMBH | 47 | 88952.0 | 1 | 8.416 |
| coordinated RL | 41 | 73880.1 | 4038 | 270.179 |
| VNS-MH | 41 | 73307.2 | 100 | 221.801 |

#### 7.5.1.4 Sequence sensitivities comparison

In this experiment, we test how different sequences of the transportation tasks affect the VNS-MH and DMBH respectively. Since market transitions are conditional with market rules, the existing commitment relationships of the tasks may pose different market evolutionary scenarios. For this reason, the final cost for the same set of tasks may vary with their different sequences. The curves in Fig. 7.6 and 7.7 show how the sequence affects these two methods for the case of 53 and 106 tasks respectively. In Fig. 7.6, we observed that the final costs for all 35 different sequences obtained from VNS-MH converge to the optimal value, and thus no variance exists in this curve. In Fig. 7.7, we observed that the curve for the VNS-MH is much flatter than the one for DAMM. Table 7.2 shows the standard deviations for different numbers of transportation tasks.

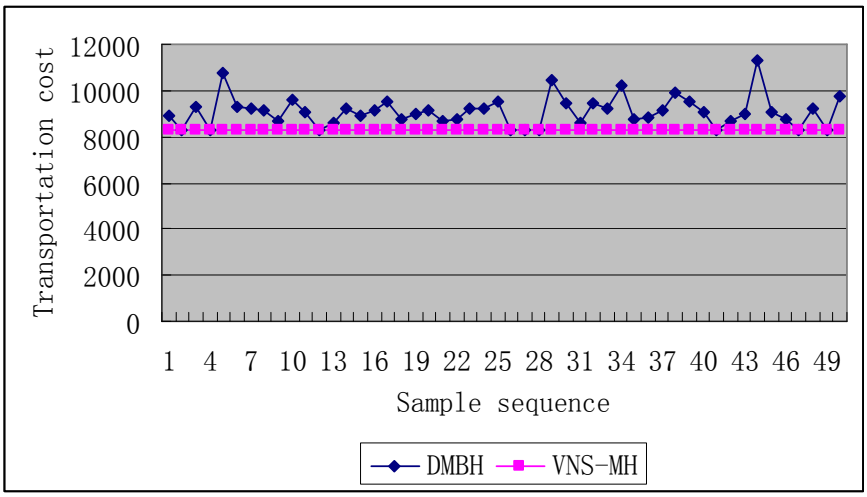


Fig. 7.6. The comparison of the final costs between DMBH and VNS-MH over different sequences of 53 tasks

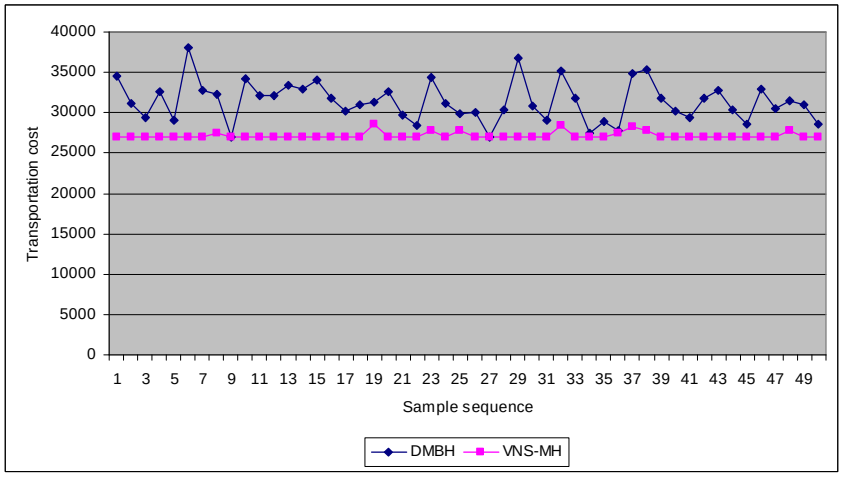


Fig. 7.7. The comparison of the final costs between DMNH and VNS-MH over different sequences of 106 tasks

Table 7.2. Comparison of standard deviations between VNS-MH and DMBH for 53, 106 and 210 tasks respectively

| # of tasks | DAMM STD | CRL STD |
|---|---|---|
| 53 | 628.5 | 0.0 |
| 106 | 2451.8 | 395.5 |
| 210 | 3823.3 | 1674.4 |

## 7.6 Conclusions

Multi-Agent System (MAS) based dynamic optimization architecture will become more and more important, especially in an information-rich environment due to the advances in information technology (IT). However, high quality solution is difficult to achieve because the underlying optimization problem is NP-hard and the distributed search algorithm is prone to converge at local optimality. Meta-heuristics, which seek global optimality, have been successfully applied to solving NP-hard problems. Thus, the combination of distributed heuristics and meta-heuristics provides a unique and effective solution for MAS optimization. In the combinations, limited global information is used to control distributed heuristics in the upper level so that the solution with higher quality can be achieved. The details of the contributions are in the following.

First of all, we propose a meta-heuristic enabled MAS dynamic optimization architecture. In this architecture, any meta-heuristic that uses global information applies a high level control over the interactions of myopic and self-interested agents. At the same time, the introduction of meta-heuristic based global controller will not reduce the flexibility of MAS, which mainly relies on local information of each agent.

Secondly, we developed Variable Neighborhood Search Meta-heuristic (VNS-MH) over the Distributed Market Based Heuristic (DMBH), in the spirit of meta-heuristic

enabled MAS dynamic optimization architecture. In this approach, another agent with global information is introduced to control DMBH. This agent has a fixed amount of memory which approximately represents the states visited in the search. Without interfering too much with the local search DMBH, VNS-MH systematically changes the starting state of the local search based on a systematic strategy. Thus, VNS-MH enables the searching process not to be stuck at local optimal solutions and guarantees high quality solutions with available computational resources.

Thirdly, VNS-MH over DMBH demonstrates better optimization performance than its local search DMBH. On the other hand, VNS-MH exhibits more efficiency than its former global controller – coordinated RL because VNS-MH requires less memory and computational time to achieve the solution with the same good quality.

For future work, it might be significant to continue in the following directions. In the architecture of meta-heuristic enabled MAS dynamic optimization, other meta-heuristics such as genetic algorithm, simulated annealing, and tabu search are open to be incorporated according to the properties of optimization problems. Among them, simulated annealing seems to be has large potential because it uses little global information, which could even be distributed among the agents. Another interesting topic in this architecture is about the interchange of different local heuristics. Over years, researchers have developed various distributed heuristics for MAS under different situations. These heuristics have their strengths and weaknesses. It might be a good research direction to study the development of an integrated architecture that enables the interchange of local heuristics in a higher level controller. Finally, in a distributed system with many software agents, it may be impossible to record the state of the whole system. The agents

may be grouped into multiple communities and meta-heuristic control can be applied in each individual community. For higher level control, another meta agent over all the communities may be designed. This will be extended to hierarchical meta control.

## Bibliography

D. Bertsekas. Auction algorithms for network flow problems: a tutorial introduction, 1992.

C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and concept comparison. Technical report, University Libre De Bruxelles and University Degli Studi di Bologna, October 2001.

Jürgen Branke. Evolutionary approaches to dynamic optimization problems. In Jürgen Branke and Thomas Bäck, editors, *Evolutionary Algorithms for Dynamic Optimization Problems*, pages 27–30, San Francisco, California, USA, 7 2001.

R. Cerulli, R.D. Leone, and G. Piacente. Modified auction algorithm for the shortest path problem. *Optimization Methods and Software*, 4(3):209–224, 1994.

R. Cerulli, P. Festa, and G. Raiconi. Graph collapsing in shortest path auction algorithms. *Computational Optimization and Applications*, 18(3):199–220, March 2001.

