AUCTION BASED SCHEDULING FOR DISTRIBUTED SYSTEMS

A THESIS

SUBMITTED TO THE DEPARTMENT OF INDUSTRIAL ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Emrah Zarifoğlu

June, 2005

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. İhsan Sabuncuoğlu (Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Erdal Erel

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Ayşegül Toptal

Approved for the Institute of Engineering and Sciences:

Prof. Dr. Mehmet Baray

Director of the Institute of Engineering and Science

ABSTRACT

AUCTION BASED SCHEDULING FOR DISTRIBUTED SYSTEMS

Emrah Zarifoğlu

M.S. in Industrial Engineering

Supervisor: Prof. Dr. İhsan Sabuncuoğlu

June, 2005

Businesses deal with huge databases over a geographically distributed supply network. When this is combined with scheduling and planning needs, it becomes too difficult to handle. Recently, Fast Consumer Goods sector tends to consolidate their manufacturing facilities on a single supplier serving to a distributed customer network. This decentralized structure causes imperfect information sharing between customers and the supplier. We model this problem as a single machine distributed scheduling problem with job agents representing the customers and the machine agent representing the supplier. For benchmarking purpose, we analyzed the problem under three different scenarios: decentralized utility case (realistic case), centralized utility case, centralized cost case (classical single machine early/tardy problem). We developed Auction Based Algorithm by exploiting the opportunity to use game theoretic approach to solve the problem in the decentralized utility case. We used optimization techniques (Lagrangean Relaxation and Branch-and-Bound) for the centralized cases. Results of our extensive computational experiments indicate that Auction Based Algorithm converges to the upper bound found for the total utility measure.

Keywords: Scheduling, Distributed Scheduling, Decentralized Scheduling, Supply Chain Scheduling, Auction Based Scheduling, Single Machine Scheduling, Auctions, Lagrangean Relaxation

ÖZET

DAĞINIK SİSTEMLER İÇİN İHALE TABANLI ÇİZELGELEME

Emrah Zarifoğlu
Endüstri Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Prof. Dr. İhsan Sabuncuoğlu
Haziran, 2005

İşler coğrafik olarak dağıtılmış tedarik ağları üzerinde yapılmaktadır. Bu durum çizelgeleme ve planlama ihtiyaçları ile birleştiğinde idare edilmesi çok zor bir hal alır. Günümüzde Hızlı Tüketim Maddeleri sektörü üretim olanaklarını dağınık müşteri ağına hizmet veren tek bir tedarikçide toplama yönünde eğilim göstermektedir. Bu merkezi olmayan yapı müşteriler ve tedarikçi arasında mükemmel olmayan bilgi paylaşımına yol açmaktadır. Biz bu problemi işler müşterileri, makine ise tedarikçiyi temsil edecek şekilde bir tek makineli dağıtılmış zaman çizelgelemesi problemi şeklinde modelledik. Bu problemi, karşılaştırmada atıf amaçlı olarak üç senaryo altında tahlil ettik: merkezi olmayan fayda durumu (gerçekçi durum), merkezi fayda durumu, merkezi maliyet durumu (klasik tek makineli erken/geç çizelgeleme problemi). Merkezi olmayan fayda durumunu çözmek için oyun teorisi yaklaşımından faydalanarak İhale Tabanlı Algoritma geliştirdik. Merkezi durumlar için en iyileme tekniklerini (Lagrangean Genişletmesi, Dallandırma ve Sınırlama) kullandık. Yaptığımız kapsamlı ölçümlemeli deneylerin sonuçları İhale Tabanlı Algoritmanın toplam fayda ölçüsü için bulunan üst sınıra yaklaşığını gösterdi.

Anahtar Sözcükler: Çizelgeleme, Dağınık Çizelgeleme, Merkezi Olmayan Çizelgeleme, Tedarik Zinciri Çizelgelemesi, İhale Tabanlı Çizelgeleme, Tek Makineli Çizelgeleme, İhaleler, Lagrangean Genişletmesi

To my family...

Acknowledgement

I would like to express my deepest gratitude to my supervisor Prof. Dr. İhsan Sabuncuoğlu for his instructive comments in the supervision of the thesis and also for all the encouragement and trust during my graduate study.

I would like to express my special thanks and gratitude to Prof. Dr. Erdal Erel and Asst. Prof. Dr. Ayşegül Toptal for showing keen interest to the subject matter, for their remarks, recommendations and accepting to read and review the thesis.

I would like to express my deepest thanks to Prof. Dr. Semih Koray, Asst. Prof. Dr. Alper Şen, Asst. Prof. Dr. Erhan Kutanoğlu, Gökhan Metan, Kürşad Derinkuyu, Banu Yüksel, Selçuk Gören for all their encouragements and supports.

I would like to extend my sincere thanks to Mark Merlino, Mehmet Ferhat Candaş, Zümbül Bulut, Serkan İmişiker, Ilyas Iyoob for their endless morale support and friendship during all my desperate times, makes me to face with all the troubles.

Finally, I would like to express my gratitude to my family for their love, understanding, suggestions and their endless support. I owe so much to my family.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The changing demand patterns and huge amount of data and information in geographically distributed supply networks require new approaches to solve existing and emerging problems in the manufacturing environment. In this complex environment, planning and scheduling becomes main hurdles to achieve. Due to the highly distributed nature of the supply networks, distributed scheduling and planning comes up as a reasonable approach to deal with the supply chain problems.

The inspiration of the distributed scheduling problem considered in this research arises from the Fast Consumer Goods sector. In this sector, there is a tendency to build a central manufacturing plant (supplier) serving to the customers spread over a geographically wide area (Figure 1.1). The Customers give orders to the same plant and compete for the same scarce resources with each other. Each order has its own operational data (release time, process time, due date) and cost structure (inventory and backorder costs). Due to the competitive nature of the customers and geographic distribution, there is no information sharing between the customers. The manufacturing plant has the operational data but it

cannot access cost structure of an order of a customer. This highly distributed structure of the supply network makes the scheduling problem of orders on the manufacturing plant even more complicated than it already is. Attempting to solve this problem with classical centralized scheduling methods does not take the objectives of the customers into account. Therefore, we get help from the recently emerged distributed scheduling techniques to solve this problem.

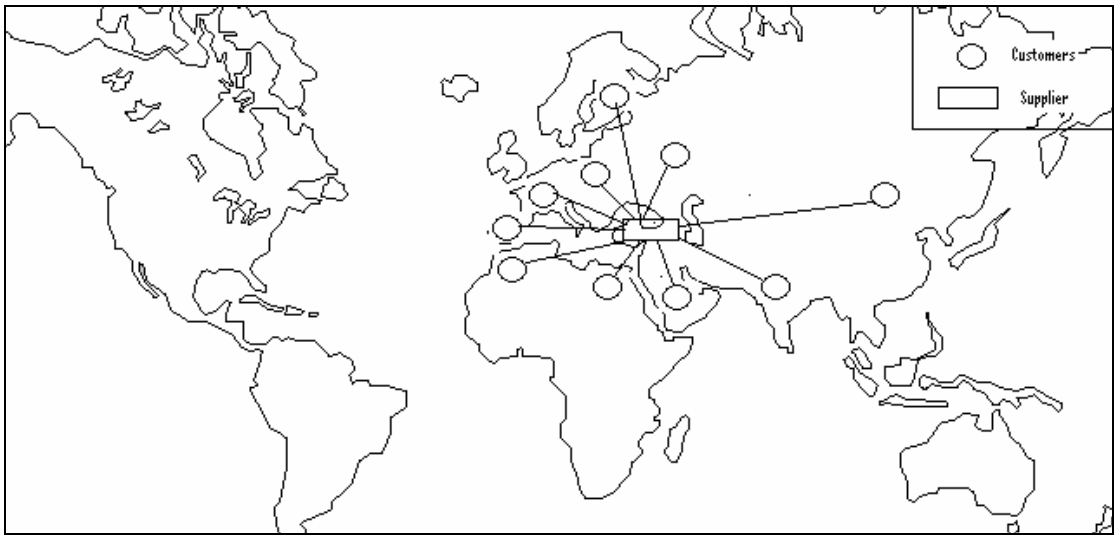


Figure 1.1 A Sample Supply Network

In order to deal with this supply chain problem in simple terms, we can induce it to a single machine scheduling problem. In this context, the manufacturing plant (supplier) is represented by a machine and the customer orders assume the jobs to be processed on the machine. From the centralized viewpoint, the problem can be defined as a classical single machine early/tardy problem. However, due to the decentralized nature of the supply chain case, we study it as a distributed scheduling problem.

We use agent structure to represent the problem environment. The manufacturing plant (supplier) is represented by a machine agent. Without loss of generality, we assume one order for each customer and represent these orders by job agents. Job agents have operational

data (release time, process time, due date) and cost data (earliness cost, tardiness cost). While machine agent can get access to the operational data, it does not know the cost data. We fit a utility function for each job agent using the load profile of the machine agent and cost structure of that job agent. The objective of each job agent is to maximize its utility while it is being scheduled on the machine. The objective of the machine agent is to make sure that there is a feasible schedule.

In order to assess the performance of distributed scheduling paradigm, we will consider three main scenarios. One of them is the centralized cost case. In this one, the problem is a classical single machine early/tardy problem. The objective is to minimize the total early/tardy cost of the jobs. We used Branch-and-Bound to solve this problem. Another one is the centralized utility case in which the only difference is in the objective function that is to maximize total utility of the jobs. We used two different methods to solve this problem, Branch-and-Bound and Lagrangean Relaxation with a feasibility heuristic. The decentralized utility case is the scenario that is compatible with our problem environment explained before. In this case, each job agent acts selfish and tries to maximize its own utility while the machine agent wants to provide a feasible schedule. We developed an Auction Based Algorithm to solve the problem in this case.

The rest of the chapters are organized as follows. Chapter 2 gives a brief literature review about distributed scheduling. Chapter 3 discusses different scenarios explained above and introduces the developed algorithms. In Chapter 4, we present the results of the computational experiments of the algorithms. In the last chapter, we make concluding remarks and discuss the future research opportunities in this field.

CHAPTER 2

Literature Review

In last two decades, distributed scheduling literature has shown a great development. Here, in this chapter, we briefly summarize some of these studies under the three categories: shop floor level scheduling, supply chain scheduling and auction based scheduling as a game theoretic solution approach (Table 2.1).

Table 2.1 A List of selected studies in the relevant literature

Shop Floor Level Scheduling	**Supply Chain Scheduling**	**Game and Auction Based Scheduling**
Tharumarajah and Bemelman (1997)	Sabuncuoglu and Toptal (1999c)	Seredynski (1997)
Sabuncuoglu and Toptal (1999a)	Chen, Cost, Finin, Labrou and Peng (1999)	Kutanoglu and Wu (1999)
Sabuncuoglu and Toptal (1999b)	Sauter and Parunak (1999)	Kaihara (2000)
Sabuncuoglu and Toptal (1999c)	Shen, Chua and Bok (1999)	Wellman, Walsh, Wurman and MacKie-Mason (2000)
Brennan, Norrie, O and Walker (2000)	Tonshoff, Seilonen and Teunis (1999)	Dewan and Joshi (2001)
Khoo, Lee and Yin (2001)	Dutta, Mukherjee and Sen (2001)	Seredynski, Koronacki and Janikow (2001)
Roy and Anciaux (2001)		Kutanoglu and Wu (2002)
Benjamin and Yen (2002)		Grimm, Riedel and Wolfstetter (2003)

Jeong and Leon (2002a) Jeong and Leon (2002b) Najid, Kouiss and Derriche (2003)		

2.1. Shop Floor Level Scheduling:

The most comprehensive survey in this area is due to Sabuncuoglu and Toptal (1999a) distributed scheduling with respect to supply chain management, shop floor level applications and some computer science applications.

In another study Sabuncuoglu and Toptal (1999b and 1999c) propose five different algorithms based on distributed scheduling approach. First three of them give general approaches that can be applied in different job shop environments. The last two are team-based algorithms regarding product teams. The first three are called Algorithm B1, Algorithm B2 and Algorithm C. The last two are called Algorithm PD-JI and Algoithm PD-TI.

Algorithm B1 is an operation initiated process employed in the system in which this algorithm assumes a group of machines each of which has different processing capabilities and processes the jobs having different operations on each other in a visitation sequence. Each local planner (resource agent) has its own objective. There is also a global objective to achieve and the master agent is interested in the sake of this objective. The algorithm has two kinds in one of which resource agents either collaborate with each other or in another one they compete with each other. This system can be classified as a single layer quasi-heterarchical system with a separate manager agent. As stated before this is an operation initiated algorithm. Manager agent first ranks the schedulable operations. Then, manager agent broadcasts a bid to the resource agents for the first schedulable operation. Then the bid preparation process starts for the resource agents. There are both competitive and

collaborative versions. Manager agent selects the best bid according to some predetermined criteria, e.g. earliest finish time.

Algorithm B2 is a machine initiated bidding algorithm. In this algorithm, the resource agents have the initiative to start a bid. Master agent's responsibility is to resolve the conflicts among the resource agents. This system may be classified as a single layer quasi heterarchical system with a separate manager agent. The main idea of the algorithm is that the operations are put into a pool according to some priority assigned by the manager agent.

Algorithm C is a job initiated algorithm. It is very similar to the Algorithm B1 but its difference is that it is based on jobs rather than operations. Three bidding mechanisms are proposed for the Algorithm C, such as H1, H2, H3. If H1 is used, the system behaves like a single layer quasi-heterarchical system with a separate manager agent. If H2 or H3 is used, the system acts like multi-layer quasi-heterarchical system with multiple bids. There are also competitive and collaborative versions of Algorithm C. After the bids are prepared, the manager agent selects the best bid according to some predetermined criteria, e.g. earliest finish time.

Algorithm PD-JI is one of the team-based algorithms proposed by Sabuncuoglu and Toptal (1999c). A team is a group of machines (resources) that are capable for certain processes, projects or products. For the PD-JI algorithm, manager agent ranks the jobs according to some predetermined criteria, such as earliest due date. Then it broadcasts a bid request to the teams. Here, the teams are assumed to process almost all or all of the operations required for the job. This system may be classified as a multi-layer quasi-heterarchical system with a separate manager agent and multiple biddings. The bid preparation mechanism is similar to Algorithm B1. After the bid preparation process, the

manager agent selects the best bid according to some predetermined criteria, such as earliest finishing time.

Algorithm PD-TI is a team initiated algorithm. The similar teams are grouped together according to their processing capabilities and each group having a rank in the expertise determined by the manager agent according to the experience in the past. Also, different job pools are assigned for the different appropriate pools. This system can be classified as a multi-layer quasi-heterarchical system with a separate manager agent and multiple biddings.

There are many examples of algorithms provided as utilizing AI based methods. The AI methods that can be applied to the distributed scheduling algorithms may be Constraint Heuristic Search (CHS), Asynchronous Teams (A-Teams) or Cooperative Interaction via Coupling Agents (CICA). CHS is applied in the multi-agents environments where job-based or resource-based agent formations are used to make scheduling. The disadvantage of CHS is it is difficult to embed this into an optimization process. It finds feasible solutions rather than optimal. A-Teams is another AI-based method. This method works as incorporated agents utilized with problem-solving methods to work together to solve a problem sharing their solutions via common memories. Jeong and Leon (2002a) employ CICA to solve a Distributed Scheduling problem in their recent work. CICA works as establishing interactions among cooperating organizations and coupling agents. Coupling agents are artificial entities utilized with some coupling constraints. Jeong and Leon work on a two-shared-machine problem in a two-machine flowshop environment. The distributed characteristics of the system come from the distribution decision authorities and information among multiple sub-production systems sharing two machines. A coupling agent is used to store one of the shared machine's information and another coupling agent is used to store the other machine's information. The system which Jeong and Leon study can be classified as single layer quasi-

heterarchical without a separate manager agent. The problem to be solved is to allocate operations of jobs to time slots on machines to achieve global objectives by interaction of sub-production systems and shared machines with a minimum sharing of global information for sub-production systems and coupling agents. 0-1 integer programming formulation is used to model scheduling problems and Lagrangian relaxation technique is employed for the solution process. The main aim of the algorithm is to find a compromise state where all coupling constraints and local constraints are satisfied and the total sum of weighted completion time of jobs is minimized (Jeong and Leon 2002a). Good coordination is provided as a result of the experimentation. The algorithm gives very close solutions to the global optimum.

Jeong and Leon (2002b) make similar work in a single machine environment. This system also consists of multiple sub-production systems and these sub-production systems share the single machine. The distributive characteristics come from the distribution of the authority of decision making among sub-production systems. The system also can be classified as single layer quasi-heterarchical without a separate manager agent. The aim is same as before such that they want to minimize the weighted sum of the completion times. As before, CICA is used to utilize coupling agents. Sub-production systems and shared machines are assumed to have not whole global information. Therefore, as in the previous work of Jeong and Leon, they use Lagrangian relaxation to solve a 0-1 integer programming formulation. Lagrangian relaxation is appropriate for the cases that there is not enough global information publicly open to the agents. This method also performed quite well for the single machine case.

Brennan, Norrie, O and Walker (2000) develope an algorithm to make dynamic job routing and job sequencing decisions. They found that the composition of reactive agent

mechanisms and appropriate job sequencing heuristics perform well in the shop floor where job congestion increases.

Khoo, Lee and Yin (2001) study an agent-based architecture for scheduling multiple shop floors using a genetic algorithm-enhanced scheduling engine (2001). The manufacturing scheduling server and shop scheduling client system are two main modules. The supervisory agent is coordinating the among the shop floor agents to arrive at a global near optimal solution and to resolve conflicts in the shop floor schedules. The algorithm generates feasible and near optimal schedule for the entire manufacturing system in the experiments made in a hypothetical six products-three shop floors and a plastic injection molding company.

Roy and Anciaux (2001) propose an approach to solve dynamic production control problems in real time to automate the control process as much as possible, to adapt the system to production plan modifications and to rationalize decision making by means of strong hierarchical structure. A twofold hybrid multi-agent platform is used for this purpose. Control is hierarchically distributed and decision making is centralized. Centralization helps avoid from competition between agents and hierarchical distribution allows each agent to take care of only one product. By this way, they gain significantly in terms of response times and reactive capabilities.

Tharumarajah and Bemelman (1997) review negotiation and the emerging behavior-based methods for scheduling and coordinating distributed entities within both hierarchical and heterarchical control structures. In the paper, they emphasize issues of practical importance relevant to a distributed shop floor environment.

Benjamin and Yen (2002) present a communication infrastructure to handle connection and communication between distributed Internet scheduling systems for

distributed applications. They present an agent communication language, syntax and semantics for the agent communication languages, and negotiation mechanism.

Najid, Kouiss and Derriche (2003) present an application of the multi-agent approach to the control of a flexible manufacturing cell. It is a physically distributed system on a set of sites and is composed of a set of cognitive and reactive agents that coordinate their tasks to carryout the dynamic control and the scheduling of the manufacturing system. The coordination of the actions of the agents emerges from the interaction of these combined agents.

2.2. Supply Chain Scheduling:

There are not many studies in the literature for the applications of the distributed scheduling approaches in the supply chain (Sabuncuoglu and Toptal 1999b). There are some studies that employ multi-agent structure in the supply chain or improve the information sharing systems. Some representative examples of these studies are given below.