P. Davidsson, S. J. Johansson, J. A. Persson, and F. Wernstedt. Agent-based approaches and classical optimization techniques for dynamic distributed resource allocation: A preliminary study. Working paper, 2003.

K. Fischer, J. P. Muller, M. Pischel, and D. Schier. A model for cooperative transportation scheduling. *Proceedings of the First International Conference on MultiAgent Systems*, pages 109–116, 1995. AAAI Press, MIT Press, San Francisco, California.

P. Hansen and Mladenovic N. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, pages 449–467, 2001.

P. Hansen and Mladenovic N. *Meta-heuristics: advances and trends in local search paradigms for optimization*, chapter An introduction to variable neighborhood search, pages 433–458. Kluwer Academic Publishers, 1999.

P. Klemperer. Auction thoery: A guide to the literature. *Journal of Economic Surveys*, 13(3):227–286, May 1999.

R. Kohout and K. Erol. In-time agent-based vehicle routing with a stochastic improvement heuristic. *Eleventh Conference on Innovative Applications of Artificial Intelligence*, 1999. Orlando, FL.

S. Kraus. Negotiation and cooperation in multi-agent environments. *Artificial Intelligence*, 94(1-2):79–97, 1997.

Y. Lee. *Market-based Dynamic Resource Control of Distributed Multiple Projects*. PhD thesis, Pennsylvania State University, August 2002.

G. Lin, S. Buckley, H. Cao, N. Caswell, M. Ettl, S. Kapoor, L. Koenig, K. Katircioglu, A. Nigam, B. Ramachandran, and K.Y. Wang. The sense-and-respond enterprise: Ibm researchers develop integrated sar system of global value chain optimization. OR/MS Today, April 2002.

V.V. Miagkikh and W.F. Punch. An approach to solving combinatorial optimization problems using a population of reinforcement learning agents. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1358–1365, Orlando, Florida, USA, 13-17 1999. Morgan Kaufmann. ISBN 1-55860-611-4.

D.C. Parkes and L.H. Ungar. An auction-based method for decentralized train scheduling. *Fifth International Conference on Autonomous Agents*, pages 43–50, May-Jun 2001.

G. Satapathy. *Distributed and Collaborative Logistics Planning and Replanning Under Uncertainty: A Multiagent based approach*. PhD thesis, Industrial and Manufacturing Engineering Department, PA, December 1999.

K. Tang and S. Kumara. Double auction market mechanism: A distributed negotiation protocol to model an e-procurement problem. *IEEE Automation Science and Engineering*, 2004a. accepted.

K. Tang and S.R.T Kumara. Using reinforcement learning to refine the distributed supply chain procurement plans. Technical report, Pennsylvania State University, 2004b.

K. Tang, S.R.T Kumara, S. Yee, and J. Tew. Wireless-based dynamic optimization for load makeup using auction mechanism. In *IIE Annual Research Conference*, 5 2004.

W. Walsh and M. Wellman. A market protocol for decentralized task allocation. *In Third International Conference on Multi-Agent Systems*, pages 325–332, 1998.

K. Wang. Demand-driven shop-floor control model for distributed manufacturing systems. *Journal of the Chinese Society of Mechanical Engineers*, 19(6):615–623, Dec 1998.

G. Weiss, editor. *Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, Cambridge, Massachusetts; London, England, 1999. ISBN 0-262-23203-0.

M. J. Wooldridge. *An introduction to multi-agent Systems*. Chichester, UK: Wiley, 2002. ISBN 0-201-36048-9.

## Chapter 8

# Cooperation in a Multi-stage Game for Modeling the Distributed Task Delegation in a Supply Chain Procurement Problem

**Abstract:** A multi-player game in multiple stages was developed to model task delegations among transportation companies with overlapped transportation networks in a supply chain procurement problem. This game is designed to deal with the infeasible transportation tasks left over by the transportation planning of each individual companies. Although the task delegation can be enabled through market interactions, the self-interest of each company has to be addressed. This turns into a game in multiple stages, each of which incurs market interactions. In the game, multiple players, representing a set of transportation companies, aim to maximize their own payoff summations over multiple stages. By playing the game repeatedly, we develop an evolutionary method, which combines the reinforcement learning and fictitious playing, to seek the solution. Numerical results show that this combinatory methodology can find the evolutionary stable equilibrium of the game.

**Keywords:** self-interested, supply chain procurement, multi-stage game, evolutionary method, reinforcement learning, fictitious playing

## 8.1 Introduction

In the environment of Business to Business (B2B), domestic and global partners need to work collaboratively to share resources or form coalitions to execute tasks [Fischer et al., 1995; Satapathy, 1999; Shehory and Kraus, 1998]. These related partners aim to pursue their own objectives during their collaborations because they are rational[1] entities. This problem is usually modeled as a multi-criteria optimization problem [Chankong and Haimes, 1983; M. and Pet-Edwards, 1997]. Recently, it has come to be modeled using game theory [Chatterjee and Samuelson, 2001], in which the rationality and the objective of each individual party is addressed. More specifically, rationality is modeled in the equilibrium, which defines the solution concept of game theory and tells that each party reaches its own individual optimality. *As a special case of general B2B problems, a supply chain procurement problem, concerning the task delegations among self-interested transportation companies, is modeled using game theory, so that its solution concept can be used to solve the problem.*

A supply chain procurement problem includes two optimization problems: the vehicle routing problem in the same transportation company and the task delegation problem among multiple transportation companies. The first problem has a unique objective, but the second one has multiple objectives because each company has its own self-interest. In order to satisfy the realtime and distributed requirements of these problems, both transportation companies and vehicles are modeled in a Multi-Agent

[1]The meaning of rationality in economics is different from its philosophical meaning. The rationality in economics can be replaced with another word: selfishness. Rational entities in the market only care about their own efficiency and has no intention to benefit others by sacrificing their own payoffs.

System (MAS), and their interactions are modeled with the Double Auction Market Mechanism (DAMM). Based on different interactions of those two problems, we develop two hierarchies of markets from the perspective of economics (see Fig. 8.1). In the first hierarchy, we solve the problems for individual transportation companies, with multiple local markets modeling the interactions among vehicles; in the second hierarchy, we solve the problem for multiple transportation companies, with a global market modeling the interactions of these local markets. These two hierarchies of virtual markets imitate real practice in the industry.

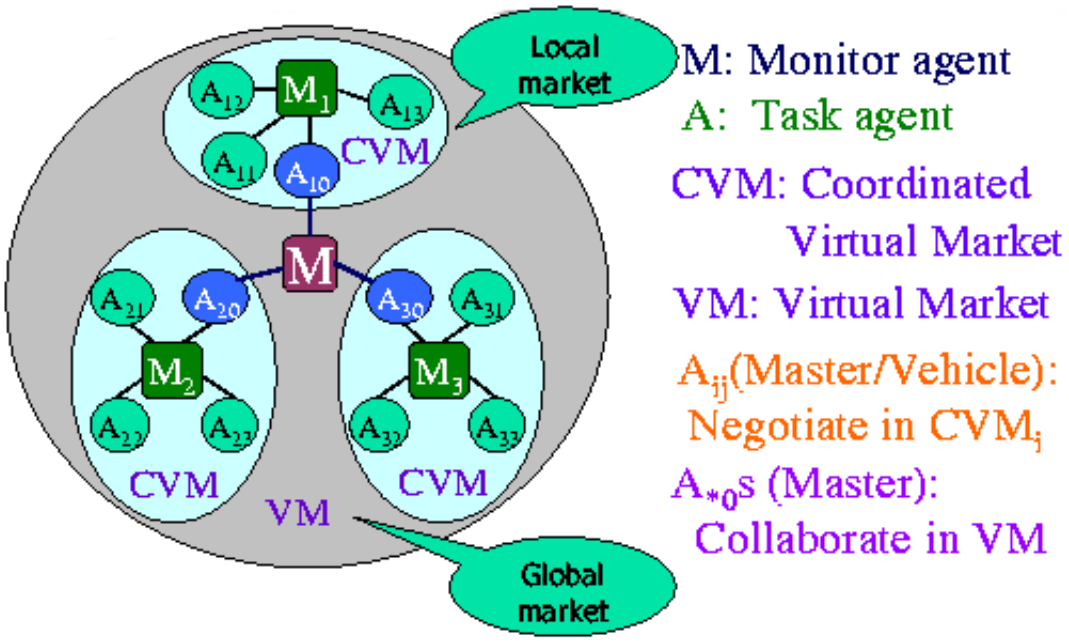


Fig. 8.1. Two hierarchies of double auction markets for modeling a supply chain procurement problem