Sabuncuoglu and Toptal (1999c) propose two team-based algorithms one is being job initiated and the other is being team-initiated as we explained before. They present these algorithms in the context of shop floor scheduling but they emphasize that these algorithms can be also used for supply chain management with some changes.

Chen, Cost, Finin, Labrou and Peng (1999) propose a multi agent-system to model the supply-chain management problem in the real business life environment using software agents. They use the concept of negotiating agent to model the self-interested entities in the market place. The system framework they designed allows negotiating agents to join, to stay or to leave the system freely. This is not a distributed scheduling problem but it gives idea about how to design a multi-agent system based on negotiating agents and how the negotiation among agents can be made.

For the planning reasons in supply chain, Shen, Chua and Bok (1999) present a distributed scheduling tool, called the Integrated Production Scheduler implementing look-ahead planning via Integrated Constraints Modeling. It is being used for planning the activities in the supply chain. A three-layered structural model (database layer, look-ahead planning layer and planning layer) for the Integrated Production Scheduler was presented. The roles of distributed systems in the supply chain are automatic messaging, internet publishing and distributed collaboration. This implementation is done via Internet technology.

One of the studies in the decentralized production environment (e.g., supply chains) is due to Tonshoff, Seilonen and Teunis (1999). Their work proposes a mediator based approach to support decentralized decision-making focusing on the communication, negotiation and scheduling process. The mediator is designed for an adequate level of decision-making integration of heterogeneous computer system by use of the Extended Mark-Up Language (XML).

Dutta, Mukherjee and Sen (2001) use general scheduling idea to provide the competitive power to the firms in supply chain. This is an idea of usage of other scheduling techniques than distributed scheduling also. Manufacturer announces contracts for tasks with given specifications (deadline and processing time). Suppliers bid on these tasks with prices. Contract is allocated by an auction to a supplier who fulfills all task constraints. A three level supply chain with primary and secondary manufacturers and a group of suppliers is studied. Ability of different strategies to produce more flexible schedules is analyzed. Also, an analysis of a price adjustment mechanism by which suppliers make up their bids before when they win contracts and reduce their bids when they fail to procure them, is made. This study uses responsive supply chain to identify scheduling policies that benefit suppliers in

managing profitability by allowing them to accommodate demands that others cannot. Supply chain managers should manage flow of materials from distributed suppliers to global manufacturing facilities. Emphasis should be on keeping low inventory, minimizing operations cost, having flexibility and providing efficient customer service. The motivation of the study is the aspect of a supply chain as being responsive to dynamically arriving tasks which is hinged upon the performance of the suppliers. Scheduling strategies are first fit (put the incoming task to the first possible available place), best fit (put the incoming task to the place such as the right and left slacks are minimized), or worst fit (put the incoming task to the place such as the right and left slacks are maximized). By using the tardiness of the task in the supply chain, manufacturers may offer different prices. If tardiness is low they offer high price, if it is low they offer high price. They use scheduling for this trade-off business.

Sauter and Parunak (1999) present ANTS as an example of a Distributed Scheduling application in the supply chain management. In supply chain management, solving the problem only in the Original Equipment Manufacturer (OEM) level just delays the problem to the other levels. Especially, the problems in sub-tier suppliers remain as very important problems. Some problems in supply chain may show itself as schedule variation in sub-tier suppliers, similar capacity bottlenecks at multiple suppliers, deviation of inventory or WIP levels from expectation. In supply chain systems, coordination with the shop floor level is a problem. Information systems such as MRP, MRP II, ERP are either limited or too complex to solve the stated problem. Smaller firms in the supply chain are too difficult to integrate to the whole system. Regarding these problems, Agent Network for Task Scheduling (ANTS) aims to provide supply chain management software that can handle complex dynamical systems while being much simpler to construct and manage. It proposes solution as a small-grained agent-based system. A small-grained agent is a simple agent that responds to its

environment using simple rules or interacts directly with other agents through predetermined protocols. These agents are inspired from the rules and the interactions that govern the insect colonies. In a manufacturing enterprise, main measurements are cost of goods produced, quality of goods produced and timing of availability of goods relative to the customer's needs. The task of the supply chain management is to deploy resources across a supply chain to produce high quality goods as inexpensively as possible and when the customer wants them. Regarding the things stated before, the decisions taken by the supply chain management are to select which suppliers for which product, order of products to be manufactured, start time of new jobs, time of new orders and inventory level. The specific requirements in a supply chain are least commitment, empowerment, frequent change, MRP functionality, metamorphosis, modality emergence, uniformity. In ANTS, the agents are the "things" in the supply chain and within the factory.

2.3. Game and Auction-Based Scheduling:

Wellman, Walsh, Wurman and MacKie-Mason (2000) sum up auction protocols used in distributed scheduling. They investigate the existence of equilibrium prices for some scheduling problems, the quality of equilibrium solutions and the behavior of an ascending auction mechanism and bidding protocol. They also discuss direct revelation mechanisms and compare them to market-based approach. They define price equilibrium as each agent getting an allocation that maximizes its utility given the current prices. The common structure for auction protocols is composed of three stages. Firstly, agents send bids to the auction mechanism indicating their willingness to exchange goods. Secondly, the auction may post price quotes to provide information about price-determination process. The first and second steps may be iterated. At the last step, the auction determines an allocation and informs agents about the allocation. Wellman et al. define ascending auction as a mechanism at which

agents are sending higher bids at each time to the mechanism. While the ascending auction is performing well for single-unit problems, it may not be the case for the multiple-unit problems and it may not find a price equilibrium even if it exists. Therefore combinatorial auction mechanisms are proposed. In these mechanisms, allocations and prices are considered regarding function of bids for all the combinations. Prices may refer to individual goods or to entire bundles. Also, this paper deals with the generalized Vickrey auction as a direct revelation mechanism. The generalized Vickrey auction computes overall payments for agents' allocations that sometimes translate into meaningful prices for individual goods. As can be seen, this study gives a good summary for the three categories of mechanisms as a spectrum, i.e. single-good, combinatorial and direct revelation.

Kutanoglu and Wu (1999) study auction-based scheduling on a combinatorial context using Lagrangean relaxation. They are interested in mechanisms that allow resource scheduling to be locally autonomous, and at the same time aligned with global interests. They consider the classical jobshop scheduling problem where a set of jobs is to be completed and each job requires a set of machines for a certain period of time for processing. They propose an iterative auction mechanism for this problem using the notion of multi-item combinatorial auction. An auctioneer sells discrete time slots (objects) to the bidders (jobs). Based on current pricing, each job gives bid to best combination of the time slots trying to maximize its utility function. The auctioneer evaluated bids and updates the reservation prices according to the conflicts among the jobs. This process goes iteratively until a conflict-free allocation is found. They investigated two auction protocols (non-adaptive Walrasian and adaptive tatonnements) and two payment functions (regular and augmented tatonnements). They show that Lagrangean relaxation using subgradient search corresponds to an adaptive regular

tatonnement. They demonstrate the prices of time slots depend heavily on the demand patterns (routing structures and processing times).

Kutanoglu and Wu (2002) study collaborative resource planning that arises when resource managers must coordinate their planning with internal or external customers. They analyze a setting where the decision makers are geographically distributed and coordinate their resource planning using asynchronous, web-based mechanisms. They design a schedule selection game where all participating agents state their preferences via a valuation scheme and the mechanism selects a final schedule based on collective input. In this mechanism, assuming the agents may not state their true evaluation of the schedules; the mechanism offers incentive to the agents. By this way, dominant strategy equilibrium exists, i.e. each player plays the strategy that is individually best for him regardless f the strategies chosen buy other players. Based on Vickrey-Groves-Clarke principles, they show that the proposed mechanism is a direct revelation mechanism that implements the optimal resource allocation under agents' dominant strategies. They show a numerical example of this mechanism in coordinating electronics component manufacturing.

Dewan and Joshi (2001) present a scheduling model similar to a combinatorial auction with mathematical programming tools used for bid construction and evaluation to solve a problem in a dynamic job shop environment. Each entity in the shop is represented by a process interacting with other processes over the network. Dewan and Joshi states that the computing benefits of distributed implementation can be realized despite network delays.

Seredynski (1997) proposes an approach based on considering a given system as a multi-agent system with game-theoretic models of interaction between players. Players compete to maximize their payoff and the global objective is represented global behavior of the team of players, measured by the average payoff received by the system. Three

distributed schemes (i.e., ε-learning automata, loosely coupled genetic algorithms and loosely coupled classifier systems) are used to evolve a global behavior in the system. Simulation results indicate that the global behavior in the systems emerges and is achieved in particular by only a local cooperation between players acting without global information about the system. He applies models of multi-agent systems to develop parallel and distributed algorithms of dynamic mapping and scheduling tasks in parallel computers.

Seredynski, Koronacki and Janikow (2001) propose a distributed approach in which the agents are associated with individual tasks of the program graph to scheduling of parallel and distributed algorithms for multiprocessor systems. Agents play an iterated game to find directions of migration in the system graph with the objective of minimizing the total execution time of the program in a given multiprocessor topology. Competitive coevolutionary genetic algorithm (i.e., loosely coupled genetic algorithm) is used to implement the multi-agent system. The algorithm with the local criteria is able to find optimal or near optimal solutions in a number of generations comparable with the number required by the algorithm with the global criterion.

The study of Grimm, Riedel and Wolfstetter (2003) is an example for multi-unit auctions. Their study analyzes the second-generation (GSM) spectrum auction as an example of a low price outcome in a simultaneous ascending-bid multi-unit auction. They show that in the unique equilibrium that survives iterated elimination of dominated strategies, the efficient allocation is reached at minimum bids.

Kaihara (2000) proposes an agent-based double auction algorithm and demonstrate the applicability of economic analysis to this framework. He uses product allocation problem in supply chain as a case study. The algorithm is proved to emerge sophisticated product flows in supply chain, and conduct a Pareto optimal solution on multi-objective problems.

CHAPTER 3

Proposed Methodology: Auction Based Algorithm for Decentralized Scheduling

3.1. Problem

The recent tendency, consolidating all manufacturing needs of any certain division, of Fast Consuming Goods sector inspired us the factory scheduling problem over a supply chain. The orders given by geographically distributed customers may share the same resource. The inventory cost arising from early completion of an order and the backorder cost caused from late completion of an order are among main issues for each customer and also whole supply chain. In such supply networks, each customer takes care of its own objectives (minimizing cost or maximizing utility). The supplier tries to obtain a feasible schedule by making negotiations with customers. The distributed structure of the problem environment makes it harder to solve.

This supply chain scheduling problem can be viewed as a single machine scheduling problem in a highly decentralized structure. We developed three different scenarios by changing degree of centralization to analyze the problem environment.

The scenario with decentralized case (decentralized utility) simulates the actual supply chain environment. The problem structure that we are working on has a highly decentralized environment. We use agents to represent the supplier (machine) and the customers (jobs). We fit a utility function for each job agent which aims to maximize its own utility. The operational data (release time, process time, due date) of the job agent is private to the job itself, it is not known by other jobs. However, the machine agent knows the release time, due date and processing time information of the job agents. The cost related data are earliness and tardiness costs. Each job agent has its own earliness and tardiness cost information. This information is not visible to other job agents and also not visible by the machine agent. This is a reasonable assumption since the cost or finance information is not usually transparent to others.

The centralized utility scenario ignores the decentralization, gives permission for the machine agent to access all data (also the utilities) of the job agents so it assumes the environment as a centralized case. We again work with utility functions. The objective is to maximize the total utility of job agents while finding a feasible schedule. An integer programming approach takes care of the modeling requirements. This second scenario is used for benchmarking purpose. It is the case whose optimization solution provides the upper bound on the utility.

The scenario of centralized cost corresponds to a centralized case. The overall optimization of the system is concerned. Hence, the objective becomes the minimization of total early/tardy cost of job agents. This is a classical single machine scheduling problem.

Moving from a single machine early/tardy problem to a distributed scheduling problem with utility functions is the main difference of this study from classical single machine literature. In centralized cost scenario, a job agent has a convex cost function as can

be seen in Figure 3.1. In centralized cost case, there is no competition among job agents. There is no concept of collaboration because there is a manager (master) agent that solves the problem for the sake of whole system. We used Branch-and-Bound to solve this problem.

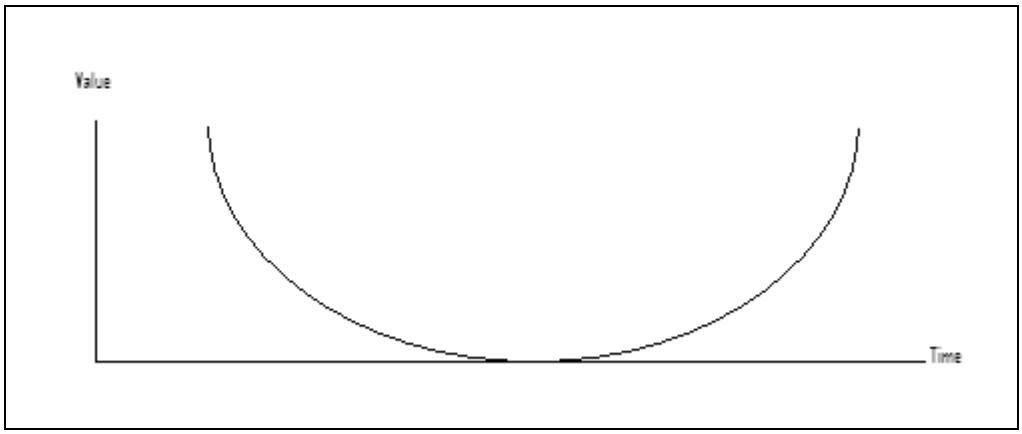


Figure 3.1 Early/Tardy Cost Function of a Job Agent

In the centralized utility scenario, the cost function of each job agent in Figure 3.1 is multiplied by -1 (Figure 3.2) to move the function to the positive side by adding a large enough positive constant. We discretize the time axis into time slots. Given the price scheme of the time slots, the utility function of each time slot is found by subtracting the prices from the converted early/tardy cost function (Figure 3.3). The details of assigning prices to time slots and finding a utility function are explained in the next section. The objective in this case is to maximize the total utility of the jobs. Note that a solution that optimizes total utility is different from the total cost case. Because the patterns of cost functions and utility functions are different. Hence, they can generate different solutions. In this case, there is forced collaboration, and no competition. The forced collaboration is also provided by a dictator agent. We used Branch-and-Bound to solve this problem. We also developed a Lagrangean Relaxation Algorithm with a feasibility heuristic for this problem.

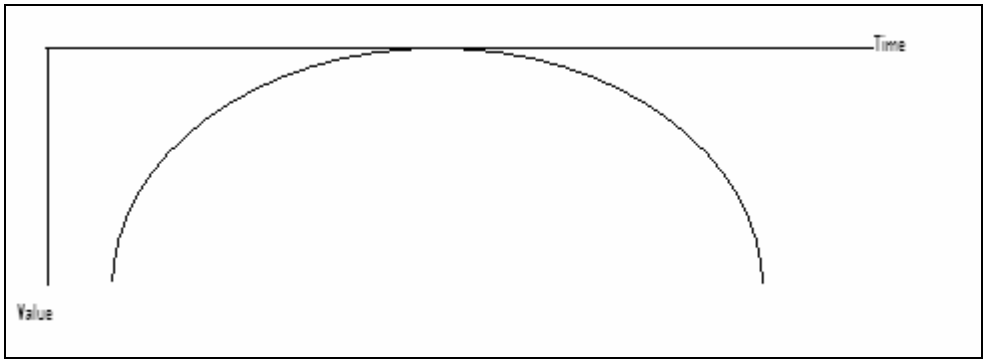


Figure 3.2 Negative Early/Tardy Cost Function of a Job Agent

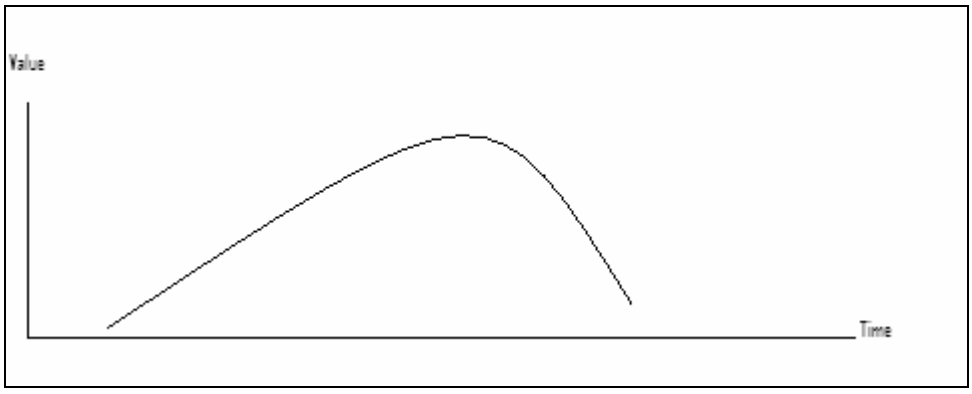


Figure 3.3 Utility Function of a Job Agent

The decentralized utility scenario is in fact the case we want to solve for the actual supply chain scheduling problem. Each job agent tries to maximize its own utility. The objective of the machine agent is to have a feasible schedule. There is not a master (or central) agent in this case. We see competition between job agents with no collaboration. The machine agent may artificially provide some collaboration between job agents by negotiation. We propose an Auction Based Algorithm to solve this highly decentralized problem.

The main difference between centralized cases and decentralized case is in the information sharing. In the centralized cases, the machine agent acts as a central agent who has access all data of job agents and the machine agent aims the good of the overall system. However, in the decentralized case, the machine agent has access to only operational data (release time,

due date and process time). It does not have access to cost data (unit earliness cost, unit tardiness cost). The jobs do not share any information among each other. Hence, there is not a central agent working for the good of the overall system. Therefore centralized case gives a bound for the performance of overall system. Figure 3.4 depicts the structure of the problem environment in agent perspective.