We have solved the problem in the first hierarchy using the double auction heuristic [Tang and Kumara, 2004] and Coordinated Reinforcement Learning [Tang and Kumara, 2002]. Although we can solve the problem in the second hierarchy with the same heuristic, it is difficult for the solution to capture the self-interests of those companies. Therefore, in our model, these companies plays a multi-stage game, where each stage invokes a round of double auction interactions. To solve this game, we propose and develop an evolutionary approach that combines reinforcement learning and fictitious playing, based on the equilibrium concept of the game theory.

The remaining of this paper is organized as follows. Section 8.2 introduces a multi-stage game for modeling the distributed task delegation in the context of the supply chain procurement problem. Section 8.3 discusses the equilibrium concept in the game theory and explains the reason to propose an evolutionary method. Section 8.4 presents the details of the evolutionary method that combines reinforcement learning and fictitious to solve the problem of multi-stage game. Section 8.5 demonstrates numerical results and discusses the effectiveness of the combinatory approach. Finally, Section 8.6 summarizes the conclusions and future work.

## 8.2 Problem definition: a Multi-stage cooperative game

Let $\mathbf{A} = \{A_1, A_2, ..., A_n\}$ denote a set of agents, which represent a group of transportation companies who participate in a virtual double auction market for task delegation. Let $\mathbf{t}_i = \mathbf{t}_i^{fea} \bigcup \mathbf{t}_i^{infea}$ denote the tasks that agent $A_i$ is responsible for from its own customers, where $\mathbf{t}_i^{fea}$ denotes the set of tasks that are feasible to be allocated to $A_i$'s vehicles with the customer requirements satisfied and $\mathbf{t}_i^{infea}$ the set of tasks that

are infeasible to be allocated to $A_i$'s vehicle agents due to the restriction of either vehicle capacity or or time windows. The set $\mathbf{t}_i^{fea}$ and $\mathbf{t}_i^{infea}$ are called feasible and infeasible sets respectively. The agents in the set $\mathbf{A}$ considers delegating their respective infeasible tasks to each other. Let $\mathbf{t} = \bigcup_{i=1}^{n} \mathbf{t}_i^{infea} = \{t_1, t_2, ..., t_m\}$ denote the sets of all the infeasible tasks that need to be delegated through the continuous interactions of the double auction market. The issue is how to find an optimal plan for all the participating agents to delegate their infeasible tasks through solving a multi-stage game.

Let $s = \{\xi_1, \xi_2, ...\xi_m\}$ denote the state of virtual market, where $\xi_i$ $(i \in [1, m])$ denote the subscript of the agent who holds the task $t_i$. $s$ is transparent to and shared by all the agents during the multi-stage game. In fact, $s$ memorizes the situation of task delegation and can be converted to and from the distribution of those infeasible tasks among the agents in the set $\mathbf{A}$.

Let $a = (\delta, \rho)$ denote an action to be played by an agent, where $\delta \in \{sell, buy, silent\}$, and $\rho$ denotes the subscript (-1 for silence) of the task $(t_\rho)$ that is dealt with in the action. For example, if an agent is to sell task $t_i$, its action can be denoted by $(sell, i)$. At a state $s$, agent $A_i$ plays with her own action $a_i$. At the same time, other agents issue their actions $\mathbf{a}^{-i} = \bigcup_{j \neq i} \{a_j\}$. So, the collective actions of all the agents can be denoted by $\mathbf{a} = \{a_i, \mathbf{a}^{-i}\}$.

The payoff function and state transition of the game are described in Equation (8.1) and (8.2). Although the payoff for an agent is represented as the function of the current state and its own action, the payoff value is determined with the collection of the actions of all the agents based on the market rule. Likewise, the result of state

transition is also related to the actions of all the agents[2]. The calculation of payoff and state transition can only be obtained through intensive computational algorithms.

$$r_i(s, a_i) = r(s, \{a_i, \mathbf{a}^{-i}\}) \tag{8.1}$$

$$s' = \eta(s, \{a_i, \mathbf{a}^{-i}\}) \tag{8.2}$$

Similar to the market interactions among task agents inside a single transportation company, the market interactions for multiple transportation companies occurs continuously until convergence. A round for a single company corresponds to a stage in the game for multiple companies. At each stage of market interactions, each agent can submit at most one proposal. With the proposals submitted, the virtual market transits to a new state. Starting from the new state, another stage of interactions will be repeated. All these interactions start with a given initial state $s_0$, which usually corresponds to the most expensive task delegation. All these states and their transitions exhibit a tree-like structure. At each node of the tree, all the agents play a strategic form game. All the sub-games distributed in the tree-like structure constitute the whole multi-stage game. In this multi-stage game, each agent tries to maximize her accumulated payoffs by choosing actions that lead to the cooperation fully with other agents. The convergence criteria for a single company corresponds to the ending status of the game for multiple companies. The ending status can be designated when all the self-interested agents decide to stop proposing due to the lack of additional profit earning.

[2]Sometimes, $s = s'$, i.e, no state transition happens because no positive payoff is received with the action set $\mathbf{a}$.

In this multi-stage game, the tree-like structure can be extended to contain all the possible states, actions, payoffs and state transitions; however, not all the possibilities that are enumerated are explicitly modeled. Two reasons cause the strategy of incomplete enumeration. First, the number of states grows exponentially with the number of tasks so that it is impossible to enumerate and store all the possible states before hand. Second, it is not necessary to store all these enumerations because a large amount of non-beneficial or even infeasible actions exist in the game. Therefore, only a partial tree is stored, and it grows as more states are explored.

## 8.3 Equilibrium concept in game theory

To seek the solution of a game means to find an equilibrium point among all the possible actions' combinations of multiple players. An equilibrium is defined at the point where each agent in the game has no intention to move away from their actions in a repeated game [Myerson, 1997; Gibbons, 1992]. In a finite game, Nash Equilibrium (NE) provides a solution concept; however, in practice it is unknown to find out all the equilibriums for a general game with more than 3 players. In order to achieve the solution of a game practically, researchers have proposed different modified versions of NE [Rubinstein, 1998]. In this paper, the concept of Evolutionary Stable Strategy (ESS) is borrowed [Samuelson, 1997] for solving the multi-stage game in the context of the supply chain procurement problem.

The NE concept is based on infinite rationality[3], i.e. an agent knows his opponents, knows that his opponent knows himself, knows that his opponents know that he

[3]Rationality here means the ability to reason.

knows his opponents, etc., till infinity. But this infinite rationality has not pragmatic meaning because software agents are limited in computation and memory, i.e., their reasoning abilities are limited. Due to the limitation, each agent is assumed to have only bounded rationality [Rubinstein, 1998]. Accordingly, for the agents with bounded rationality, the concept of equilibrium needs also to be updated to reflect this limitation.