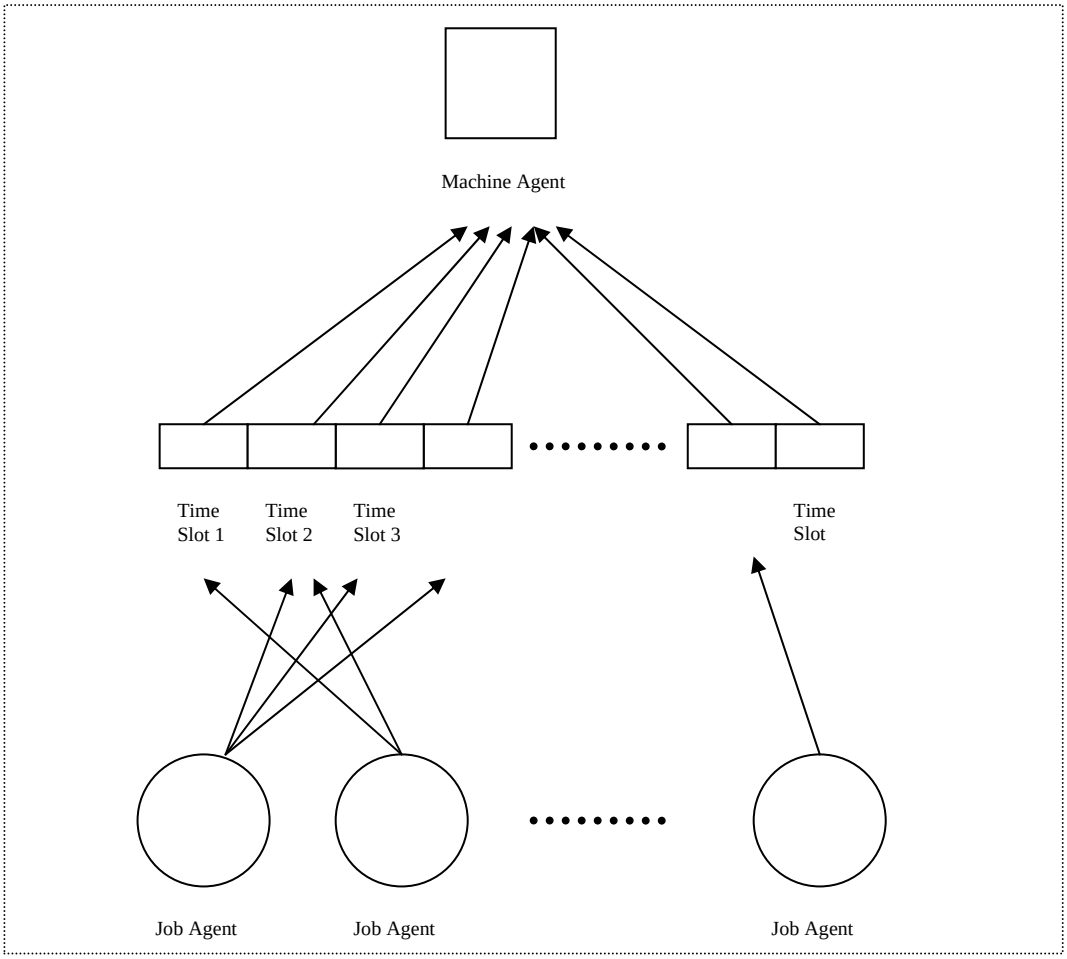


Figure 3.4 Machine Agent, Job Agents and Time Slots

3.2. Decentralized Utility (Auction Based Algorithm)

First we give a brief introduction to auction theory. Then we present the mechanism of Auction Based Algorithm. A numerical example of the algorithm is given at the end of this section.

3.2.1. Auction Theory

Auction can be defined as a market institution with an explicit set of rules determining resource allocation and prices on the basis of bids from the market participants. Auctions are widely used in the markets to sell goods and to determine prices for those goods. It is one of the oldest ways of selling goods. Auctions are usually used in the markets in which seller does not have the ability to estimate or determine price of goods. Because the seller cannot determine price by itself, market employs some mechanism consisting of some rules to determine the price and to assign the goods to the demanding customers.

The main issues of auctioning can be classified into: i) auctioning mechanism, ii) number of units put to the auction at once, and iii) number of stages at which the auction is ended.

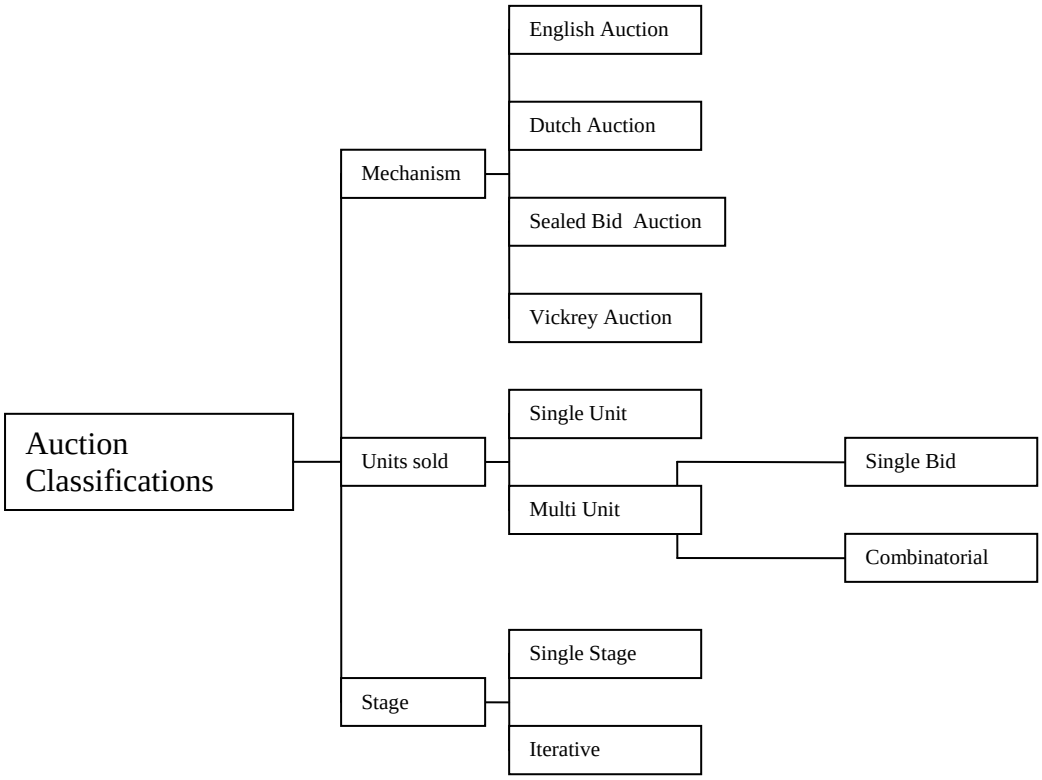


Figure 3.5 Auction classifications

3.2.1.1 Auctioning Mechanisms

In the literature, there are four commonly referred auctioning mechanisms: *English auction*, *Dutch auction*, S*ealed bid auction*, and *Vickrey auction*.

The *English auction* is similar to the ones found in some antique goods outcry auctions. It is based on sequential bidding. Bids are made by bidders for the good until no one remains in the bidding process. The winner is the last bidder and buys the good for the price it offers. *English auction* can be implemented in such a way that the bid is increased by a constant amount or as much as bidders want. The seller usually sets a reservation price (or minimum price) below which it cannot be sold.

The *Dutch auction* is originated from the flower market in Netherlands. It is still used worldwide in flower markets. In the Dutch auction, the seller sets a price (it is usually more than the value of the good) and then he (or she) decreases the price sequentially. The auction stops whenever a bidder raises his (or her) hand to stop the bidding process. The good is awarded to the bidder with that price.

Sealed bid auction is implemented by bidders offering their bids secret from the others to the seller. The seller chooses the bidder with the highest bid. The highest bid's value is assigned as the price of the good. This type of action can be seen in most of governmental contracts.

Vickrey auction is similar to sealed bid auction. Again, the seller chooses the bidder with the highest price. However, the value of the second highest bid is assigned as the price of the good. The motivation for *Vickrey auction* is to give incentive to bidders to tell their true evaluation for the good in the auction. By paying the second highest bid, the winner makes an advantage by paying less than his estimated amount. The difference between the

winner bid and the second highest bid is the incentive for the winner. Thus, we can call the sealed bid auction as first-price sealed bid auction and the Vickrey auction as second-price sealed bid auction.

Note that, English auction is equivalent to Vickrey auction and Dutch auction is equivalent to sealed-bid-auction. Recall that in Vickrey auction, the winner pays the second best price. In English auction, however the winner also pays the second best price plus a prespecified amount of increment to win the bid. Similar relation can be found between Dutch auction and sealed-bid-auction. In both cases, the winner pays the highest price.

3.2.1.2. Classification of Auctions

Figure 3.5 gives an overview about the classification of auctions. These classifications are discussed next.

3.2.1.2.1. Classification based on the Number of Units in Auction

An auction can be classified into *single unit*, or *multi unit*. In the *single unit auctions*, bidders prepare bids for only one item at a time. In the *multi unit auctions*, bidders give bids for more than one items at a time. A *multi unit auction* can be realized in the form of either *single bid* or *combinatorial auction*. In *single bid auctions*, the bidder offers one bid for each item that he is interested among a whole set of goods available in the auction. In the *combinatorial auctions*, the bidder gives one bid for the combination of all the items that he wants to buy.

For example, consider three different pictures to be sold in a sealed-bid-auction and there are two bidders (customers). The seller puts these three different pictures into auction at the same time (i.e., a *multi unit auction* case). Bidder 1 is interested in buying pictures 1 and 2. Bidder 2 wants all the pictures. In the *single bid auction*, the seller asks for the bids from the bidders as one distinct bid for each picture. Bidder 1 bids $300 for picture 1 and $250 for

picture 2. Bidder 2 bids $250 for picture 1, $300 for picture 2, $150 for picture 3. At the end of first iteration (i.e. collecting bids from the bidders at one stage) the seller gives picture 1 to bidder1, pictures 2 and 3 to bidder 2. In the combinatorial auction case, the seller wants bids from the bidders for the whole combination of the pictures they are interested in buying. Bidder 1 bids $550 for the combination of pictures 1 and 2. Bidder 2 bids $700 for the combination of pictures 1, 2 and 3. The seller assigns all three pictures to bidder 2 because bidder 2 offers more money as a total amount.

3.2.1.2.2. Classification based on the Number of Stages at Auction

The auctions are also classified as either *single stage* auctions or *iterative* auctions. In the *single stage auctions*, the auction is finalized at the end of one single stage (or iteration) after taking bids from the bidders. In the *iterative auctions*, auction is implemented in a number of iterations. Each iteration has its own characteristics by the means of auction methods utilized.

In the previous example, now consider a *single bid* in a *sealed-bid-auction*. Bidder 1 bids $300 for picture 1 and $250 for picture 2. Bidder 2 bids $300 for picture 1, $300 for picture 2 and $150 for picture 3. The seller assigns picture 2 and picture 3 to bidder 2. Since the seller can not decide on picture 1, he decides to go on to the second iteration. This time, he opens an English outcry auction with the constant increment of $10. Bidder 1 increments bid to $310. Bidder 2 responds by a bid of $320 and so on. At the end, after bidder 2 bids $390, bidder 1 bids $400. Bidder 2 does not respond to bidder 1 therefore the picture goes to bidder 1 with the price of $400. The auction finishes at second iteration. At the end bidder 1 gets picture 1, and bidder 2 gets picture 2 and picture 3.

3.2.2. Distributed Scheduling Algorithm Based on Auction Theory:

In this subsection, we will explain the proposed scheduling algorithm based on auction theory. After we give the necessary notation and related background, we present the steps of the algorithm by using an example.

Distributed Scheduling has lots of opportunities for the contribution of different fields of science and technology. Artificial intelligence is one of these fields that also makes a contribution to the scheduling by using agents. While there seems to be no single, formal definition for an agent, Karsai et al. (2000) consider them as sophisticated objects that are dynamically created, and through communication, cooperation and competition solve complex problems. In Distributed Scheduling applications, scheduling problems are decomposed to subproblems. Objectives of these subproblems become the objectives of agents whose decision making abilities are bounded by the constraints of the subproblems.

The introduction of agent structure to the solution of scheduling problems gives us the opportunity to elaborate on rational agents and to attach rational human characteristics to them. Interpreting behaviors of these rational agents by employing utility functions to them helps we use economic –especially microeconomic- analysis. In the scheduling context, the rational agent competing for scarce resources can be subject to game theoretic analysis (as a part of microeconomics). Due to the communicative, cooperative and competitive properties of agents, market mechanisms can be employed to model scheduling problems in a distributed manner as the agents being actors in the market setting. Auctions, part of market mechanisms, can also be used in modeling the behaviors of agents in a scheduling problem. In recent years these approaches have become popular in distributed scheduling.

After we reduce our problem from a supply chain scheduling problem to a single machine distributed scheduling problem, we employ an auction theoretic mechanism to solve

this problem. Preemption is not allowed. The setting is such that there is a single machine and multiple jobs are waiting to be processed on this machine. Each job has two sets of information data. One set consists of operational data and the other set consists of cost related data. Operational data are ready (or release) time, due date and processing time.

We formulate the problem such that there are job agents representing jobs and the machine agent representing the machine. The operational data is private to the job itself, it is not known by other jobs. However, the machine agent can get access to the release time, due date and processing time information of the job agents. The cost related data are earliness and tardiness costs. Each job agent has its own earliness and tardiness cost information. This information is not visible to other job agents and also not visible by the machine agent.

In summary, there is not any information sharing among job agents, and there is partial information sharing between job agents and the machine agent. Therefore this is an imperfect information sharing case. Because there is not perfect information sharing and there is not a central authority collecting all the information to make decision for the good of whole system, this is a distributed system and it is highly decentralized.

We employ an auction mechanism to schedule the jobs on the machine. The machine agent is seller and the jobs are bidders (buyers). Assuming discrete time if we divide the planning horizon into equal parts, we acquire equal-length, discrete time slots. These time slots on the machine represent units to be sold in the auction.

In the distributed setting of the problem that we developed an algorithm to solve, the objective of job agents is to maximize their own utilities and the objective of the machine agent is to find a feasible schedule taking the preferences of the job agents into account.

3.2.2.1. Auction Mechanism

All the time slots (or items) are identical and they are put to the auction at the same time. Thus, we have a *multi-item auction* environment. Reservation prices are set for each item by the machine agent. Reservation price is a limit for a bidder's bid to be accepted. The seller determines a minimum bid price and accepts bids higher than this price. Also, the bidders are expected to give one unique bid for a combination of the items they want to buy (i.e. *combinatorial auction*).

The *combinatorial auction* setting is a result of the assumption that preemption is not allowed. In this case, job agent is interested in a bundle of time slots in which these time slots reside next to each other. This bundle makes sense for the job agent as a whole instead of distinct time slots in that bundle. Thus, the job agent evaluates whole bundle with a unique price which leads us to *combinatorial auction*.

In this setting, it is almost impossible to assign the time slots to the bidders at one stage. The seller needs an iterative procedure to solve the conflicts among the bidders. The *English auction* mechanism is iterative in its nature. Therefore we use *English auction*.

The reservation prices are set by the seller (the machine agent) by using the load information obtained from the bidders (job agents). Each bidder sends its information in the form of due date, release time and processing time. The seller determines the ideal time interval for each bidder regardless of the other bidders. Then the seller uses all this information to determine a load profile (tendency graph) for the time slots (items to be sold). This profile shows the relation between time slots and the demand on each time slot. The seller converts this tendency graph to reservation price graph. This new graph shows the reservation price of each time slot. The conversion is made by multiplying the load profile by the value of increment of English auction.

The structure of the algorithm is such that the machine agent first takes the required information (operational information) from the job agents to determine the reservation prices, then it asks the bids from the job agents. The job agents give bids according to the rules that will be explained in the next section. The machine agent asks bids from the job agents in the order of their indices. Then the next pass starts and the machine agent again asks bids from the job agents. Each passage from one job agent to the next job agent for bid request is called a iteration. Each tour or pass consisting of asking bids from all job agents is called a cycle. These cycles and iterations are repeated a number of times until the stopping criteria for the auction are met. At each iteration, the machine agent revises the prices of the time slots. If some jobs are not scheduled after a realization of an auction, the same steps above are repeated for the empty time slots and unscheduled jobs. It means a new auction is opened to unscheduled jobs for empty time slots. Previously scheduled jobs keep their places and they do not enter the new auction. These auctions are repeated until all jobs are scheduled.

3.2.2.1.1Behavior of Bidders

The bidder briefly tries to fulfill its local objectives. In general, each bidder acts selfishly and tries to maximize its benefit (i.e. tries to get e time slot at which its processing requirement is satisfied on time without incurring any earliness or tardiness penalty). Economic interpretation is that the bidder has a utility function and its rational behavior is to maximize this utility function. This utility function is revised at each iteration. The utility function is used to determine the bidder's strategy in the auction by the bidder. The utility function has two components. The first component is the evaluation of the time slots by the job agent. The second component is the actual price of the time slots taken from the seller.

The first component is easy to compute. The bidder calculates this value using its earliness and tardiness information. The bidder applies its earliness and tardiness costs to all

the time slots as if the job starts being processed at each time slot. Application of the earliness and tardiness costs to all the time slots gives the bidder a discretized convex curve showing its actual evaluation of the time slots. Then the bidder multiplies this curve by -1 to determine the valuation. The second component that is the price of the time slots is taken from the machine agent at each iteration.

The difference between the valuation of the time slots and the price gives the bidder's utility function depending on which time slot it starts processing. In addition to these terms, we search all the early/tardy cost values of the job agents we previously calculated. The maximum of these costs is taken and added to all the utility functions for convenience. This assures us all the utility functions have positive values at the beginning of the auction. This also helps us to set convenient stopping criteria as explained below. The bidder determines the time slots on which this utility function has its maximum value. The bidder chooses as many time slots as its processing time as its target time slots and gives one unique bid for the combination of all these time slots.

This auction mechanism works with any utility function independent of its shape. It can be observed throughout the iteration. Because the shape of the utility function changes constantly in the auction process and it takes different shapes randomly.

There are three factors considered in the bidding process. These factors are summed up by the bidder and form the value of the bid. The first factor is the addition of the prices of the utility maximizer (target) time slots. The second factor is more difficult to compute. When a bidder gives bid for a combination of time slots, some of these time slots could previously have been assigned to other job agents. If these job agents also possess other time slots than the target time slots, the previous assignment to these time slots are cancelled because preemption is not allowed. Therefore, some time slots other than the target time slots

given bid by the current job agents will be affected by this current bid. Their assignments will be cancelled, they will be empty so their prices will decrease to their reservation prices. This is a loss for the machine agent. Therefore, the current job agent who is giving bid compensates this loss. Thus, the second factor consists of the sum of the differences between the previously assigned prices to the affected time slots and their reservation prices. The third factor is the increment value of the English auction. These three factors are added and give the value of the bid for the target time slots.

The utility function of the seller is the sum of the prices assigned to the time slots at that iteration. If a time slot is not assigned to any of the jobs, its price is assigned as the reservation price of that time slot at the previous iteration.

3.2.2.1.2. Stopping Criteria

Stopping criteria are explained for two cases, auctions and algorithm.

An auction stops when none of the bidders give bid in a cycle. A job agent does not give bid if one of two conditions holds. First condition is that a job agent possesses the utility maximizing time slots already at its hand when its turn comes to give bid. Since it already has the utility maximizing time slots, it does not want to change its place or it does not give higher price for those time slots. Therefore it does not give bid. Second condition occurs when all points of a job agent's utility function take negative values. One of the two conditions is guaranteed for all the job agents because prices are increasing in an auction. If there are not any other agents, who have higher values of utility functions, giving bid to a job agent's utility maximizing time slots, then that job agent buys those time slots and do not give away. Or, because of increasing prices (both actual and reservation prices if necessary), some job agents who have small values of utility functions have negative values of utility functions after some iterations. Since these job agents do not have any time slots and they do

not give any bids, they are excluded from the bidding process (i.e. game). This kind of exclusion results with not being scheduled on the machine. In this case a new auction is opened to the unscheduled jobs for the empty time slots and same rules are valid for the subsequent auctions. They have the same stopping criteria as the auction at the beginning.

The algorithm stops when all jobs are scheduled. The time complexity of the problem is $O(nm^2)$.

3.2.3. Algorithm: English Auction Based Scheduling

The algorithm consists of five parts. The main part is called Algorithm MainAuction. The other four parts (or subroutines) support the main algorithm.

The first subroutine, Algorithm GenerateReservationPrices, is used by the machine agent to determine the reservation prices before the auction. This algorithm uses due-date and processing time information of the job agents as input and generates reservation prices of the time slots as output. This output is sent to Algorithm GenerateUtilityFunction.

Each job agent invokes the second subroutine (Algorithm GenerateUtilityFunction) to find its utility function before starting bidding. Inputs of Algorithm GenerateUtilityFunction are earliness and tardiness costs, release time, due-date and processing time of each job agent, and reservation prices coming from the machine agent as an output of Algorithm GenerateReservationPrices. The output of GenerateUtilityFunction is utility function of each job agent. This output is sent to the third subroutine called Algorithm PrepareBid.

Algorithm PrepareBid is used to prepare a bid by each job agent. The output of Algorithm GenerateUtilityFunction is used to set initial value of utility function. This utility function is revised at each iteration. The output of Algorithm PrepareBid consists of time slots currently under bid and the value offered for these time slots by the job agent. This output is sent to the fourth subroutine (Algorithm DeterminePrices).

The MachineAgent employs Algorithm DeterminePrices to determine the new price of the time slots at each iteration. The inputs of this algorithm are the old prices (previous reservation prices and actual prices) and the output of the Algorithm PrepareBid (the bid given by a job agent to the machine agent at that iteration). The output of this algorithm is the new prices and reservation prices of the time slots for the next iteration. This output is used to revise their utility functions by the job agents.

Algorithm MainAuction works by the interaction of these four subroutines. The logical flow chart of the algorithm can be seen in the Figure 3.6.

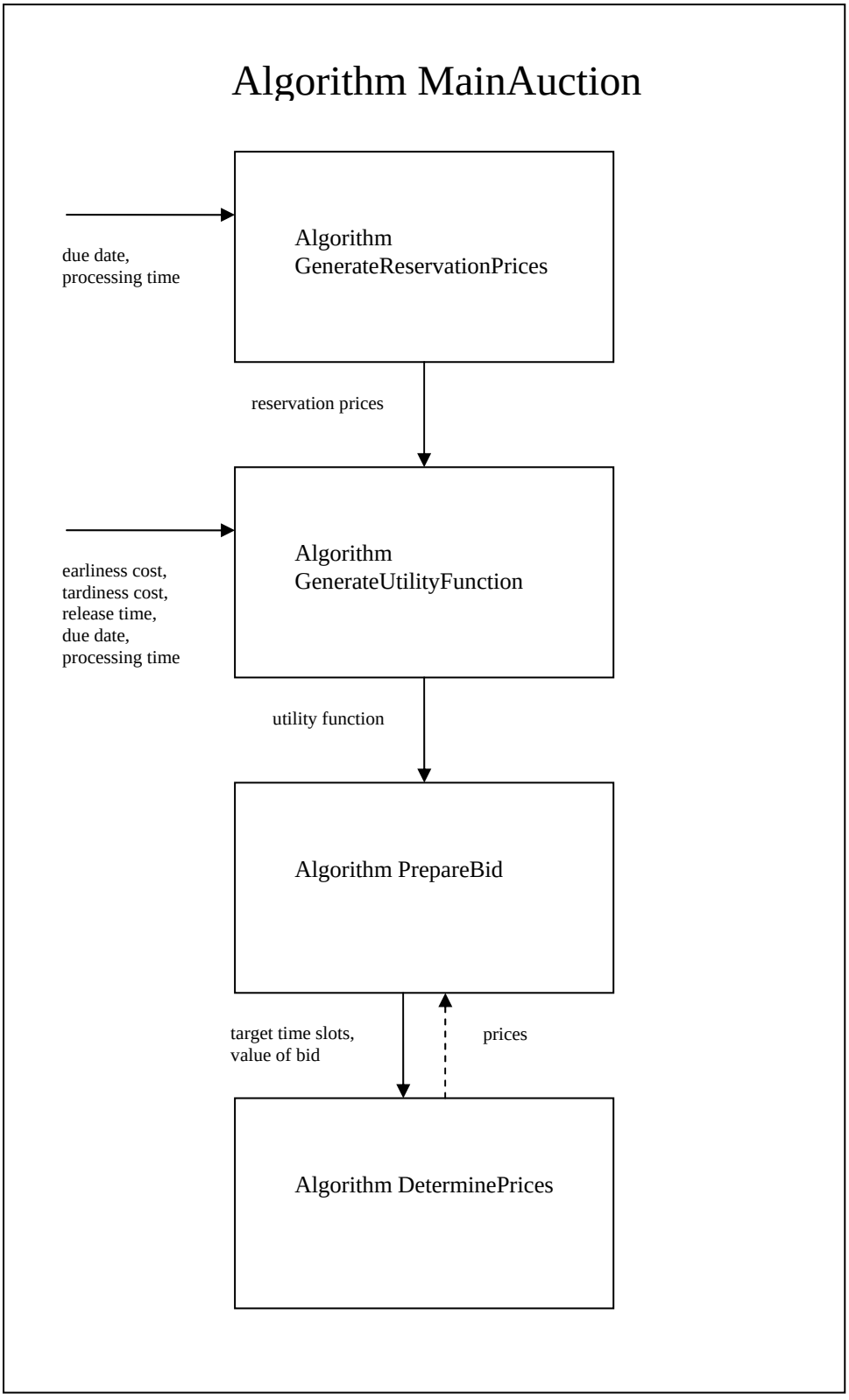


Figure 3.6 Logical flow chart of the algorithm

Algorithm GenerateReservationPrices:

Step 1: The machine agent requires the available time information, due date information and the processing time information from all the job agents (r_j, d_j, p_j).

Step 2: The job agents give the required information (r_j, d_j, p_j).

Step 3: The machine determines the most demanded time slots for each job agent. These time slots are denoted by $i : d_j - p_j + 1, d_j - p_j + 2, ..., d_j$

Step 4: The machine agent forms a tendency graph (load profile) using the information of most demanded time slots. It simply adds up the demands on each time slot.

Step 5: The machine agent converts tendency graph to the reservation price list for the time slots. It applies this by multiplying the tendency graph values by the value of the increment in the English auction.

Step 6: The machine agent opens the time slots on itself to the auction and announces the reservation prices for these time slots to the job agents.

Algorithm GenerateUtilityFunction:

Step 1: Each job agent calculates its earliness/tardiness cost depending on each time slot.

Step 2: The values found in step 1 are multiplied by (-1).

Step 3: Find the minimum among all the values found in Step 2.

Step 4: Add the absolute value of the term found in Step 3 to the values found in Step 2 (Normalization of the utility functions of the job agents).

Step 5: Subtract the sum of the prices of as many time slots as the processing time of each job agent starting from the time slot being considered.

Algorithm PrepareBid:

Step 1: Job agent finds the time slot combination that maximizes its utility.

Step 2.1: If the job agent already has the time slot combination that maximizes its utility, or the maximum utility it has is negative, it does not give any bid.

Step 2.2: Else it gives bid to the time slot combination that maximizes its utility (if there are various combinations of time slots maximizing the utility, the job agent chooses the time slots combinations having the largest value of total indices) by adding the increment value of the English auction to summation of the declared prices of the time slots. The time slots, whose status change from busy to idle, are also added to this calculation as opportunity cost.

Algorithm DeterminePrices:

Step 1: The machine gets the bid from the job agent j and this bid becomes the total price of the target time slots.

Step 2.1: If a time slot does not get a bid and its status is not affected its price does not change.

Step 2.2: If a time slot does not get a bid but its status changes from busy to idle, its price becomes its reservation price.

Step 2.3: If a time slot gets a bid, the sum of the prices of such time slots becomes the whole price given by the bid minus the summation of the prices of the time slots explained in Step 2.1 and Step 2.2.

Step 2.3.1: If there is only one time slot as explained in Step 2.3, it gets the price calculated in Step 2.3

Step 2.3.2: If there are multiple time slots as explained in Step 2.3, they equally share this price.

Step 2.4: If a job agent leaves its assigned time slots to give bid for some other time slot combination, the current prices of these time slots are assigned as their reservation price.

Algorithm MainAuction:

Step 1: The machine agent employs Algorithm **GenerateReservationPrices**

Step 2: The job agents employ Algorithm **GenerateUtilityFunction**

Step 3: Set j = 1

Step 4: The machine agent wants bid from the job agent j.

Step 5: The job agent employs Algorithm **PrepareBid**

Step 6: The machine agent employs Algorithm **DeterminePrices**

Step 7: All job agents revise their utility functions according to the new prices.

Step 8: if j < n, Set j = j + 1, Go to Step 4

else if no job agent gives bids in one cycle, STOP, else Go to step 3

Algorithm MainAuction is repeated if there are unscheduled jobs until all jobs are scheduled. The algorithm runs over empty time slots as unscheduled jobs being the bidders. Previously scheduled jobs keep their time slots. Their time slots and they do not attend the next realizations of Algorithm MainAuction.

3.2.4. Numerical Example

Suppose that there are two jobs waiting to be scheduled on a single machine. These jobs are named as Job 1 and Job 2. The necessary data of these jobs are in Table 3.1:

Table 3.1 Data set of Jobs

	r	d	p	e	t
Job 1	0	3	2	3	3
Job 2	1	2	1	6	3

r: release time

d: due date

p: processing time

e: earliness cost per unit time

t: tardiness cost per unit time

As seen in Figure 3.7a, suppose that the planning horizon (timespan) is between time 0 and time 5. The following is the names of the time slots in the planning horizon:

Time slot between 0 and 1: Time Slot 1

Time slot between 1 and 2: Time Slot 2

Time slot between 2 and 3: Time Slot 3

Time slot between 3 and 4: Time Slot 4

Time slot between 4 and 5: Time Slot 5

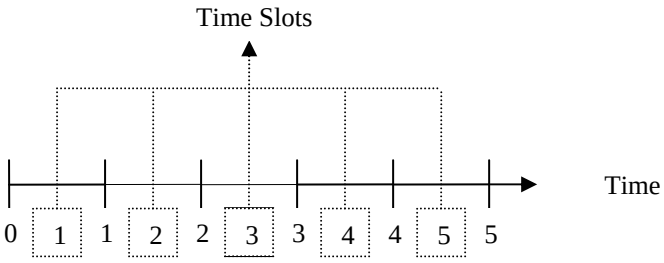


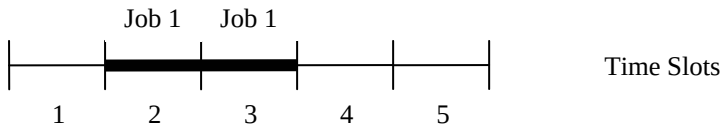
Figure 3.7a The names of the time slots

(i) Application of Algorithm GenerateReservationPrices:

The machine agent opens its time slots to the auction. First, it requests the operational information from the job agents. The job agents give the following necessary data:

Table 3.2 Operational Data of Jobs

	r	d	p
Job 1	0	3	2
Job 2	1	2	1



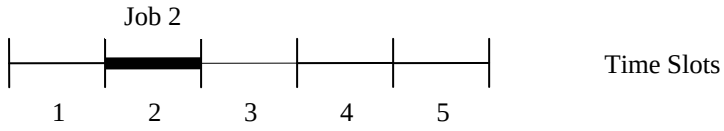


Figure 3.7b Demand information of Job 1 and Job 2

The machine agent uses this information to generate a load profile:

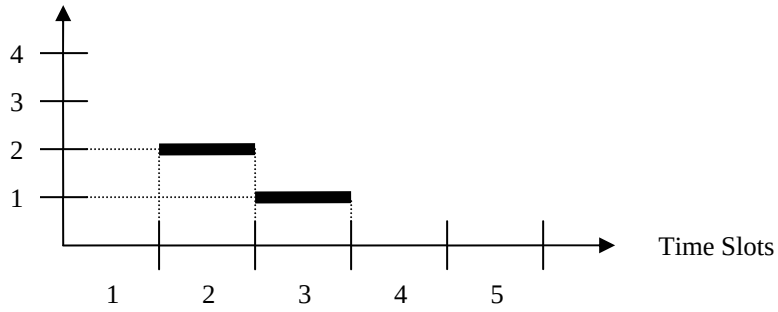


Figure 3.7c Load profile

Then, the machine agent multiplies load profile values by the increment value of the auction, i.e. $1, to compute the reservation prices.

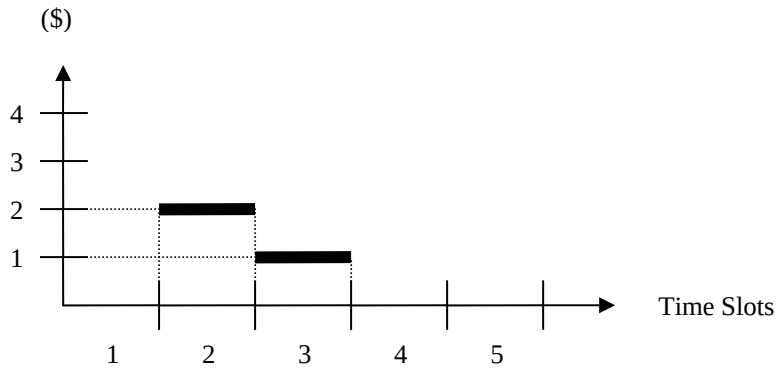


Figure 3.7d Reservation prices of the time slots

(ii) Aplication of Algorithm GenerateUtilityFunction:

The job agents firstly calculate their earliness/tardiness costs depending on which time slot the jobs start to be processed.

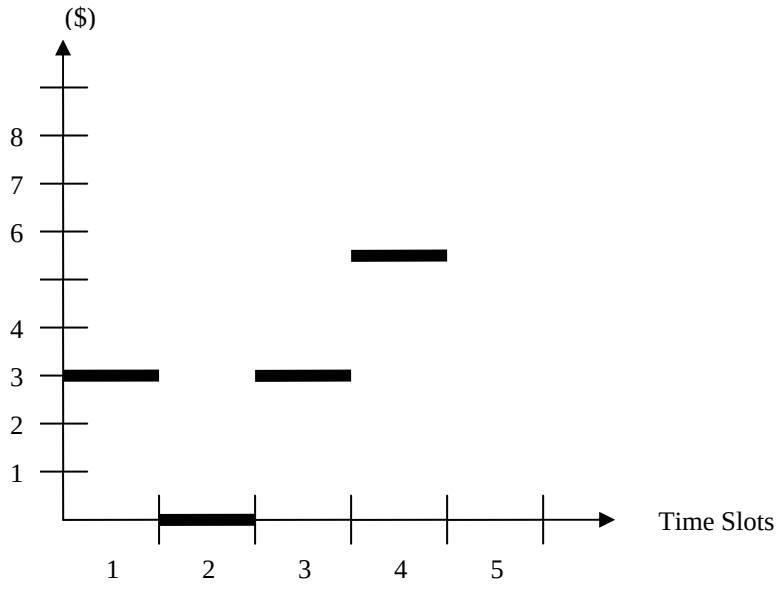


Figure 3.7e Earliness/tardiness cost function of Job 1

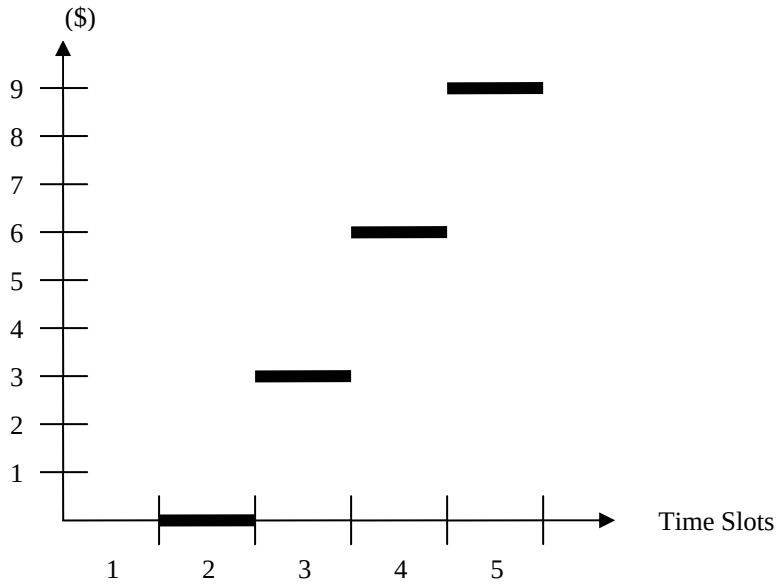


Figure 3.7f Earliness/tardiness cost function of Job 2

Then, job agents multiply these cost function by -1:

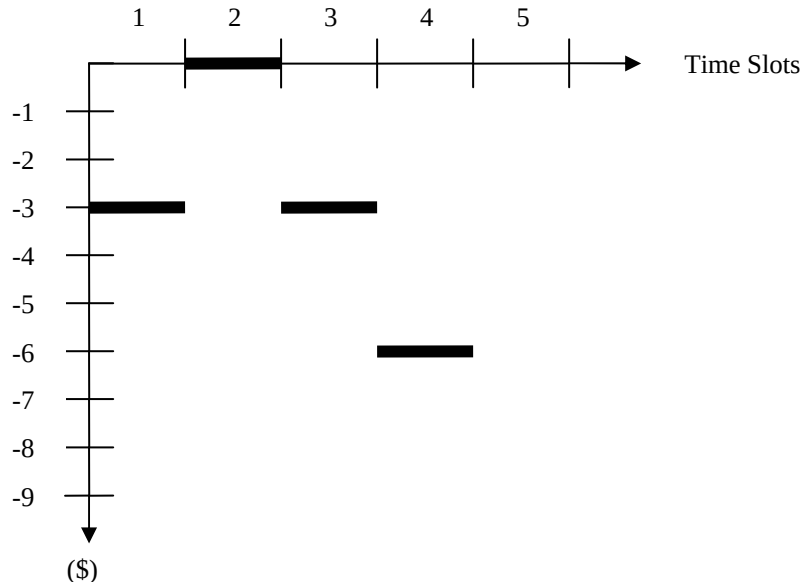


Figure 3.7g Negative earliness/tardiness cost function of Job 1

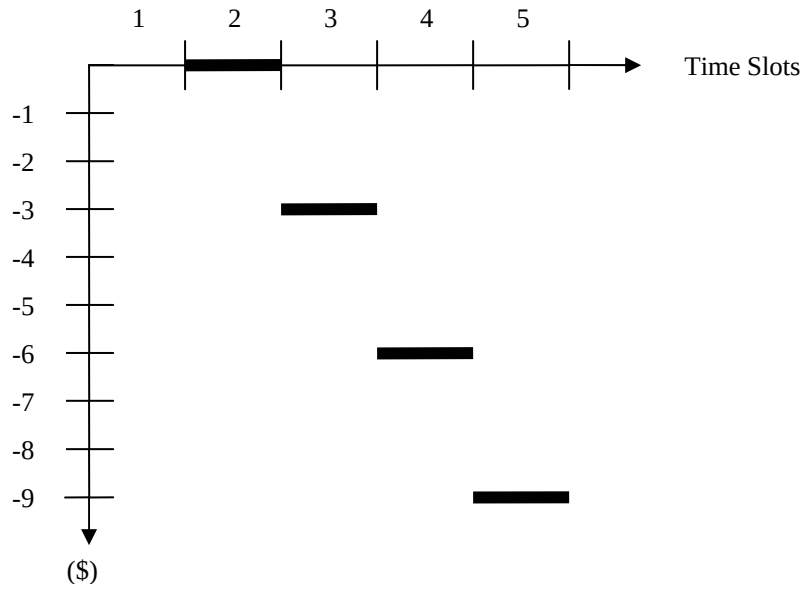


Figure 3.7h Negative earliness/tardiness cost function of Job 2

The maximum earliness/tardiness costs is 9. Therefore, the job agents add 9 to their negative earliness/tardiness cost functions for the normalization.