Therefore, the concept of ESS is introduced. This concept is a combination of NE and convergence theory of the evolutionary computation. In specific, the concept is defined at the point where the proposed actions of all the agents in the game converges during a repeated game based on their own interests [Samuelson, 1997]. The repetition of a game such that the agents can update their decision policies is usually termed as a learning process. In other words, the concept of ESS is borrowed to judge the learning process to see whether the convergence criteria of learning is satisfied or not. Hence, this concept is remarkably important in solving a multi-stage game.

In the literature of game learning, two main types of methods are developed to seek the equilibrium solution based on the evolutionary computations: Fictitious Play (FP) [Fudenberg and Levine, 1998] and reinforcement learning (RL) [Borgers and Sarin, 1997].

FP provides a way for an agent with her limited memory to model other agents. By memorizing and updating the occurrences of other agents' actions, a modeling agent builds her belief about other agents to be modeled. Based on the belief, the modeling agent can guess the actions of all the modeled agents, and then find out her own action in a myopically optimal way. During a repeated process of playing the game, as the belief about other agents becomes convergent, the optimized action of the modeling agent

will converge too. If the agents in a game learn concurrently, the agents' actions will finally converge to the equilibrium solution[4]. In practice, if the chosen action of each agent shows some cyclical pattern [Sonsino, 1997], we can derive that the convergence is achieved. The action profile at the convergence is defined as the equilibrium point.

Although FP has a promising property of convergence, researchers discuss its limitations [Krishna and Sjostron, 1995; Williams, 2001; Bowling and Veloso, 2002]. First, the objective functions of all the agents need to be identical in order for FP to succeed [Monderer and Shapley, 1996]. Second, the convergence is very sensitive to the assumed initial belief about other agents and it seems that this initial belief is the driving force for FP to move forward [Garcia et al., 2000]. Finally, FP can only make the game with the dominated pure strategies converge [Bowling and Veloso, 2002]. Therefore, it is risky to apply the pure fictitious play for the game learning. In the whole multi-stage game learning for the supply chain procurement problem, FP serves only as a supplementary method to RL.

The process of RL can be considered as an implicit Markov Decision Processes (MDP) without modeling the probabilities of state transitions explicitly [Puterman, 1994; Sutton et al., 1991]. It has been successfully applied in many applications for single-agent learning [Bertsekas and Tsitsiklis, 1996; Kaelbling et al., 1996; Singh et al., 1996; Barto et al., 1993]. Based on the Bellman optimization equation, it is proved that the control policies will converge to the optimal solution after infinite number times of reinforcement play [Mitchell, 1997]. However, when RL is applied to the game learning with multiple agents, two distinctions with single-agent cases need to addressed. First, instead of

[4]Based on the game theory, there may exist multiple equilibrium points.

exploring states and actions as a combination in the single-agent learning, each agent in the game learning needs to explore states and actions separately because a state is a piece of shared information but its actions are private decisions. Thus, the objective of the state exploration is to go through as many states as possible, but the objective of the action exploration is to provide knowledge of action selection based on some biases [Thrun, 1992; Meuleau and Bourgine, 1999] as accurate as possible. Second, since the self-interested policy of an agent is based on other agents' self-interested policies that are changing during the learning process, each agent is learning a moving target in a non-stationary environment [Mundhe and Sen, 2000; Sen et al., 1994; Bowling and Veloso, 2002], whatever the action profiles of other agents are learned. The characteristics of moving target and non-stationarity default the ordinary RL that aims to learn the mean optimal values in a stationary environment. For this reason, the learning process itself needs to select actions based on some biases. This is realized through decayed RL [Claus and Boutilier, 1998].

There are two types of game-oriented RL according to whether to model the joint Q values[5] with other agents or not [Claus and Boutilier, 1998]. The joint Q values can be explicitly modeled when only two agents play a game. However, the model becomes exponentially complex as the number of agents grows. Even if only three agents are considered, the model of joint Q values is difficult to describe. In a virtual market, it is common to see more than three agents involved in the play. As a compromised solution, we model her own Q values in a Q table, instead of modeling joint Q values for each agent; as a supplement, we model their action preferences in the framework of FP with

[5]Q-learning is one of the most popular updating strategies of RL.

limited memories. Consequently, FP and RL work together to solve the multi-stage game with multiple self-interested agents.

## 8.4 Approach to seek the equilibrium solution

The details about how the evolutionary method is applied to seek the equilibrium solution are explained in this section.

### 8.4.1 Assumptions and Corollaries

The multi-stage game designed for distributed task delegations in this paper is similar to a Poker gamble. At each stage, each agent choose an action without knowing other agents' actions, and then the payoff for the play will be distributed to each agent immediately based on the cooperation and competition in the game. However, the distinctions between them need to be addressed.

First of all, the objectives of the players in these two games are different. The objective of a Poker player is to win based on the competition with other players, while the objective of a transportation company agent is to obtain the highest accumulated payoffs starting from an initial state $s_0$ based on the cooperation with other agents. Although a lot efforts have been made to model a Poker player using learning algorithms [Billings et al., 2002], they are not able to be used directly in the multi-stage game for task delegations for two reasons. First, the former system is built for a single player through simulating the possible interactions of multiple players; the latter system is built in a MAS for multiple agents to update their virtual and dynamic plans concurrently. Second, the former needs to explore the optimal actions at any possible state in the game

in case the situations may change without notification during playing; the latter needs to seek the optimal actions only starting from the initial state $s_0$.

These distinctions result in different learning systems, which are developed based on different assumptions about the corresponding games. The multi-stage game for task delegations is based on the following assumptions:

**Assumption** 1. *States and the game tree defined by the state transition are transparent to all the agents.*

**Assumption** 2. *The communication cost is considered as a small value in the computation of the marginal cost.*

**Assumption** 3. *The actions of other agents can be observed.*

**Assumption** 4. *The actions to choose and the Q values of the actions are private information.*

**Assumption** 5. *The calculation of marginal cost is public to all the agents.*

Based on the above assumptions, the following corollaries are derived:

**Corollary** 1. *The learning objective for an agent is the Q values of individual actions for all the states, according to assumption 1 and 4.*

**Corollary** 2. *If an agent cannot compete with other agents by issuing either selling or buying action, she should keep silent to save the communication cost, according to assumption 2.*

**Corollary** 3. *All the agents will keep silent if no task delegation will benefit each other, according to corollary 2.*

**Corollary** 4. *At each state, each agent has $m+1$ possible actions, $m$ of which are related to selling or buying and one of which is the silent action.*

### 8.4.2 Multi-stage reinforcement learning

RL plays an important role in seeking the equilibrium solution in the multi-stage game for task delegations. Usually, Q-learning, a special RL is applied for a single agent playing with its environment. In the Q-learning, updating the Q table and exploring the states and actions are two important components. Their importance still holds in the Q learning for multiple agents in a multi-stage game. However, in this paper, these two components are adapted to the context of task delegations in a supply chain procurement problem. This adaption results in modifications of the multi-stage RL based on the RL for a single agent. In the multi-stage RL, the modifications concerning these two components are in the following.

- **Private Q tables**

  Each agent has a Q table, which stores the Q values for the possible state-action pairs; each agent updates each Q table independently. Although all the agents in the market share the state and state transition of the whole market based on Assumption 1, each agent has its own decision policy to learn in the repeated games. Therefore, the updating processes of Q values need to be separated between any two different agents.

- **Two-step exploration**

  The exploration in the multi-stage RL consists of two steps. In the first step, the state explorations for all the agents are synchronized through the monitor agent. In this step, each agent does not deal with the knowledge to learn, however, the design of this step influence the learning speed. In the second step, each agent does her own action exploration for the explored state in the first step. This exploration is based on the relative Q values, which are automatically decaying such that the actions with large Q values will be played more and more frequently. These Q values are the important knowledge for this agent to learn.