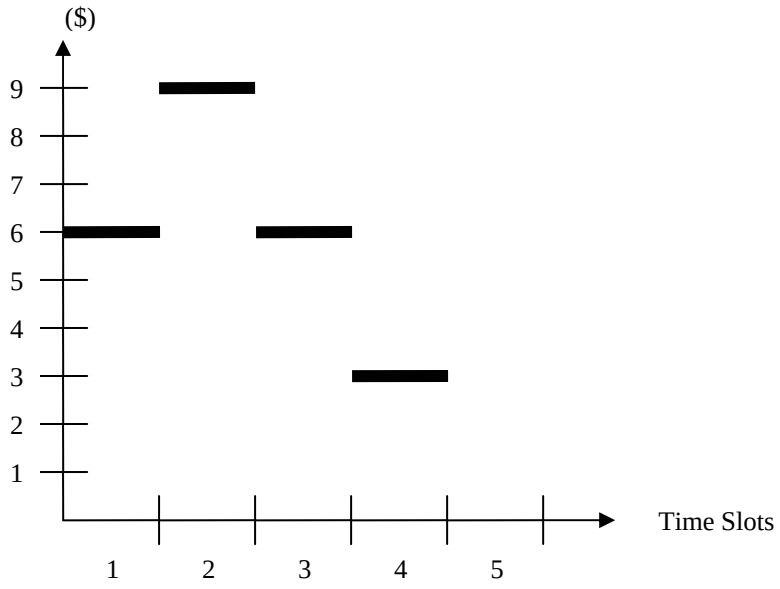


Figure 3.7i Normalized negative earliness/tardiness cost function of Job 1

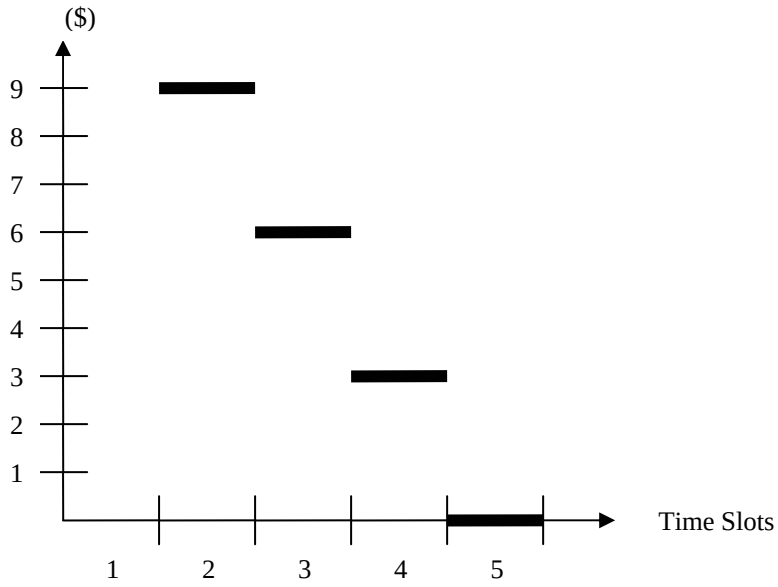


Figure 3.7j Normalized negative earliness/tardiness cost function of Job 2

After finding the first component of the utility function, the job agents subtract the prices of the time slots from the function found above. Job 2 computes directly subtracting the reservation prices from the normalized negative earliness/tardiness cost function values because its processing time is 1. However, Job 1 has processing time of 2. Therefore, it firstly adds the values of the related time slot and the next time slot, then subtracts this value from its normalized negative earliness/tardiness cost function values. This last step gives job agents their utility functions at the beginning of the auction.

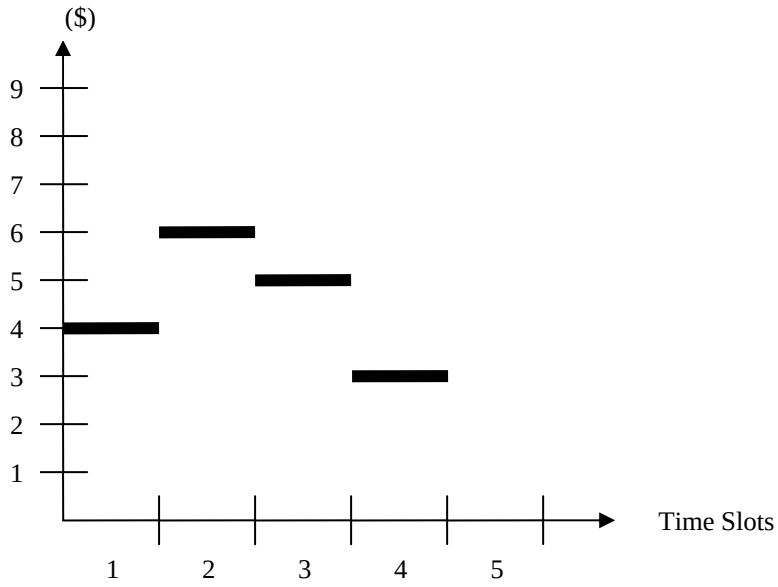


Figure 3.7k Utility function of Job 1 at the beginning of the auction

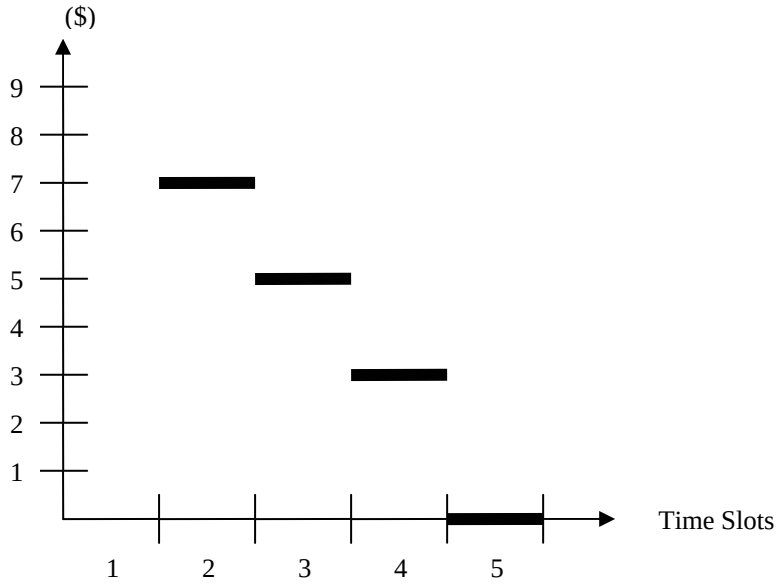


Figure 3.7l Utility function of Job 2 at the beginning of the auction

(iii) Application of Algorithm PrepareBid and Algorithm DeterminePrices:

The machine agent asks a bid from Job 1. Starting at Time Slot 2 maximizes Job 1's utility. Therefore, it gives bid of $4 to Time Slot 2 and 3 (price of the Time Slot 2 + price of the Time Slot 3 + the increment value = 2+1+1).

The machine agent assigns Time Slots 2 and 3 to the Job 1 and revises the prices of the time slots according to this bid. The values of the time slots already add up to $3, there is an additional $1. The machine agent divides this $1 among Time Slots 2 and 3. Now, the prices are as the following:

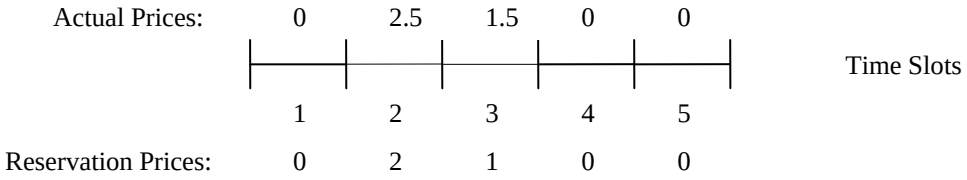


Figure 3.7m Prices of the time slots after the first iteration

Before starting the next iteration, Job 2 revizes its utility function to find the new utility maximizer time slot.

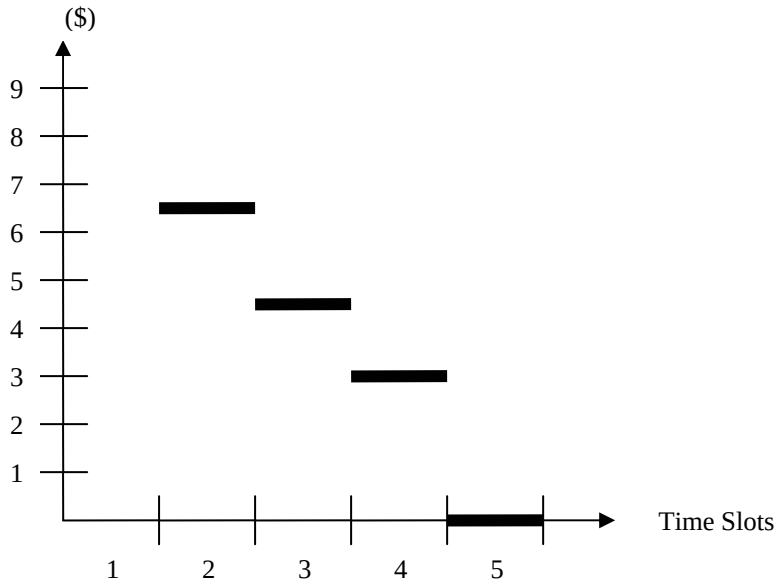


Figure 3.7n Utility function of Job 2 after first iteration

Time Slot 2 maximizes Job 2's utility. However, it was assigned to Job 1 in the previous iteration. Therefore, Job 2 takes the compensation of the loss of the machine agent due to assignment cancellation of Time Slot 3 from Job 1 in addition to the price of Time Slot 2 and the increment value of the auction. Job 2 gives a bid of $4 for Time Slot 2 (price of Time Slot 2 + compensation of the loss due to Time Slot 3 + increment value of the auction = 2.5 + (1.5 -1) +1).

Since this bid has come only for Time Slot 2, the machine agents assigns this value of $4 to Time Slot 2. Because Time Slot 3 is empty now, its value decreases to its reservation price ($1). New prices are as follows:

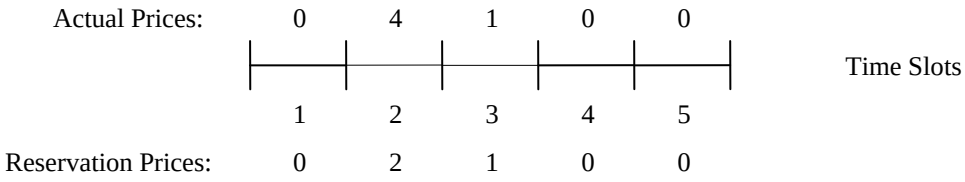


Figure 3.7o Prices of the time slots after the second iteration

Job 1 bids next. Firstly, it revises its utility function.

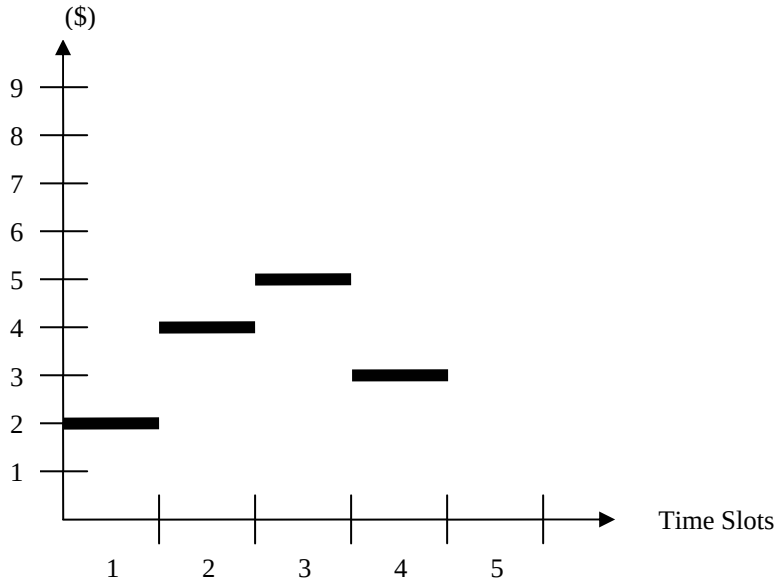


Figure 3.7p Utility function of Job 1 after second iteration

Starting at Time Slot 3 maximizes Job 1's utility. Therefore it gives the bid for Time Slots 3 and 4 with the value of $2 (price of Time Slot 3 + price of Time Slot 4 + increment value of auction = 1+0+1).

The machine agent assigns Time Slots 3 and 4 to Job 1 and they share this bid. So price of Time Slot 3 increases to $1.5 and Time Slot 4 goes up to $0.5.

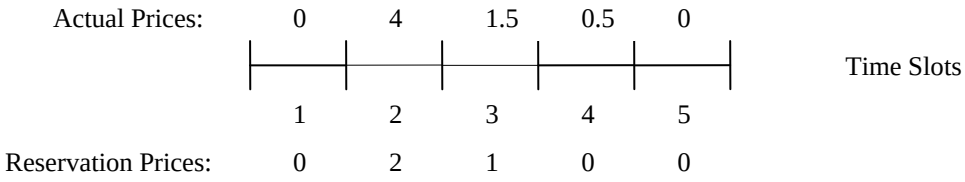


Figure 3.7q Prices of the time slots after the third iteration

Job 2 revises its utility function while starting the fourth iteration.

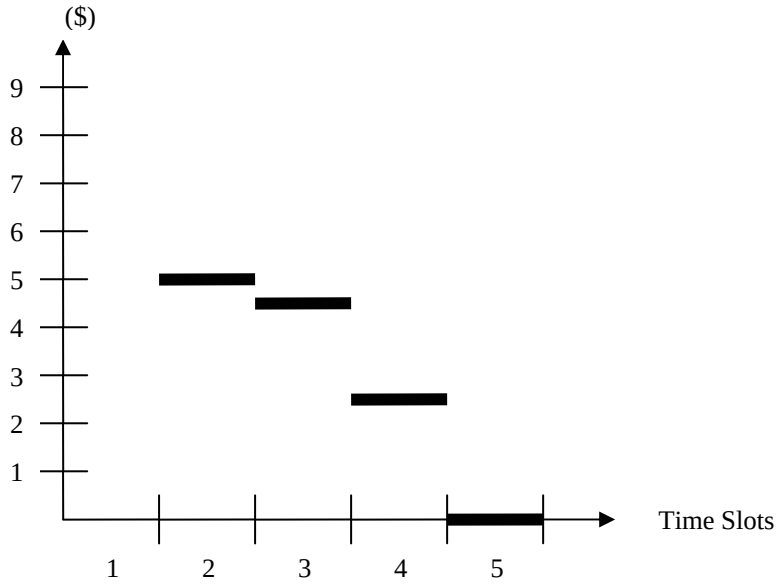


Figure 3.7r Utility function of Job 2 after the third iteration

Starting at Time Slot 2 maximizes Job 2's utility. In fact, Job 2 already possesses this utility maximizer, i.e. Time Slot 2. Therefore, it does not give a bid. The prices do not change after the fourth iteration.

Job 1 revises its utility function after the fourth iteration.

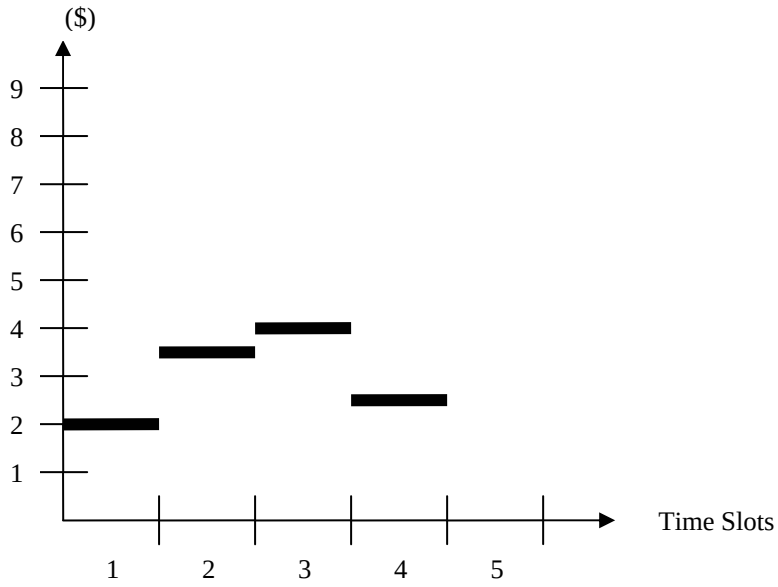


Figure 3.7s Utility function of Job 1 after the fourth iteration

Starting either at Time Slot 3 maximizes Job 1's utility. Job 1 already starts at Time Slot 3, i.e. Time Slots 3 and 4 are assigned to Job 1. Therefore, it does not give a bid, and prices do not change.

After this point, because prices do not change, the utility functions remain the same. Since both job agents have obtained their utility maximizing time slots they do not give any further bid. Thus, the algorithm stops.

Job1 has Time Slots 3 and 4, Job 2 has Time Slot 2. It means Job 1 starts being processed at time 2 and finishes at time 4, Job 2 starts being processed at time 1 and finishes at time 2. Job 1 incurs a tardiness cost of $3; Job 2 incurs no earliness or tardiness cost. Total earliness/tardiness cost is $3. This is also the optimal value for the centralized version of this problem.

3.3. Centralized Utility

The centralized utility scenario is the centralized case of the decentralized real life problem of factory scheduling in Fast Consumer Goods sector. In this setting, there is a central agent dictating job agents their schedule by maximizing the total utility functions of the job agents.

3.3.1. Branch-and-Bound Algorithm

The centralized utility problem is formulated as an integer programming problem. We call it as aggregate problem formulation because we will relax it in the next subsection. In the aggregated problem the objective function is to maximize the total utility function. The objective of the machine agent is achieved by finding a feasible schedule.

$$\max \quad \sum_{i=1}^{m} \sum_{j=1}^{n} u_{ij} x_{ij} \qquad (1)$$

s.t.

$$\sum_{j=1}^{n} \sum_{s=\max(i-p_i+1,1)}^{i} x_{sj} \leq 1 \quad i = 1,...,m \qquad (2)$$

$$\sum_{i=1}^{m-p_i+1} x_{ij} = 1 \qquad j = 1,...,n \qquad (3)$$

$$x_{ij} = 0,1 \qquad i = 1,...,m \quad j = 1,...,n \qquad (4)$$

x_{ij} is the binary variable, it is 1 if job j starts on time slot i, 0 otherwise. (1) is the objective function, that maximizes the total utility. u_{ij} is the parameter representing the utility of job j if it starts on time slot i. (2) is the capacity constraint stating that at most one job can start on a time slot and no other job can start until its processing finishes. Second set of constraints, (3), is the assignment constraint that puts the restriction that each job can and

must be scheduled only one time. As most scheduling problems, this problem is also NP-hard.

3.3.2. Lagrangean Relaxation Algorithm

The alternative algorithm is Lagrangean based scheduling algorithm. Lagrangean Relaxation has become very popular in last two decades because it provides good bounds over a problem and it is very applicable as a decomposition method. Our interest on Lagrangean Relaxation arose partially because it is used in integer programming applications of auctions. Also some researchers such as Kutanoglu and Wu (1999) and Leon and Jeong (2002) employ it in the distributed scheduling context. Furthermore, as an optimization based technique, Lagrangean Relaxation is among the most promising decomposition methods to fit in the distributed nature of our problem structure (Kutanoglu and Wu 1999). However, practically, it assumes a more centralized structure than the original problem due to the initial solution, feasibility restoration and the subgradient algorithm working under the Lagrangean Relaxation routine. Therefore we classify it under centralized utlity scenario.

We revise the Lagrangean Relaxation scheme proposed by Guignard and Rosenwein (1989) for Generalized Assignment Problem to fit into our aggregate problem. We relax (3), assignment constraints and add to the objective function with the Lagrange multipliers, w_j. The Lagrangean Relaxation Formulation is as below:

$$\max \sum_{i=1}^{m}\sum_{j=1}^{n} u_{ij}x_{ij} - \sum_{j=1}^{n}\left[w_j\left(1 - \sum_{i=1}^{m-p_i+1} x_{ij}\right)\right] \qquad (5)$$

s.t.

$$\sum_{j=1}^{n}\sum_{s=\max(i-p_i+1,1)}^{i} x_{sj} \le 1 \quad i = 1,...,m \qquad (2)$$

$$x_{ij} = 0,1 \qquad i = 1,...,m \quad j = 1,...,n \qquad (4)$$

We employ the subgradient algorithm explained by Guignard and Rosenwein (1989) to revise Lagrange multipliers. The iterations of the Lagrangean Relaxation algorithm are sketched below after the Table 3.3 which describes the parameters of the algorithm.

Table 3.3 Lagrangan Relaxation Algorithm Parameters

w_j^k :	Lagrange multiplier at iteration k
UB^k:	Upper bound at iteration *k*
LB^k:	Lower bound at iteration *k*
UBbest:	Best upper bound found
LBbest:	Best lower bound found
$target^k$:	Hypothesized target objective value to be achieved at iteration *k*
$step^k$:	Step size taken in the subgradient optimization at iteration *k*
x_{ij}^k :	solution of Lagrangean Relaxation problem at iteration k
$slack_j^k$:	slack of the j^{th} assignment constraint at iteration k
$norm^k$:	summation of the squares of the assignment constraints

3.3.2.1. Iterations of the Lagrangean Relaxation Algorithm

Iteration 0: Solve linear programming relaxation of aggregate problem.

Set UB^0 and *UBbest* to the objective value of the solution of linear programming relaxation of aggregate problem.

Set w_j^0 to the dual prices of corresponding assignment constraints of linear programming relaxation of aggregate problem solution.

Solve aggregate problem with branch-and-bound method.

Set and LB^0 and *LBbest* to the objective function value of the branch-and-bound solution of the aggregate problem.

Iteration k: Solve Lagrangean Relaxation problem.

Set UB^k to the objective function value of the solution of Lagrangean Relaxation problem.

Set $UBbest = min(UB^k, UBbest)$.

If the solution of the Lagrangean relaxation problem is feasible to the aggregate problem, set LB^k to the objective value of the aggreagate problem found by using solution of the Lagrangean Relaxation problem.

Set $target^k = (UBbest + LB^0)/2$

Set $slack_j^k = 1 - \sum_{i=1}^{m-p_i+1} x_{ij}$

Set $norm^k = \sum_{j=1}^{n} \left(slack_j^k\right)^2$

Set $step^k = (alpha^k(UB^k - target^k))/norm^k$

Update $w_j^{k+1} = w_j^k + step^k \times slack_j^k$

If there is no improvement in *UBbest* for 5 steps,

update $alpha^{k+1} = alpha^k / 2$,

else $alpha^{k+1} = alpha^k$

3.3.2.2. Feasibility Restoration Heuristic

In most cases, this Lagrangean Relaxation does not give feasible solutions for the aggregate problem. Therefore we developed a feasibility restoration heuristic to resolve this issue. The steps are described below:

Step 1: If $\sum_{i=1}^{m-p_i+1} x_{ij} > 1$ for job *j*, among x_{ij} with value 1, keep $x_{ij} = 1$ for *i* with the minimum absolute value. Assign all other 0 to x_{ij}.

Repeat the same procedure for all *j = 1,…,n*

Step 2: If $\sum_{i=1}^{m-p_i+1} x_{ij} = 0$ for job j,

Start with *i* = 1,

Step 2.i.: Assign $x_{ij} = 1$,

Check for all capacity constraints.

If all constraints hold, STOP

Else, update *i* → *i*+1

Repeat the same procedure for all *j = 1,…,n*

3.4. Centralized Cost (Classical Early/Tardy Single Machine Scheduling Problem)

Centralized cost scenario is reduced from the centralized utility case. It assumes a centralized environment in the scheduling problem of fast consumer goods sector. There is a machine agent behaving as a dictator, not taking the utilities of the job agents into account and aiming to minimize the total early/tardy cost (total inventory/backorder cost in supply chain viewpoint).

We formulated the classical single machine scheduling problem as an integer programming formulation. The constraints remain same as the aggregate problem formulation above. The objective function is changed with minimization of total early/tardy cost.

Let $\delta^+ = 1 \quad if \delta > 0$, $\delta^+ = 0 \quad o.w.$

The formulation of classical single machine early/tardy problem is below

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} e_j (d_j - ix_{ij})^+ + t_j (ix_{ij} - d_j)^+ \qquad (6)$$

s.t.

$$\sum_{j=1}^{n} \sum_{s=\max(i-p_i+1,1)}^{i} x_{sj} \le 1 \quad i = 1,...,m \qquad (2)$$

$$\sum_{i=1}^{m-p_i+1} x_{ij} = 1 \qquad j = 1,...,n \qquad (3)$$

$$x_{ij} = 0,1 \qquad i = 1,...,m \quad j = 1,...,n \qquad (4)$$

We solve this problem by Branch-and-Bound. It is an NP-hard problem.

CHAPTER 4

Experimental Design and Computational Results

4.1. Experimental Data

The design of experiments follows a previous study of Ow and Morton (1989) for the single machine early/tardy problem. The main control factors are the tardiness factor of the set of jobs to be scheduled, the due date range and the correlation coefficient between due dates and processing times.

The tardiness factor, τ, is a measure of the proportion of jobs expected to be tardy in a sequence. For a given average due date, $\bar{d}$, average processing time, $\bar{p}$, and given number of jobs, n, τ can be calculated as $\tau = 1 - \bar{d}/(n\bar{p})$.

The due date range factor, R, is used to control the range of the due date distribution calculated as $Rn\bar{p}$.

Processing times, due dates and costs are integers. The parameters of the experiment are summarized as below:

- Processing times and due dates are generated using a bivariate normal distribution which incorporates the variation in processing times, variation in due dates and the correlation between the processing times and due dates. Numbers generated are rounded to the nearest integer. The parameter levels are set as in the list below:

- Mean for processing time variate of bivariate normal is 15.
- Coefficient of variation for processing time variate of bivariate normal, i.e, standard deviation/mean, is 0.2.
- Due date range factor is, R, is 0.4 and 1.0.
- Correlation coefficient between processing times and due dates, ρ, is 0 and 0.5.
- Tardiness factor, τ, is set at 0.2 and 0.6.

- Tardy cost rate, t_j, is calculated in relation to work content: $t_j = (t/p) \times p_j$, the cost per unit processing time, t/p, is obtained from a uniform distribution in range [0, 5].
- Early cost rate, e_j, is set proportional to a early/tardy cost rate, *e/t*. It is calculated as $e_j = (e/t) \times t_j$. Early/tardy cost rate is set to 5%, 10%, 25%.
- Number of jobs, *n*, is set to 8, 15, and 25.

For each experimental point (72 in total) we generate 20 different test problems and we assign the mean of the results 20 different test data as the representative of corresponding experimental point (1440 data sets in total). The summary of experimental factors can be seen in Table 4.1 (release times are assumed to be same for all jobs, i.e. release times are 0).

Table 4.1 Summary of Experimetal Parameters

Experimental Factors:	**Levels:**
Number of Jobs (*n*)	*n* = 8
	n = 15
	n = 25
Tardiness Factor (τ): proportion of jobs that might be expected to be tardy in an arbitrary sequence	τ = 0.2
	τ = 0.6
Due Dates Range Factor (*R*)	*R =0.4*
	R =1.0
Correlation Coefficient between Processing Times and Due Dates (ρ)	ρ = 0
	ρ = 0.5
Early Cost Rate (*e/t*): proportion of unit earliness cost to unit tardiness cost	*e/t* = 0.05
	e/t = 0.10
	e/t = 0.25

4.2. Results

We ran experiments for Auction Based Algorithm (Auction), Lagrangean Relaxation Algorithm (LR), Branch-and-Bound Utility (B&B-Utility) and Branch-and-Bound Cost (B&B-Cost). We developed the code in Java (see Appendix A) for Auction Based Algorithm and in GAMS (see Appendix B) working over CPLEX optimizer for the rest. Our primary performance criterion for Auction, LR and B&B-Utility is total utility of the job agents, our secondary performance criterion for Auction, LR and B&B-Cost is total cost of the job agents as discussed in Chapter 3.

We used different number of maximum time slots and increment values for different number of jobs cases. We assigned enough number of time slots to obtain a feasible schedule. The reason to use different incremental values for different job sizes is to make the Auction Based Algorithm work faster. Table 4.2 gives the corresponding maximum time slots and increment values:

Table 4.2 Computational Parameters

	n = 8	n = 15	n = 25
Maximum Number of Time Slots	250	500	1000
Increment Value of the English Auction	2.5	20	250

The maximum number of iterations for the Lagrangean Relaxation Algorithm is 20 for all different job sizes.

Table 4.3 gives the total utility results for Auction, LR and B&B-Utility. Table 4.4 shows the total cost results for Auction, LR and B&B-Cost for all experimental points:

Table 4.3 Total Utility Results for Auction, LR, and B&B-Utility

	n = 8			*n = 15*			*n = 25*		
	Auction	LR	B&B-Utility	Auction	LR	B&B-Utility	Auction	LR	B&B-Utility
τ=0.2, R=0.4, ρ=0.0, e/t=0.05	72,358	79,723	78,871	308,391	349,853	352,004	1,216,943	1,405,109	1,396,388
τ=0.2, R=0.4, ρ=0.0, e/t=0.10	72,593	79,173	78,324	303,313	349,536	350,658	1,208,288	1,385,964	1,389,422
τ=0.2, R=0.4, ρ=0.0, e/t=0.25	72,973	78,468	77,902	298,634	344,991	346,518	1,195,810	1,385,034	1,377,576
τ=0.2, R=0.4, ρ=0.5, e/t=0.05	68,957	78,672	78,210	306,075	343,243	342,828	1,180,278	1,366,355	1,338,916
τ=0.2, R=0.4, ρ=0.5, e/t=0.10	69,943	78,129	77,559	286,348	341,362	341,316	1,153,677	1,348,618	1,332,012
τ=0.2, R=0.4, ρ=0.5, e/t=0.25	69,965	77,115	76,945	288,073	335,583	336,447	1,159,601	1,330,016	1,317,782
τ=0.2, R=1.0, ρ=0.0, e/t=0.05	74,995	81,320	81,261	314,821	358,379	362,189	1,258,794	1,407,976	1,417,839
τ=0.2, R=1.0, ρ=0.0, e/t=0.10	76,349	80,759	81,647	321,742	358,975	360,392	1,232,355	1,409,623	1,412,666
τ=0.2, R=1.0, ρ=0.0, e/t=0.25	77,613	80,648	81,248	309,870	358,436	357,753	1,224,274	1,375,230	1,402,493
τ=0.2, R=1.0, ρ=0.5, e/t=0.05	69,755	76,953	76,615	284,247	330,390	331,982	1,115,410	1,285,047	1,293,403
τ=0.2, R=1.0, ρ=0.5, e/t=0.10	68,413	76,602	76,757	274,342	329,498	330,651	1,089,440	1,285,381	1,289,404
τ=0.2, R=1.0, ρ=0.5, e/t=0.25	69,949	75,850	76,095	291,492	328,137	326,604	1,110,351	1,261,432	1,277,327
τ=0.6, R=0.4, ρ=0.0, e/t=0.05	94,640	100,656	101,047	391,063	440,811	440,569	1,399,795	1,649,088	1,628,770
τ=0.6, R=0.4, ρ=0.0, e/t=0.10	95,362	100,685	101,252	392,582	442,057	440,342	1,375,043	1,652,626	1,621,310
τ=0.6, R=0.4, ρ=0.0, e/t=0.25	95,057	100,500	101,085	377,411	440,021	439,668	1,433,874	1,636,188	1,618,871
τ=0.6, R=0.4, ρ=0.5, e/t=0.05	92,602	100,480	99,634	378,653	428,410	431,981	1,379,487	1,639,924	1,584,088
τ=0.6, R=0.4, ρ=0.5, e/t=0.10	92,103	100,222	99,567	378,047	429,601	430,698	1,352,917	1,631,856	1,580,887
τ=0.6, R=0.4, ρ=0.5, e/t=0.25	92,347	99,671	99,652	382,634	428,133	430,555	1,366,930	1,631,682	1,573,953
τ=0.6, R=1.0, ρ=0.0, e/t=0.05	97,899	102,617	102,699	399,761	448,885	449,978	1,449,879	1,639,262	1,655,923
τ=0.6, R=1.0, ρ=0.0, e/t=0.10	96,356	102,418	102,830	398,638	449,442	449,876	1,445,337	1,636,408	1,655,295
τ=0.6, R=1.0, ρ=0.0, e/t=0.25	95,686	102,370	102,580	410,619	450,431	450,328	1,428,477	1,629,210	1,656,271
τ=0.6, R=1.0, ρ=0.5, e/t=0.05	92,488	98,815	98,891	352,192	420,128	420,721	1,317,817	1,557,404	1,536,329
τ=0.6, R=1.0, ρ=0.5, e/t=0.10	94,115	98,244	99,209	369,790	420,017	420,599	1,325,818	1,555,147	1,535,629
τ=0.6, R=1.0, ρ=0.5, e/t=0.25	93,700	98,881	98,896	377,624	421,292	420,404	1,331,997	1,539,549	1,526,441

Table 4.4 Total Cost Results for Auction, LR, and B&B-Cost

	n = 8			*n = 15*			*n = 25*		
	Auction	LR	B&B-Cost	Auction	LR	B&B-Cost	Auction	LR	B&B-Cost
T=0.2, R=0.4, ρ=0.0, e/t=0.05	7,664	511	656	45,218	1,286	2,800	178,353	3,409	15,232
T=0.2, R=0.4, ρ=0.0, e/t=0.10	7,889	832	1,198	52,317	2,309	3,706	192,205	6,371	19,910
T=0.2, R=0.4, ρ=0.0, e/t=0.25	7,531	1,636	1,886	52,175	5,094	7,201	200,212	5,094	33,181
T=0.2, R=0.4, ρ=0.5, e/t=0.05	10,652	645	815	40,745	1,788	3,006	174,530	4,715	10,378
T=0.2, R=0.4, ρ=0.5, e/t=0.10	9,652	1,023	1,350	57,281	3,099	4,542	208,870	8,419	15,519
T=0.2, R=0.4, ρ=0.5, e/t=0.25	8,798	2,058	2,342	53,868	6,515	8,543	195,485	18,260	29,523
T=0.2, R=1.0, ρ=0.0, e/t=0.05	7,492	243	1,058	49,118	561	1,528	162,349	1,420	14,505
T=0.2, R=1.0, ρ=0.0, e/t=0.10	6,099	431	1,598	42,801	1,084	2,975	189,664	2,731	18,052
T=0.2, R=1.0, ρ=0.0, e/t=0.25	4,829	940	1,702	53,415	2,488	5,281	192,260	6,156	27,663
T=0.2, R=1.0, ρ=0.5, e/t=0.05	8,109	435	822	50,987	1,144	2,515	179,362	2,910	13,937
T=0.2, R=1.0, ρ=0.5, e/t=0.10	9,460	748	1,161	59,278	2,154	3,628	207,623	5,598	17,960
T=0.2, R=1.0, ρ=0.5, e/t=0.25	7,902	1,639	1,907	42,676	4,780	6,701	196,837	12,528	31,699
T=0.6, R=0.4, ρ=0.0, e/t=0.05	11,108	3,473	5,015	62,328	8,822	12,340	306,927	23,132	47,388
T=0.6, R=0.4, ρ=0.0, e/t=0.10	10,378	3,562	4,994	60,899	9,129	12,509	304,014	23,988	48,547
T=0.6, R=0.4, ρ=0.0, e/t=0.25	10,697	3,802	5,164	74,627	10,194	13,169	271,047	27,230	51,137
T=0.6, R=0.4, ρ=0.5, e/t=0.05	12,292	3,595	4,308	66,217	9,790	12,986	290,847	25,867	41,602
T=0.6, R=0.4, ρ=0.5, e/t=0.10	12,500	3,720	4,563	65,089	10,248	14,020	307,679	27,301	44,766
T=0.6, R=0.4, ρ=0.5, e/t=0.25	12,529	4,061	5,112	62,210	11,613	13,985	302,033	31,028	46,136
T=0.6, R=1.0, ρ=0.0, e/t=0.05	8,514	3,096	3,724	58,826	6,661	9,887	269,919	15,027	39,639
T=0.6, R=1.0, ρ=0.0, e/t=0.10	10,103	3,163	3,924	62,163	6,813	10,059	280,773	15,259	39,893
T=0.6, R=1.0, ρ=0.0, e/t=0.25	10,761	3,301	3,976	49,510	7,136	9,531	288,883	16,464	40,516
T=0.6, R=1.0, ρ=0.5, e/t=0.05	10,224	3,083	4,116	82,984	8,162	11,867	289,273	20,106	44,557
T=0.6, R=1.0, ρ=0.5, e/t=0.10	8,872	3,177	4,686	63,067	8,396	11,983	290,542	21,021	46,345
T=0.6, R=1.0, ρ=0.5, e/t=0.25	9,315	3,453	4,036	55,815	9,336	12,140	280,874	24,137	47,295

Table 4.5 and Table 4.6 show the averages of the experimental points for total utility and total cost:

Table 4.5 Averages of the Experimental Points for Total Utility

	n = 8			*n = 15*			*n = 25*		
	Auction	LR	B&B-Utility	Auction	LR	B&B-Utility	Auction	LR	B&B-Utility
Overall Average	83,176	89,540	89,532	341,515	390,211	389,484	1,281,358	1,475,791	1,485,172

Table 4.6 Averages of the Experimental Points for Total Cost

	n = 8			*n = 15*			*n = 25*		
	Auction	LR	B&B-Cost	Auction	LR	B&B-Cost	Auction	LR	B&B-Cost
Overall Average	9,307	2,921	2,193	56,817	8,204	5,775	240,023	32,724	14,507

As can easily be seen, LR gives very close results to B&B-Utility in terms of total utility. Even LR is slightly better than B&B-Utility in small problem sizes. However, there is a considerable gap between LR and B&B-cost in terms of total utility. The reason for such good results for total utility is it works in the same environment with B&B-Utility. However, it does not try to minimize total cost as B&B-Cost does with a master (dictator) agent with no competition and full collaboration.