#### 8.4.2.1 Q value update

As described above, each possible state-action pair has its own Q value, and it will be updated with some delay. Let $q(s, a)$ denote the Q value of action $a$ at state $s$. Let $\alpha$ denote the proportion of the new explored value in the updated Q value. $\alpha$ is also called the learning rate [Bowling and Veloso, 2002]. Let $\gamma$ denote the discounted rate of the Q values across adjacent stages. Let $s'$ denote the next state resulting from the state transition at the state $s$. Let $\mathbf{a}'$ denote the action set at state $s'$. Thus, the update of a Q value is shown as in Equation 8.3, where $q(s, a)$ and $q(s', a')$ are the Q values for the state-action pairs $(s, a)$ and $(s', a')$ respectively.

$$q(s, a) \leftarrow (1 - \alpha)q(s, a) + \alpha(r + \gamma \max_{a' \in \mathbf{a}'} q(s', a')) \tag{8.3}$$

#### 8.4.2.2 State level update

In the game tree shared by all the agents, each node represents a state and the root of the tree is the initial state. Based on the structure of multi-stage RL, all the state transitions can be traced to the root of the game tree. Let $T(S, R)$ denote this game tree, where $S$ represents the set of all the states explored and $R$ represents the set of all the state transitions in $S$. In specific, let $r \in S$ denote the root of the tree and $S_{leaf}$ denote the set of leaf states. Let $l(s)$ denote a state's level, which is calculated recursively. If $\{l(s) = 0 | s \in S_{leaf}\}$ denotes the levels of the leaf states, the levels of all the states can be calculated in Equation (8.4),

$$l(s) = \max(l(s), 1 + \max_{s'=\eta(a,\cdot), a \in \mathbf{a}_s} l(s')) \tag{8.4}$$

where $\mathbf{a}_s$ is the set of available actions at state $s$ and $s'$ is one of the next states transited from $s$.

The level of a state is updated together with the update of Q values. When a new state is generated, the levels of the states traced back from this new state to the root state are updated according to Equation (8.5). These levels are useful for the state exploration, which is coordinated by the monitor agent.

$$l(s) \leftarrow \begin{cases} 1 + l(s), & \text{if } 1 + l(s') > l(s) \\ l(s), & \text{otherwise.} \end{cases} \tag{8.5}$$

#### 8.4.2.3 State exploration

Based on RL theory, the learning agent needs to visit every state infinite number of times in order for all the Q values to converge to the true values. This can be realized with a random walk along all the possible states, if the learning speed is not crucial. However, random walk is the worst exploration strategy, so we need to design an efficient exploration strategy which is based on heuristic rules and adjusted to specific problems. Based on the relationship between the level of a state and its accumulated reward, i.e., the higher level of a state, the higher Q value starting from the sate, we can take advantage of exploring the states of high level more frequently than those of low level. At the same time, the states of low level need to be explored with some frequency. Thus, the Bolzmann's exploration technique, which fosters biases toward the most advantageous states, is applied to all the existing states based on their levels. Of course, the levels of the states are being updated as a new state is explored.

Let $S$ denote the set of all existing states. Let $L(s)$ denote the number of visited times a state $s \in S$ is visited. Let $\theta_1$ denote a constant called the temperature parameter required in the Bolzmann's exploration. Then the probability distribution over all the states can be calculated as in Equation (8.6). With this probability distribution, a discrete random generator chooses a state from all the available states by taking the advantage of the state with the highest probability, still yielding opportunities to other states of low levels. Some deterministic exploration strategies, which are not listed here, can also be applied [Thrun, 1992]. These represent some heuristic strategies which are closely related to the problems to solve.

$$p(s) = \frac{e^{l(s)\theta_1^{-1}}}{\sum_{s' \in S} e^{l(s')\theta_1^{-1}}} \tag{8.6}$$

#### 8.4.2.4 Action exploration

Action explorations is performed after state explorations, whose results tell the starting states at which to explore actions. However, the processes of action explorations among multiple agents are independent of each other. Therefore, the action exploration in a single agent is discussed as an individual component, and the action explorations in other agents can be copied from this one. Similar to the state exploration, the modified Bolzmann exploration is applied to possible actions.

According to the characteristics of reinforcement learning, the action exploration ought to help the process of updating Q values so that they can converge to the true values; furthermore, in a multi-stage game, the exploration ought also to build up the necessary knowledge for the agents to make decisions after the learning is completed. With these insights, it can be derived that the probability distribution for the possible actions needs to converge as well. In addition, the convergence of the Q values and that of the probability distribution are interwoven and influence each other.

Deliberately, a decayed Bolzmann exploration process is designed to realize the above two objectives. During the early stage of learning, the actions with different Q values are equally important. As the learning progresses, the more number of times is a state explored, the more important is the action with a high Q value. Thus, before being forced to converge, the exploration process spends a relatively long period adjusting actions because every possible action is given enough opportunity to increase its Q value.

Since the actions with higher Q values are played with more times, these Q values will become more and more close to the true Q value. As this gradual convergence spreads to the whole agent community, the actions of other agents are gradually fixed so that the environment for each agent to learn becomes more stable. This gradual stability marks the closeness to the objectives of all the agents.

At state $s$, agent $A_i$ has a set $\mathbf{a}_i^s = \{a_1, a_2, ..., a_{m+1}\}$ of possible actions to explore. Let $k_s$ denote the number of visited times of the state $s$. Let $\mathbf{q}_i^s = \{q_1, q_2, ..., q_{m+1}\}$ denote the set of Q values of these actions. Let $\theta_2$ denote a constant called the temperature parameter. Then the probability distribution to explore among the actions in the set $\mathbf{a}_i^s$ is shown in Equation (8.7).

$$p(i, s, a_j) = \frac{e^{q_j \theta_2^{-1} k_s}}{\sum_{l=1}^{m+1} e^{q_l \theta_2^{-1} k_s}} \tag{8.7}$$

If $k_s$ is taken out of Equation (8.7), it will be an ordinary Bolzmann exploration equation. When $k_s < \theta_2$ at the early stages of learning, the probability to explore each action is almost the same due to the effect of exponential function. However, when $k_s > \theta_2$ at the later stages, the probability to explore the action with the highest Q value becomes high due to the amplification of $k_s/\theta_2$ in the exponential function.

#### 8.4.2.5 Convergence

The convergence property of RL for a single agent has been discussed in [Mitchell, 1997; Bertsekas and Tsitsiklis, 1996]. It was proven that the learned Q values are asymptotically close to the true Q value. Yet, in the multi-agent context, the convergence

property of an agent is dependent on that of other agents. In other words, each individual agent is learning in a non-stationary environment and the multi-stage RL will never converge. However, since the action exploration process is a deliberate decaying process and the process of choosing actions definitely converges to the one with the highest Q values, the multi-stage RL converges. This is true only for the game problem with pure strategy (no mixed actions with probabilities). In this paper, only pure strategy solution is meaningful based on the engineering and business meaning of the game, i.e., a transportation company cannot delegate a *half* task to another company.

### 8.4.3 Fictitious play

#### 8.4.3.1 Fictitious play and its limitation

FP is an evolutionary process in which the rational equilibrium solution of a game can be found. In a repeated game, each agent models the action beliefs of other agents using probability distributions. These probability distributions are based on the occurrence times of the actions of other agents. The actions with a higher number of occurrence times have higher probabilities. Based on these probabilities, also called the belief, the agent can guess the actions of other agents although the guess is not accurate enough. With the guessed actions of other agents, this agent can find her own optimal action during this iteration. Based on the actions chosen by each agent, all the agents in the community update their beliefs about the action preferences toward other agents. As the game repeats many times, the actions chosen by an agent show fixed patterns, which can be considered as the convergence criteria.