The Auction gives much worse results than LR and B&B-Cost by the means of total cost. However, it gives close results to LR and B&B-Utility by the means of total utility. There is not a general aim of the system (no master agent) in Auction. The jobs are in competition and there is collaboration partly provided by negotiations. Auction works in a highly decentralized environment although LR, B&B-Utility and B&B-Cost work in a completely centralized environment.

B&B-Cost beats LR and Auction because the master agent solves the minimization of total cost problem not allowing competition and dictating job agents their schedules because it is accessible to all data sets of job agents.

In the experiments, lower and upper bounds for B&B-Utility and B&B-Cost do not converge to an optimal solution. We finish the B&B immaturely. Therefore, for some cases LR gives better results than B&B-Utility.

Table 4.7 shows the average total cost and averaged percentage gap between Auction Based Algorithm and Lagrangean Relaxation Algorithm for different jobs sizes:

Table 4.7 Average Total Cost and Percentage Gap

	n = 8		*n = 15*		*n = 25*	
	Auction	LR	Auction	LR	LR	LR
Overall Average	9,307	2,921	56,817	8,204	240,023	32,724
Average Percentage Gap	218.58%		592.54%		633.48%	

There is very big gap between the total cost results of Auction Based Algorithm and Lagrangean Relaxation Algorithm. However as can be seen in Table 4.8, the gap decreases a lot for total utility values.

Table 4.8 Average Total Utility and Percentage Gap

	n = 8		*n = 15*		*n = 25*	
	Auction	LR	Auction	LR	Auction	LR
Overall Average	83,176	89,540	341,515	390,211	1,281,358	1,475,791
Average Percentage Gap	-7.11%		-12.48%		-13.17%	

As expected, the results of the Auction Based algorithm are all below the Lagrangean Relaxation Algorithm. Because we start with a very good incumbent (the solution of the original problem) and very good Lagrangean multipliers (the dual costs of the constraints of the Linear Programming Relaxation of the original problem). Also, because the Lagrangean Relaxation upper bound actually converges to Linear Programming Relaxation of this type of problems, solutions to the problems of the iterations of the Lagrangean Relaxation Algorithm alter in a very small range. The total utility results mean that the Auction Based Algorithm works very well for the maximization of total utility function although it works on a highly distributed environment whereas Lagrangean Relaxation Algorithm works on a centralized environment. Unfortunately, we do not have a measure to evaluate the degree of decentralization. We conclude that Auction converges to good solutions for the total utility measure.

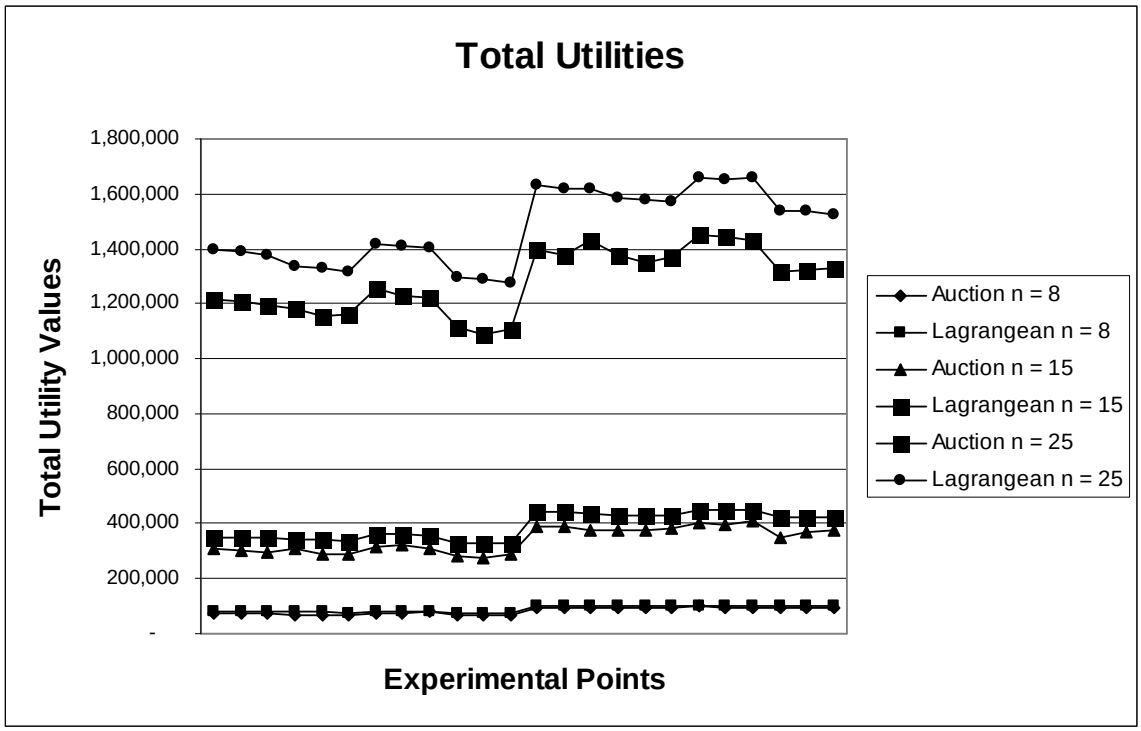


Figure 4.1 Comparison Chart of Total Utilities of Auction Based Algorithm and Lagrangean Relaxation Algorithm

Figure 4.1 depicts us the comparison of total utilities of Auction and LR. There is an increase observed in total utilities when tardiness factor, τ, is higher. When correlation coefficients between processing times and due dates, ρ, is higher, total utility decreases.

Chapter 5

Conclusion and Future Research Directions

In this thesis, we solve the scheduling problem of a geographically distributed supply chain in Fast Consumer Goods sector. We analyze the problem under three scenarios, centralized cost (classical single machine scheduling problem), centralized utility and decentralized utility. We solve the centralized cost case with Branch-and-Bound. We also solve the centralized utility case with Branch-and-Bound and we develop a Lagrangean Relaxation Algorithm for this case. The most realistic scenario is the decentralized cost case and we developed an Auction Based Algorithm to solve this case.

We reduced the supply chain scheduling problem to a single machine scheduling problem by introducing utility function concept instead of early/tardy cost. This gives us an opportunity to use game theoretic approaches on our highly decentralized problem structure.

By employing different solution methodologies from the ideal case of the problem to the most realistic case, we compare different scenarios. While Branch-and-Bound beats every other method in the most ideal case (centralized cost), Lagrangean Relaxation Algorithm and Brach-and-Bound produce similar results for the centralized utility function case. However, in the decentralized utility case, the Auction Based Algorithm is the only alternative that fits to the structure of the real life problem. While it does not work very efficient for the total cost performance, the total utility results are very promising.

In our opinion, by introducing Auction Based Algorithm, we add game theoretic insight to the scheduling problem. While optimization methods work well for centralized cases, game theoretic approaches offer more diverse solution opportunities for decentralized case. By this algorithm, we introduce a very realistic negotiation mechanism between the agents of the system.

Auction Based Algorithm is a pioneer in our problem structure. It works quiet well, however, it does not contain any optimization features. It is a possible research opportunity to define optimization submodels and employ them in the auction mechanism.

We work in a single supplier (or single machine) environment. A possible extension is to move the problem in a two-supplier (two-machine) or multi-supplier (multi-machine) environment.

Lagrangean Relaxation Algorithm is also a promising algorithm for our problem structure. Although it is working on centralized environment, with some revisions it can be made to work on decentralized case. One of the ways is instead of employing subgradient algorithm under Lagrangean Relaxation Algorithm, we can develop a pricing heuristic. Then, it behaves like an auction algorithm.

Different utility functions can be fit to the job agents. By adding extra components to the utility function we gain great flexibility for the scheduling solution.

There is not a measure of degree of decentralization. If there was one, it would be better to evaluate the performance of an algorithm working on a decentralized environment. This is an intensively theoretic research.

Bibliography

Benjamin P., Yen, C., “Communication Infrastructure in Distributed Scheduling”, Computers and Industrial Engineering 42, 2002.

Brennan, R.W., Norrie, D.H., O, W., Walker, S.S., “Job Sequencing and Dispatching in Multi-agent Heterarchical Control Systems”, University of Calgary, Calgary, Alberta, Canada: 2000.

Chen, Y., Cost, S., Finin, T., Labrou, Y., Peng Y., “Negotiating Agents for Supply Chain Management”, Department of Computer Science and Electrical Engineering, University of Maryland, USA: 1999.

Dewan, P., Joshi, S., “Implementation of an Auction-Based Distributed Scheduling Model for a Dynamic Job Shop Environment”, International Journal of Computer Integrated Manufacturing 14, 2001.

Dutta, P.S., Mukherjee, R., Sen, S., “Scheduling to be Competitive in Supply Chains”, Department of Mathematical and Computer Sciences, University of Tulsa, USA: 2000.

Grimm, V., Riedel, R., Wolfstetter, E., “Low Price Equilibrium in Multi-Unit Auctions: the GSM spectrum auction in German”, International Journal of Industrial Organization 21, 2003.

Jeong, I.J. Leon, V.J., “A Distributed Scheduling Methodology for a Two-machine Flowshop using Cooperative-interaction via Multiple Coupling-agents”, Texas A&M University, College Station Texas, USA: 2002a.

Jeong, I.J. Leon, V.J., A Single-machine Distributed Scheduling Methodology Using Cooperative-interaction via Coupling-agents, Texas A&M University, College Station Texas, USA: 2002b.

Kaihara, T., "Agent-Based Double Auction Algorithm for Global Supply Chain System", IEEE, 2000.

Khoo, L.P., Lee, S.G., Yin, X.F., "Agent-based Multiple Shop Floor Manufacturing Scheduler", International Journal of Production Research 39, 2001.

Kutanoglu, E., Wu, S.D., "On Combinatorial Auction and Lagrangean Relaxation for Distributed Resource Scheduling", IIE Trans. 31, 1999.

Kutanoglu, E., Wu, S.D., "Collaborative Resource Planning with Distributed Agents, The University of Texas at Austin-Lehigh University, USA: 2002.

Najid, N.M., Kouiss, K., Derriche, O., "Agent Based Approach for a Real-Time Shop Floor Control", Technical Paper, IRCCyN/IUT de Nantes-LIMOS/IFMA, France: 2003.

Ow, P., S., Morton, E., M., "The Single Machine Early/Tardy Problem", Management Science, 35(2), 1989.

Roy, D., Anciaux, D., "Shop Floor Control: A Multi-Agents Approach", International Journal of Computer Integrated Manufacturing 14, 2001.

Sabuncuoglu I., Toptal A., "Distributed Scheduling: Part 1 – A Review of Concepts and Applications", Technical paper: IE99-XX, Bilkent University, Turkey: 1999a.

Sabuncuoglu I., Toptal A., "Distributed Scheduling: Part 2 – Bidding Algorithms and Performance Evaluations", Technical paper: IE99-XX, Bilkent University, Turkey: 1999b.

Sabuncuoglu I., Toptal A., "Distributed Scheduling: Part 3 – Product Team Based Algorithms", Technical paper: IE99-XX, Bilkent University, Turkey: 1999c.

Sauter, J., A., Parunak, H.V.D., "ANTS in the Supply Chain", Workshop on Agent based Decision Support for Managing the Internet-Enabled Supply Chain, Agents, 99, USA: 1999.

Seredynski, F., "Competitive Coevolutionary Multi-Agent Systems: The Application to Mapping and Scheduling Problems", Journal of Parallel and Distributed Computing 47, 1997.

Seredynski, F., Koronacki, J., Janikow, C.Z., "Distributed Multiprocessor Scheduling with Decomposed Optimization Criterion", Future Generation Computer Systems 17, 2001.

Shen L., J., Chua, D., K., C., Bok, S., H., “Distributed Scheduling with Integrated Production Scheduler”, Department of Civil Engineering, National University of Singapore, Singapore: 1999.

Tharumarajah, A., Bemelman, R., “Approaches and Issues in Scheduling a Distributed Shop Floor Environment”, Computers in Industry 34, 1997.

Tonshoff, H.K., Leitao, P., Seilonen, I., Teunis, G., “A Mediator-based Approach for Decentralized Production Planning Scheduling and Monitoring”, University of Hannover, Germany: 1999.

Wellman, M.P., Walsh, W.E., Wurman, P.R., MacKie-Mason, J.K., “Auction Protocols for Decentralized Scheduling”, Games and Economic Behavior 35, 2001.