Although it is shown that FP converges to the equilibrium solution for some special kinds of games[6] [Garcia et al., 2000; Monderer and Shapley, 1996], it is demonstrated that FP is very sensitive to the given probability distributions for the actions of other agents [Krishna and Sjostron, 1995]. In practice, FP usually fails to converge because many contrary trials are needed in order to change the preference profiles of other agents.

#### 8.4.3.2 Necessities of FP

Two deficiencies exist in seeking an equilibrium solution using RL. First of all, it takes a long time for a reinforced agent to converge. This is due to the failure of modeling the joint Q values. From a long-term view, the Q values obtained for each individual agent are not the Q values when other agents take their best responses, but really the average Q values when other agents evenly chose their possible actions. This phenomenon is very apparent during the early period of the decaying reinforcement learning so that the convergence to the best response actions slows down.

On the other hand, reinforcement learning cannot distinguish rational and irrational actions sometimes. Since a Q value without modeling the joint actions is an aggregate value of all the possible actions of other agents, the distinction between different payoffs cannot be observed on different Q values. This makes the converged actions in reinforcement learning not rational. For example, a useless sell or buy action will be chosen instead of the action to keep silent because the communication cost is not large enough. Of course, each agent is expected to choose a rational action.

[6] Bowling and Veloso [2002] summarized that FP succeeds on the games with two players and dominated pure actions or the games with multiple players and identical payoffs.

Since the FP provides a way for an agent to model other agents in the community and so that her action can be decided deterministically, the incorporation of FP into the reinforcement learning can overcome the two difficulties described above. To sum up, FP will bring two advantages: accelerating the convergence process by providing a deterministic action exploration, and distinguishing different payoffs between different actions more accurately.

#### 8.4.3.3 Working mechanism of fictitious play

We assume that an agent $A_i$ can observe other agents' actions in playing the multi-stage game. Whenever FP is activated, two things need to be carried out:

1. Guess other agents' actions based on the their belief models. This can be deterministic or stochastic;

2. Solve a myopic optimization problem to find the best response to the current guess of other agents.

At a state $s$ of the agent $A_i$, let $\mathbf{k}^{-i}(s) = \bigcup_{j \neq i} \mathbf{k}_j(s)$ denote the profile of occurrence times of other agents' actions, where $\mathbf{k}_j(s) = \{k_j^1(s), k_j^2(s), ..., k_j^{m+1}(s)\}$ denotes the profile of occurrence times of the agent $A_j$ at the state $s$. A simple way to guess the actions of other agents deterministically is to find the action with the maximum number of occurrence times (see Equation (8.8)), where $a^{guess}(s)$ is the guessed action at the state $s$.

$$a^{guess}(s) = \underset{a_j^l(s), l \in [1, m+1]}{\arg\max} \; k_j^l(s) \tag{8.8}$$

Another way to guess other agents' actions stochastically is based on probability distribution over the set of actions. In the probability distribution, each probability is the ratio of the occurrences times for a specific action over the total number of occurrences throughout all the actions. Such a probability distribution is calculated in Equation (8.9).

$$p_j^l(s) = \frac{k_j^l(s)}{\sum_{l' \in [1,m+1]} k_j^{l'}(s)} \tag{8.9}$$

After other agents' actions are guessed, the agent $A_i$ will solve an optimization problem to find the best action only based on her own self-interest. Let $\bigcup_{j \neq i} \{a^{guess}(s)\}$ denote the set of guessed actions of other agents, and then the best action of agent $A_i$ at state $s$ can be found out in Equation (8.10).

$$a_i^{best}(s) = \underset{a_i^l(s), l \in [1,m+1]}{\arg\max} (r(s, (a_i^l(s), \mathbf{a}_{-i}^{guess})) + V(s')) \tag{8.10}$$

where $s'$ is the next state of $s$ with the joint actions $(a_i^l(s), \mathbf{a}_{guess}^{-i})$, and $V(s')$ is shown as in Equation (8.4.3.3).

$$V(s') = \max_{\forall a_i(s')} q(s, a_i(s'))$$

### 8.4.4 Combining the reinforcement learning and fictitious play

Through analyzing the advantages and the disadvantages of RL and FP, we discern that the integration of them solves the problem of the multi-stage game learning nicely. In the integration, RL maintains the tree-like structure and updates Q values, while FP plays a supplementary role in modeling other agents' action profiles approximately. In

order for FP to function as planed in the game learning, two parameters were designed to control its attendance. First, at the beginning of each round of game playing, a decision variable saying "yes" or "no" is designed to decide whether the FP or the RL scheme is taken. This decision process is based on a random process with a discrete probability distribution, in which the probability to take FP should be kept small such that the FP is only a supplementarily played during the whole learning process. Another parameter is the starting time when FP attends the learning process. FP does not begin too early because other agents' occurrences profiles are still oscillating. On the other hand, FP does not begin too late because it may not be necessary due to enough knowledge obtained from the reinforcement learning during the previous periods. Thus, this parameter needs to be defined carefully at a moderate level. With these two parameters, we develop detailed algorithms in Section 8.4.5.

### 8.4.5 Algorithms

In the distributed multi-stage game, the monitor agent and each transportation company agent make decisions using two different algorithms respectively. The algorithm residing in the monitor agent is to coordinate the learning process. This algorithm explores states and updates the levels of the states in a tree-like hierarchy (see Algorithm 8.1). The learning algorithm for each transportation company agent is to explore actions and to update the Q table with the given state and reward from the monitor. Algorithm 8.2 presents how the learning process is being carried on for a transportation company agent.

In Algorithm 8.1, Line 1 is based on Equation (8.6) dealing with how to explore a state. Line 2 is based on Equation (8.5) dealing with how the state levels are updated.

**Algorithm 8.1**: The algorithm to explore states and update state hierarchies in a multi-stage game

```
    Add s_0 into the state collection S;
    while forever do
1       Explore a state s from S and broadcast it;
        while forever do
            Wait messages from all transportation company agents;
            switch message do
                case new state s'
                    Add s' to S;
                    Improve the hierarchy of s by one;
                end
                case old state s'
2                   Update the hierarchy of s;
                end
                case ending message
                    Start next round of learning;
                end
            end
        end
    end
```

In Algorithm 8.2, Line 1 is based on Equation (8.8) or (8.9) dealing with how to guess the actions of other agents. Line 2 is based on Equation (8.10) dealing with how to obtain the best response given the actions of other agents. Line 3 is based on Equation (8.7) dealing with how to explore an action if a state is given. Line 4 is based on Equation (8.1) and (8.2) dealing with how the reward and state transition are obtained. Line 5 and 6 are based on Equation (8.3) dealing with how to update the Q values of each action.

**Algorithm 8.2**: Learning algorithm for transportation company agents in a multi-stage game

**input** : Starting state $s_0$.
**output**: The hash table $\Gamma$ containing decision knowledge.

*Add $s_0$ and all its possible actions into $\Gamma$*;
**while** *forever* **do**
  *Obtain a state $s$ with exploration scheme from the monitor agent*;
  **if** *the scheme is fictitious play* **then**
1   *Guess the actions of other agents*;
2   *Find the best action $a$*;
  **else**
3   *Explore a action $a$*;
  **end**
4  *Obtain $s'$ and $r$ while playing the game*;
  **if** *$s'$ is new* **then**
    *Add $s'$ and its possible actions to $\Gamma$*;
5   *Update the $Q$ value of $a$ with $r$ and zero $V(s')$*;
    *Report new $s'$ to the monitor agent*;
  **else**
6   *Update the $Q$ value of $a$ with $r$ and $V(s')$*;
    *Report old $s'$ to the monitor*;
  **end**
  *Report an message of ending one round of learning to the monitor agent*;
**end**

## 8.5 Computational experiments

### 8.5.1 Test problem

In order to test our multi-stage game learning algorithm, we formulate a simple problem of task delegations in a supply chain procurement problem. In this problem, there are three transportation company agents, and each agent holds a task at an extremely high cost[7] at the beginning of the game. The simplified costs for each agent to carry out each task are shown in Table 8.1. The insertion cost and the deletion cost are the same for the same match of a task and an agent[8]. A communication cost of 0.1 units is assumed. A communication cost is defined as the cost of sending a message to the monitor. Except for the action of keeping silent, all other actions invoke this cost.

Table 8.1. The cost matrix of agents by tasks

| | $A_1$ | $A_2$ | $A_3$ |
|---|---|---|---|
| $t_1$ | 10.0 | 0.3 | 1.2 |
| $t_2$ | 1.0 | 10.0 | 0.7 |
| $t_3$ | 0.5 | 0.6 | 10.0 |

In the above formulated problem, the length of the string representing the state of the market is 3 and at each state, each agent has 4 possible actions. Each agent is restricted to issue only one action at each state. Based on the issued actions from all the agents, a reward and a possible next state will obtained. At the new transited state, next round of double auction will happen, and this continues until the end of learning. This

[7] This marks the infeasibility of this task.

[8] In the real supply chain procurement problem scenario, a cost value depends on the allocation of tasks in an agent, and usually, the insertion cost and the deletion cost are different for the same match of a task and an agent.

is how a multi-stage game is played. The self-interest of each agent can be described as the summation of the rewards over the rounds of playing starting from the initial state.

At each state, the payoff matrix is a cubic with size of $4 \times 4 \times 4$, which leads to 64 possible plays. The number of stages of the game is unknown and depends on the state transition process of the game. If all the agents decide to keep silent, the game stops, but another round of learning can start from any available state.

The learning algorithm is applied to this multi-stage game. The experiment results show that the probabilities of choosing the best actions for each agent converges after 200 rounds of learning. The convergence history is shown in Figs. 8.2 and 8.3.

For this test problem, the computational results produced by the learning algorithms with fictitious play and without fictitious play are compared. It was discovered that the learning algorithm without fictitious play converges to some solution as well. If the communication cost is not counted, the solution obtained is good. However, this method is not able to distinguish between the decisions of keeping silent and that of submitting trading proposals. The reason was explained in section 8.4.3. The difference is shown in table 8.2 that demonstrates how the agents choose actions to play after learning for a period of enough time.

## 8.6 Conclusion and future work

In this paper, we propose a model of multi-stage game and develop a combinational evolutionary learning method to solve the problem of distributed task delegations among a group of self-interested transportation companies in the context of supply chain procurement. The multi-stage game models the self-interests of the companies. The

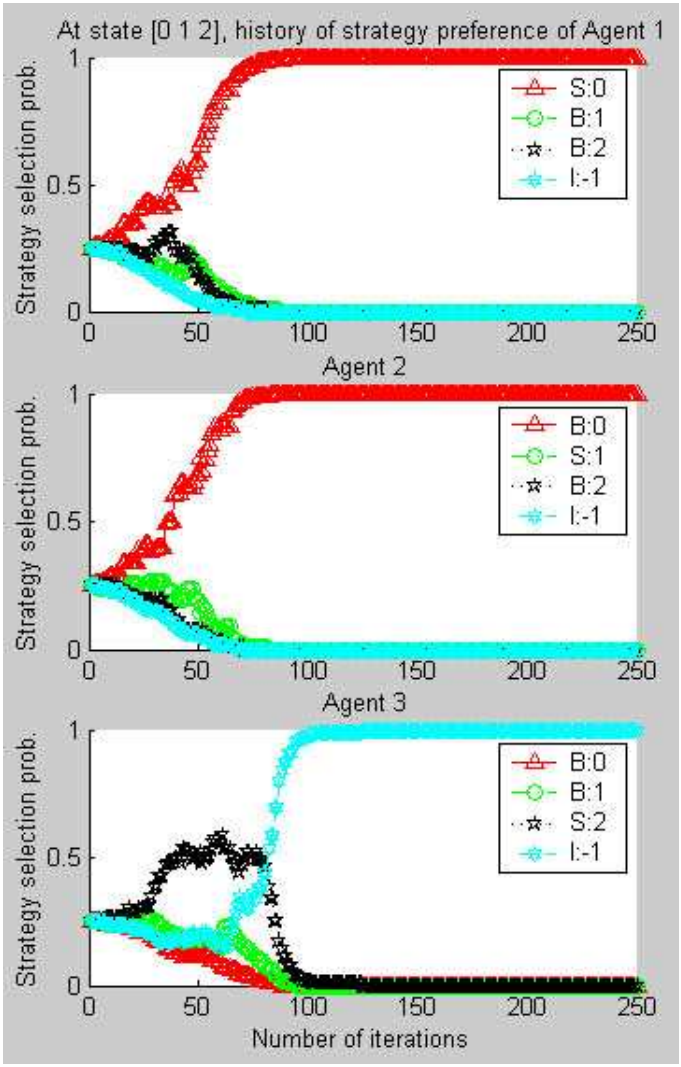


Fig. 8.2. The convergence history of the probability of action selection at state [0 1 2] for all the three agents

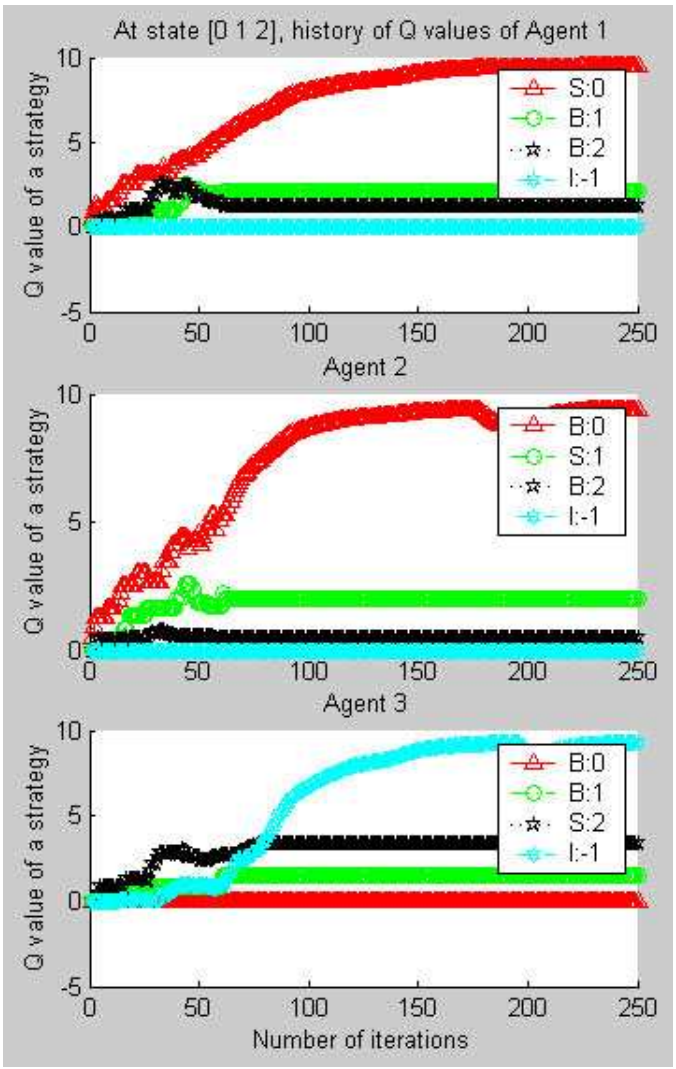


Fig. 8.3. The convergence history of the Q values at state [0 1 2] for all the three agents

Table 8.2. The comparison of action selection after learning between the algorithms with and without fictitious play

| St-age | Age-nt | Play with FP | | Play without FP | |
|---|---|---|---|---|---|
| | | State | Actions | State | Actions |
| 1 | $A_1$ | [0 1 2] | [B:2,r=4.70,q=9.50] | [0 1 2] | [B:2,r=4.70,q=9.50] |
| | $A_2$ | | [I:9,r=0.00,q=9.40] | | [B:0,r=-0.05,q=9.35] |
| | $A_3$ | | [S:2,r=4.70,q=9.30] | | [B:0,r=-0.05,q=9.35] |
| 2 | $A_1$ | [0 1 0] | [I:9,r=0.00,q=4.80] | [0 1 0] | [S:0,r=4.80,q=4.80] |
| | $A_2$ | | [S:1,r=4.60,q=9.40] | | [B:0,r=4.80,q=9.40] |
| | $A_3$ | | [B:1,r=4.60,q=4.60] | | [B:0,r=-0.05,q=4.55] |
| 3 | $A_1$ | [0 2 0] | [S:0,r=4.80,q=4.80] | [1 1 0] | [I:-1,r=0.00,q=0.00] |
| | $A_2$ | | [B:0,r=4.80,q=4.80] | | [S:1,r=4.60,q=4.60] |
| | $A_3$ | | [I:9,r=0.00,q=0.00] | | [B:1,r=4.60,q=4.60] |
| 4 | $A_1$ | [1 2 0] | [I:9,r=0.00,q=0.00] | [1 2 0] | [I:-1,r=0.00,q=0.00] |
| | $A_2$ | | [I:9,r=0.00,q=0.00] | | [I:9,r=0.00,q=0.00] |
| | $A_3$ | | [I:9,r=0.00,q=0.00] | | [I:9,r=0.00,q=0.00] |

combination of reinforcement learning and fictitious play work cooperatively to seek the equilibrium solution based on the concept of evolutionary stable strategy at each stage of the multi-stage game. Numerical results on a small test problem show that the Nash Equilibrium solution for the whole multi-stage game is obtained using this combinational method. This model and approach might also be applied into other B2B procurement problems and optimization problem with multiple objectives by modeling each objective in a self-interested agent.

Upon the future research, first, we can extend the model and approach to large scale problems by using the approximation techniques to store the table of states, actions and Q values. Second, a new game learning method that does not model the state transition in the tabular format will make the solving method for a multi-stage game more flexible. Some approximation methods such as neural networks can be applied. Thirdly, other types of exploration methods on either states or actions, especially deterministic

exploration methods used in the reinforcement learning literature, probably will make the learning process more efficient.

## Bibliography

A. G. Barto, S. J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. Technical Report UM-CS-1993-002, UM, 1993.

P. D. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1996. ISBN 1-886529-10-8.

Darse Billings, Aaron Davidson, Jonathan Schaeffer, and Duane Szafron. The challenge of poker. *Artificial Intelligence*, 134, 2002.

T. Borgers and R. Sarin. Learning through reinforcement and replicator dynamics. *Journal of Economic Theory*, 77(1):1–14, Nov 1997.

M. H. Bowling and M. M. Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250, 2002.

V. Chankong and Y.Y. Haimes. *Multiobjective Decision Making: Theory and Methodology.* North-Holland, 1983.

K. Chatterjee and W. F. Samuelson, editors. *Game Theory and Business Applications*, chapter 6, pages 189–211. Kluwer Academic Publishers, 2001.

C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *AAAI/IAAI*, pages 746–752, 1998.

K. Fischer, J. P. Muller, and M. Pischel. Cooperative transportation scheduling: an application domain for dai. *Journal of Applied Artificial Intelligence (Special Issue on Intelligent Agents - eds M. J. Wooldridge and N. R. Jennings)*, 1995.

D. Fudenberg and D. K. Levine. *The Theory of Learning in Games.* MIT Press series on economic learning and social evolution, 1998.

A. Garcia, D. Reaume, and R. L. Smith. Fictitious play for finding system optimal routings in dynamic traffic networks. *Transportation Research B*, 34:147–156, 2000.

R. Gibbons. *Game Theory For Applied Economists.* Princeton University Press, 1992. ISBN 0-691-00395-5.

L. P. Kaelbling, M. L. Littman, and A. P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

V. Krishna and T. Sjostron. On the convergence of fictitious play. Working paper, 1995.

M. and J. Pet-Edwards. *Making multiple-objective decisions:an executing brief*. unknown, Jan 1997. ISBN 0-8186-7407-5.

Nicolas Meuleau and Paul Bourgine. Exploration of multi-state environments: Local measures and back-propagation of uncertainty. *Machine Learning*, 35(2):117–154, 1999.

T.M. Mitchell. *Machine Learning*. McGra-Hill Companies, 1997.

D. Monderer and L. S. Shapley. Fictitious play property for games with identical interests. *Journal of Economic Theory*, 68(1):258–265, Jan 1996.

M. Mundhe and S. Sen. Evaluating concurrent reinforcement learners. In *Proceeding of the International Conference on Multi-agent Systems*, 2000. to appear as a poster paper.

R. B. Myerson. *Game Theory: Analysis of Conflict*. Harvard University Press, 1997. ISBN 0-674-34116-3.

M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc, 1994. ISBN 0-471-61977-9.

A. Rubinstein. *Modeling Bounded Rationality*. The MIT Press, 1998. ISBN 0-262-18187-8.

L. Samuelson. *Evolutionary Games and Equilibrium Selection*. The MIT Press, 1997. ISBN 0-262-69219-8.

G. Satapathy. *Distributed and Collaborative Logistics Planning and Replanning Under Uncertainty: A Multiagent based approach*. PhD thesis, Industrial and Manufacturing Engineering Department, PA, December 1999.

S. Sen, M. Sekaran, and J. Hale. Learning to coordinate without sharing information. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, pages 426–431, Menlo Park, CA, USA, 31– 4 1994. AAAI Press. ISBN 0-262-61102-3.

O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165–200, 1998.

S. Singh, P. Norvig, and D. Cohn. How to make software agents do the right thing: An introduction to RL, 1996.

D. Sonsino. Learning to learn, pattern recognition and nash equilibrium. *GAMES AND ECONOMIC BEHAVIOR*, 18:286–331, 1997.

R. S. Sutton, A. G. Barto, and R. J. Williams. Reinforcement learning is direct adaptive optimal control. In *Proceedings of the 1991 American Control Conference*, 1991.

K. Tang and S. Kumara. Double auction market mechanism: A distributed negotiation protocol to model an e-procurement problem. *IEEE Automation Science and Engineering*, 2004. accepted.

Kaizhi Tang and Soundar Kumara. Using reinforcement learning to refine the distributed supply chain procurement plan. Technical report, PSU, University Park, Pennsylvania, 2002. Working paper, manuscript completed.

S. B. Thrun. Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, CMU, Pittsburgh, Pennsylvania, 1992.

N. Williams. Stability and long run equilibrium in stochastic fictitious play. Working paper, 2001.

# Vita

Kaizhi Tang is a Ph.D. candidate in the department of Industrial and Manufacturing Engineering at the Pennsylvania State University. His interests are in information systems, artificial intelligence, operations research and macro economics. During his study at Penn State, he led and served as a core member in several research projects: scalable OTD (Order-to-delivery) simulation architecture and shipping yard deployment and load makeup systems (DLMS), funded by General Motors Research Center. He is a member of IIE and INFORMS. Currently, he works as a research scientist at Intelligent Automation, Inc., Rockville, MD, to continue his research. He is married and has a lovely daughter.