APPENDIX A

Java Code of Auction Based Algorithm

SystemCore.java

```
import java.io.*;

public class SystemCore{

public static void main (String[] args) throws IOException
{
	int release2 = 0;
	int duedate2 = 0;
	int process2 = 0;
	double early2 = 0;
	double tardy2 = 0;
	int counter = 0;
	final int MAXJOBS = 8;
	int CORDOBA = 20;
	int[] earliness = new int[MAXJOBS];
	int[] tardiness = new int[MAXJOBS];
	double[] earlycost = new double[MAXJOBS];
	double[] tardycost = new double[MAXJOBS];
	int[] starttime = new int[MAXJOBS];
	int[] finishtime = new int[MAXJOBS];
	double[] machineutility = new double[CORDOBA];
	boolean[] scheduled = new boolean[MAXJOBS];
	double[] totalutility = new double[CORDOBA];
	double[] totalutility2 = new double[CORDOBA];
	double[] totalcost = new double[CORDOBA];
	Job[] jobs = new Job[MAXJOBS];
	double avgtotalutility = 0;
	double avgtotalutility2 = 0;
```

```
	double avgtotalcost = 0;
	double avgmachineutility = 0;

	for (int carew = 1; carew <= CORDOBA; carew++)
	{
		System.out.println(carew);
		int numberscheduled = 0;
		int numbernotscheduled = 0;
		String line, name, file = "input8-02-04-00-005-"+carew+".dat";

		FileReader fr = new FileReader(file);
		BufferedReader infile = new BufferedReader (fr);
			int count7 = 0;
			for (int index = 1; index <= 5 * MAXJOBS; index++)
			{
				line = infile.readLine();
				if(index%5 == 1){
					release2 = Integer.parseInt(line);
				}
				if(index%5 == 2){
					duedate2 = Integer.parseInt(line);
				}
				if(index%5 == 3){
					process2 = Integer.parseInt(line);
				}
				if(index%5 == 4){
					early2 = Double.parseDouble(line);
				}
				if(index%5 == 0){
					tardy2 = Double.parseDouble(line);
					jobs[count7] = new Job(release2, duedate2, process2,
early2, tardy2);
					count7++;
				}

			}

			/*close file*/
			infile.close();

			Machine machine = new Machine(jobs);

		double[] reservationprices1 = machine.GenerateReservationPrices();

		for (int index = 1; index <= MAXJOBS; index++)
		{
			jobs[index-1].FindMaxcost();
```

```
		}


		double normalizer1 = machine.FindNormalizer();

		for (int index = 1; index <= MAXJOBS; index++)
		{
			jobs[index-1].GenerateUtilityFunction(reservationprices1,
normalizer1);
		}

		while (counter < MAXJOBS)
		{
			for (int index = 1; index <= MAXJOBS; index++)
			{

				jobs[index-1].FindMaxUtility();

				for (int subindex = 1; subindex <= Job.MAXTIMESLOTS;
subindex++)
				{
					Machine.oldstatus[subindex-1] =
Machine.status[subindex-1];
					Machine.oldreservationprices[subindex-1] =
Machine.reservationprices[subindex-1];
					Machine.oldprices[subindex-1] =
Machine.prices[subindex-1];
				}


				int target1 = jobs[index-1].argmaxutility;
				boolean result1 = false;
				int count1 = 0;
				boolean resultmaxutility = false;

				for (int subindex = 1; subindex <= Job.MAXTIMESLOTS;
subindex++)
				{
					if (Machine.oldstatus[subindex-1] == index &&
jobs[index-1].utilities[subindex-1] == jobs[index-1].maxutility)
					{
						result1 = true;
						subindex = Job.MAXTIMESLOTS+1;
					}
				}

				if (jobs[index-1].maxutility > 0 && result1 == false)
```

```
                                {
                                        jobs[index-1].PrepareBid();

                                        machine.AssignTimeSlots(index);

                                        machine.DeterminePrices(index);
                                }

                                else
                                {
                                        for (int subindex = 1; subindex <=
Job.MAXTIMESLOTS; subindex++)
                                        {
                                                Machine.status[subindex-1] =
Machine.oldstatus[subindex-1];
                                                Machine.prices[subindex-1] =
Machine.oldprices[subindex-1];
                                                Machine.reservationprices[subindex-1] =
Machine.oldreservationprices[subindex-1];
                                        }
                                }



                                for (int subindex = 1; subindex <= MAXJOBS; subindex++)
                                {
                                        jobs[subindex-1].ReviseUtilities();
                                }

                                if (machine.isChanged())
                                {
                                        counter++;
                                }
                                else
                                {
                                        counter = 0;
                                }

//                              System.out.println("Counter = " + counter);
                        }
                }


                for (int index = 1; index <= MAXJOBS; index++)
                {
```

```
                    for (int subindex = 1; subindex <= Job.MAXTIMESLOTS;
subindex++)
                    {
                        if (Machine.status[subindex-1] == index)
                        {
                            scheduled[index-1] = true;
                            subindex = Job.MAXTIMESLOTS+1;
                        }

                        else
                        {
                            scheduled[index-1] = false;

                        }

                    }

                    if (scheduled[index-1] == true)
                    {
                        numberscheduled++;
                    }
                    else
                    {
                        numbernotscheduled++;
                    }
                }

                machine.assignstatus();

        //biter insallah
                counter = 0;

                int counter10 = 0;
                while (numbernotscheduled != 0)
                {
                    counter10++;
                    double[] reservationprices2 =
machine.GenerateReservationPrices2(scheduled);
        /*          for(int i=0; i<100; i++)
                    {
                        System.out.println(reservationprices2[i]);
                    }*/
                    for (int index = 1; index <= MAXJOBS; index++)
                    {
                        if (scheduled[index-1] == false)
                        {
                            jobs[index-1].FindMaxcost2();
```

```
                        }
                }
        /*      for(int i=0; i<8; i++)
                {
                        System.out.println(jobs[i].maxcost);
                }*/

                double normalizer2 = machine.FindNormalizer2(scheduled);
        //      System.out.println(normalizer2);

                for (int index = 1; index <= MAXJOBS; index++)
                {
                        if (scheduled[index-1] == false)
                        {
                                jobs[index-
1].GenerateUtilityFunction2(reservationprices2, normalizer2);
                        }
                }

        /*      for(int i=0; i<240; i++)
                {
                        System.out.println(jobs[3].utilities[i]);
                }*/

                int counter101 = 0;
                int counter11 = 0;

                while (counter11 < numbernotscheduled)
                {
                        counter11 = 0;
                        counter101++;
                        for (int index = 1; index <= MAXJOBS; index++)
                        {
                                if (scheduled[index-1] == false)
                                {
                                        jobs[index-1].FindMaxUtility2();
/*                                      if (counter101 == 1)
                                        {
                                                System.out.println(jobs[index-
1].maxutility);
                                                System.out.println(jobs[index-
1].argmaxutility);
                                        }*/

                                        for (int subindex = 1; subindex <=
Job.MAXTIMESLOTS; subindex++)
                                        {
```

```
                                        Machine.oldstatus[subindex-1] =
Machine.status[subindex-1];
                                        Machine.oldreservationprices[subindex-
1] = Machine.reservationprices[subindex-1];
                                        Machine.oldprices[subindex-1] =
Machine.prices[subindex-1];
                                }

                                int target2 = jobs[index-1].argmaxutility;
                                boolean result2 = false;
                                int count2 = 0;
                                boolean resultmaxutility2 = false;

                                for (int subindex = 1; subindex <=
Job.MAXTIMESLOTS; subindex++)
                                {
                                        if (Machine.status2[subindex-1] == 0)
                                        {
                                                if (Machine.oldstatus[subindex-1]
== index && jobs[index-1].utilities[subindex-1] == jobs[index-1].maxutility)
                                                {
                                                        result2 = true;
                                                        subindex =
Job.MAXTIMESLOTS+1;
                                                }
                                        }
                                }

                                if (jobs[index-1].maxutility > 0 && result2 ==
false)
                                {
                                        jobs[index-1].PrepareBid2();

                                        machine.AssignTimeSlots2(index);

                                        machine.DeterminePrices2(index);
                                }

                                else
                                {
                                        for (int subindex = 1; subindex <=
Job.MAXTIMESLOTS; subindex++)
                                        {
                                                Machine.status[subindex-1] =
Machine.oldstatus[subindex-1];
                                                Machine.prices[subindex-1] =
Machine.oldprices[subindex-1];
```

```
Machine.reservationprices[subindex-1] = Machine.oldreservationprices[subindex-1];
                                        }
                                }

                                for (int subindex = 1; subindex <= MAXJOBS;
subindex++)
                                {
                                        if (scheduled[subindex-1] == false)
                                        {
                                                jobs[subindex-
1].ReviseUtilities2();
                                        }
                                }

                                if (machine.isChanged())
                                {
                                        counter11++;
                                }
                                else
                                {
                                        counter11 = 0;
                                }

//                              System.out.println("Counter = " + counter11);


                        }
                }
        }



        for (int index = 1; index <= MAXJOBS; index++)
        {
                if (scheduled[index-1] == false)
                {
                        for (int subindex = 1; subindex <=
Job.MAXTIMESLOTS; subindex++)
                        {
                                if (Machine.status[subindex-1] == index)
                                {
                                        scheduled[index-1] = true;
                                        subindex = Job.MAXTIMESLOTS+1;
                                }

                                else
```

```
                                        {
                                                scheduled[index-1] = false;

                                        }

                                }

                                if (scheduled[index-1] == true)
                                {
                                        numberscheduled++;
                                        numbernotscheduled--;
                                }
                        }
                }

//                      System.out.println("counter10 = "+ counter10);
//                      System.out.println("not scheduled = " + numbernotscheduled);
//                      System.out.println("scheduled = " + numberscheduled);
        }

        System.out.println("not scheduled = " + numbernotscheduled);
        System.out.println("scheduled = " + numberscheduled);

        for (int index = 1; index <= MAXJOBS; index++)
        {
                for (int subindex = 1; subindex <= Job.MAXTIMESLOTS;
subindex++)
                {
                        if (Machine.status[subindex-1] == index)
                        {
                                starttime[index-1] = subindex-1;
                                finishtime[index-1] = subindex-1+jobs[index-
1].getProcTime();
                                subindex = Job.MAXTIMESLOTS+1;
                        }
                }
        }

        totalutility[carew-1] = 0;
        for (int index = 1; index <= MAXJOBS; index++)
        {
                totalutility[carew-1] += jobs[index-1].firstutilities[starttime[index-1]];
        }
        System.out.println("Total Utility = " + totalutility[carew-1]);


        for (int index = 1; index <= MAXJOBS; index++)
```

```
		{
			if (finishtime[index-1] < jobs[index-1].getDueDate())
			{
				earliness[index-1] = jobs[index-1].getDueDate()-
finishtime[index-1];
				tardiness[index-1] = 0;
			}
			else
			{
				tardiness[index-1] = finishtime[index-1]-jobs[index-
1].getDueDate();
				earliness[index-1] = 0;
			}

			earlycost[index-1] = jobs[index-1].getEarly()*earliness[index-1];
			tardycost[index-1] = jobs[index-1].getTardy()*tardiness[index-1];
		}


		totalcost[carew-1] = 0;
		for (int index = 1; index <= MAXJOBS; index++)
		{
			totalcost[carew-1] = totalcost[carew-1]+earlycost[index-
1]+tardycost[index-1];
		}
		System.out.println("Total Cost = " + totalcost[carew-1]);

		totalutility2[carew-1] = 0;
		for (int index = 1; index <= MAXJOBS; index++)
		{
			totalutility2[carew-1] += jobs[index-1].utilities[starttime[index-1]];
		}
		System.out.println("Total Utility2 = " + totalutility2[carew-1]);


		machineutility[carew-1] = 0;
		for (int index = 1; index <= Job.MAXTIMESLOTS; index++)
		{
			machineutility[carew-1] += Machine.prices[index-1];
		}
		System.out.println("Machine Utility = " + machineutility[carew-1]);

		String file6 = "costs"+carew+".gms";//cost data

		FileWriter fw6 = new FileWriter (file6);
		BufferedWriter bw6 = new BufferedWriter (fw6);
		PrintWriter outFile6 = new PrintWriter (bw6);
```

```
		for (int index=1; index<=MAXJOBS; index++)
		{
			outFile6.print("\t"+"i"+index);
		}
		outFile6.println();
		for (int index=1; index<=Job.MAXTIMESLOTS; index++)
		{
			outFile6.print("r"+index);
			for (int subindex=1; subindex<=MAXJOBS; subindex++)
			{
				outFile6.print("\t"+jobs[subindex-1].firstcosts[index-1]);
			}
			outFile6.println();
		}
		outFile6.close();

		String file5 = "process"+carew+".gms";//utility data

		FileWriter fw5 = new FileWriter (file5);
		BufferedWriter bw5 = new BufferedWriter (fw5);
		PrintWriter outFile5 = new PrintWriter (bw5);

		outFile5.print("/");
		for (int index=1; index<=MAXJOBS; index++)
		{
			if(index<MAXJOBS)
			{
				outFile5.println("i"+index+"\t"+jobs[index-1].getProcTime());
			}
			else
			{
				outFile5.println("i"+index+"\t"+jobs[index-
1].getProcTime()+"/");
			}
		}
		outFile5.close();


		String file4 = "tablebi.gms";//utility data

		FileWriter fw4 = new FileWriter (file4);
		BufferedWriter bw4 = new BufferedWriter (fw4);
		PrintWriter outFile4 = new PrintWriter (bw4);

		outFile4.print("/");
		for (int index=1; index<=Job.MAXTIMESLOTS; index++)
```

```
{
	if(index<Job.MAXTIMESLOTS)
	{
		outFile4.println("r"+index+"\t"+"1");
	}
	else
	{
		outFile4.println("r"+index+"\t"+"1"+"/");
	}
}
outFile4.close();

String file3 = "tableaij.gms";//utility data

FileWriter fw3 = new FileWriter (file3);
BufferedWriter bw3 = new BufferedWriter (fw3);
PrintWriter outFile3 = new PrintWriter (bw3);

for (int index=1; index<=MAXJOBS; index++)
{
	outFile3.print("\t"+"i"+index);
}
outFile3.println();
for (int index=1; index<=Job.MAXTIMESLOTS; index++)
{
	outFile3.print("r"+index);
	for (int subindex=1; subindex<=MAXJOBS; subindex++)
	{
		outFile3.print("\t"+"1");
	}
	outFile3.println();
}
outFile3.close();

String file2 = "utilities"+carew+".gms";//utility data

FileWriter fw2 = new FileWriter (file2);
BufferedWriter bw2 = new BufferedWriter (fw2);
PrintWriter outFile2 = new PrintWriter (bw2);

for (int index=1; index<=MAXJOBS; index++)
{
	outFile2.print("\t"+"i"+index);
}
outFile2.println();
for (int index=1; index<=Job.MAXTIMESLOTS; index++)
{
```

```
                    outFile2.print("r"+index);
                    for (int subindex=1; subindex<=MAXJOBS; subindex++)
                    {
                            outFile2.print("\t"+jobs[subindex-1].firstutilities[index-1]);
                    }
                    outFile2.println();
            }
            outFile2.close();


/*          String file7 = "deniyorum"+carew+".dat";//"output8-02-04-00-005.dat";

            FileWriter fw7 = new FileWriter (file7);
            BufferedWriter bw7 = new BufferedWriter (fw7);
            PrintWriter outFile7 = new PrintWriter (bw7);

            outFile7.println ("TotalNumberofJobs: " + "\t" + MAXJOBS);
            outFile7.println ("NumberofScheduledJobs: " + "\t" + numberscheduled);
            outFile7.println ("NumberofUnscheduledJobs: " + "\t" +
numbernotscheduled);
            outFile7.println ("TotalCost: " + "\t" + totalcost[carew-1]);
            outFile7.println ("UtilityofMachine: " + "\t" + machineutility);
            for (int index = 1; index <= MAXJOBS; index++)
            {
                    if (scheduled[index-1] == false)
                    {
                            outFile7.println ("Job" + index + " is not scheduled.");
                    }
            }

            outFile7.println ("JobIndex" + "\t" + "ReleaseTime" + "\t" + "DueDate" + "\t"
                                    + "ProcessTime" + "\t" + "StartTime" + "\t" +
"FinishTime");

            for (int index = 1; index <= MAXJOBS; index++)
            {
                    outFile7.println (index + "\t" + jobs[index-1].getReleaseTime() + "\t" +
jobs[index-1].getDueDate() + "\t"
                                            + jobs[index-1].getProcTime() + "\t" +
starttime[index-1] + "\t" + finishtime[index-1]);
            }


            outFile7.println ("JobIndex" + "\t" + "Earliness" + "\t" + "Tardiness" + "\t"
                                    + "EarlinessCost" + "\t" + "TardinessCost" + "\t"
+ "MaximumUtility");
```

```
                for (int index = 1; index <= MAXJOBS; index++)
                {
                        outFile7.println (index + "\t" + earliness[index-1] + "\t" +
tardiness[index-1] + "\t"
                                                        + earlycost[index-1] + "\t" +
tardycost[index-1] + "\t" + jobs[index-1].maxutility);
                }

                outFile7.close();*/

        }

        for (int carew=1; carew<=CORDOBA; carew++)
        {
                avgtotalutility += totalutility[carew-1];
        }
        avgtotalutility = avgtotalutility/CORDOBA;

        for (int carew=1; carew<=CORDOBA; carew++)
        {
                avgtotalcost += totalcost[carew-1];
        }
        avgtotalcost = avgtotalcost/CORDOBA;

        for (int carew=1; carew<=CORDOBA; carew++)
        {
                avgtotalutility2 += totalutility2[carew-1];
        }
        avgtotalutility2 = avgtotalutility2/CORDOBA;

        for (int carew=1; carew<=CORDOBA; carew++)
        {
                avgmachineutility += machineutility[carew-1];
        }
        avgmachineutility = avgmachineutility/CORDOBA;

        System.out.println (avgtotalutility);
        System.out.println (avgtotalcost);
        System.out.println (avgtotalutility2);
        System.out.println (avgmachineutility);

        String file1 = "output.dat";

        FileWriter fw = new FileWriter (file1);
        BufferedWriter bw = new BufferedWriter (fw);
        PrintWriter outFile = new PrintWriter (bw);
```

```
        outFile.println (avgtotalutility);
        outFile.println (avgtotalcost);
        outFile.println (avgtotalutility2);
        outFile.println (avgmachineutility);

        outFile.close();
}
}
```

Job.java

```
public class Job{

        public static final int MAXTIMESLOTS = 250;//DIKKAT ETTTTTTTTTT
        private final int CONORMALIZER = 1;
        public double maxcost;
        private int release, duedate, process;
        private double early, tardy;
        public double[] utilities = new double[MAXTIMESLOTS];
        public int[] firstutilities = new int[MAXTIMESLOTS];
        public double[] oldutilities = new double[MAXTIMESLOTS];
        public double bid;
        public double maxutility;
        public int argmaxutility;
        private double[] costs = new double[MAXTIMESLOTS];
        public int[] firstcosts = new int[MAXTIMESLOTS];

        public Job(int release1, int duedate1, int process1, double early1, double tardy1){
                release = release1;
                duedate = duedate1;
                process = process1;
                early = early1;
                tardy = tardy1;
        }

        public void FindMaxcost()
        {

                for (int index = release+1; index <= MAXTIMESLOTS-process+1; index++)
                {
                        if (index+process-1 < duedate)
                                costs[index-1] = early*(duedate-index-process+1);
                        else
                                costs[index-1] = tardy*(index+process-1-duedate);
                }

                maxcost = costs[release+1-1];
```

```
		for (int index = release+2; index <= MAXTIMESLOTS-process+1; index++)
		{
			if (costs[index-1] > maxcost)
			{
				maxcost = costs[index-1];
			}
		}
	}

	public void GenerateUtilityFunction (double[] reservationprices, double normalizer)
	{
		for(int i = 0; i < reservationprices.length; i++){
			Machine.prices[i] = reservationprices[i];
		}

		for (int index = release+1; index <= MAXTIMESLOTS-process+1; index++)
		{
			utilities[index-1] = CONORMALIZER*Machine.normalizer-
costs[index-1];

			for (int subindex = index; subindex < index+process; subindex++)
			{
				utilities[index-1] = utilities[index-1]-Machine.prices [subindex-
1];
			}
			firstutilities[index-1] = (int) utilities[index-1];
		}

		double bigcost = Machine.normalizer;
		for (int index = 1; index <= Job.MAXTIMESLOTS; index++)
		{
			if (index>=release+1 && index<=MAXTIMESLOTS-process+1)
			{
				firstcosts[index-1] = (int )costs[index-1];
			}
			else
			{
				firstcosts[index-1] = (int) bigcost;
			}

			if (firstcosts[index-1] == 0 && index+process-1!= duedate)
			{
				firstcosts[index-1] = 1;
			}
		}
	}
```

```
	public void FindMaxUtility()
	{
		maxutility = utilities[release+1-1];
		argmaxutility = release+1;

		for (int index = release+2; index <= MAXTIMESLOTS-process+1; index++)
		{
			if (maxutility <= utilities[index-1])
			{
				maxutility = utilities[index-1];
				argmaxutility = index;
			}
		}

	}

	public void PrepareBid()
	{
			bid = 0;

			for (int subindex = argmaxutility; subindex < argmaxutility+process;
subindex++)
			{
				bid = bid+Machine.prices[subindex-1];
			}

			for (int index = release+1; index < argmaxutility; index++)
			{
				if (Machine.oldstatus [argmaxutility-1] != 0 &&
					Machine.oldstatus[index-1] == Machine.oldstatus
[argmaxutility-1])
				{
					bid = bid+Machine.oldprices[index-1]-
Machine.oldreservationprices [index-1];
				}

				else
				{
					bid = bid;
				}
			}

			for (int index = argmaxutility+process; index <= MAXTIMESLOTS-
process+1; index++)
			{
				if (Machine.oldstatus [argmaxutility+process-1-1] != 0 &&
```

```
                                        Machine.oldstatus[index-1] == Machine.oldstatus
[argmaxutility+process-1-1])
                                {
                                        bid = bid+Machine.oldprices[index-1]-
Machine.oldreservationprices [index-1];
                                }

                                else
                                {
                                        bid = bid;
                                }
                        }

                        bid = bid+process*Machine.INCREMENT;
        }

        public void ReviseUtilities()
        {
                for (int index = release+1; index <= MAXTIMESLOTS-process+1; index++)
                {
                        utilities[index-1] = Machine.normalizer-costs[index-1];

                        for (int subindex = index; subindex < index+process; subindex++)
                        {
                                utilities[index-1] = utilities[index-1]-Machine.prices [subindex-
1];
                        }
                }
        }

        public void FindMaxcost2()
        {

                for (int index = release+1; index <= MAXTIMESLOTS-process+1; index++)
                {
                        if (index+process-1 < duedate)
                                costs[index-1] = early*(duedate-index-process+1);
                        else
                                costs[index-1] = tardy*(index+process-1-duedate);
                }

                maxcost = costs[release+1-1];

                for (int index = release+2; index <= MAXTIMESLOTS-process+1; index++)
                {
                        if (costs[index-1] > maxcost)
                        {
```

```
                        maxcost = costs[index-1];
                    }
                }
            }

            public void GenerateUtilityFunction2 (double[] reservationprices, double normalizer)
            {
                for(int i = 0; i < reservationprices.length; i++){
                    if (Machine.status[i] == 0)
                    {
                        Machine.prices[i] = reservationprices[i];
                    }
                }

                for(int i = 0; i < utilities.length; i++){
                    utilities[i] = 0;
                }


                for (int index = release+1; index <= MAXTIMESLOTS-process+1; index++)
                {
                    utilities[index-1] = CONORMALIZER*Machine.normalizer-
costs[index-1];

                    for (int subindex = index; subindex < index+process; subindex++)
                    {
                        if (Machine.status2[subindex-1] != 0)
                        {
                            utilities[index-1] = 0;
                            subindex = index+process;
                        }
                        else
                        {
                            utilities[index-1] = utilities[index-1]-Machine.prices
[subindex-1];
                        }
                    }
                }

            }

            public void FindMaxUtility2()
            {
                maxutility = 0;
                argmaxutility = release+1;

                for (int index = release+1; index <= MAXTIMESLOTS-process+1; index++)
```

```
		{
			if (Machine.status2[index-1] == 0)
			{
				if (maxutility <= utilities[index-1])
				{
					maxutility = utilities[index-1];
					argmaxutility = index;
				}
			}
		}

	}

	public void PrepareBid2()
	{
			bid = 0;

			for (int subindex = argmaxutility; subindex < argmaxutility+process;
subindex++)
			{
				bid = bid+Machine.prices[subindex-1];
			}

			for (int index = release+1; index < argmaxutility; index++)
			{
				if (Machine.status2[index-1] == 0)
				{
					if (Machine.oldstatus [argmaxutility-1] != 0 &&
						Machine.oldstatus[index-1] ==
Machine.oldstatus [argmaxutility-1])
					{
						bid = bid+Machine.oldprices[index-1]-
Machine.oldreservationprices [index-1];
					}

					else
					{
						bid = bid;
					}
				}
			}

			for (int index = argmaxutility+process; index <= MAXTIMESLOTS-
process+1; index++)
			{
				if (Machine.oldstatus [argmaxutility+process-1-1] != 0 &&
```

```
                                        Machine.oldstatus[index-1] == Machine.oldstatus
[argmaxutility+process-1-1])
                                {
                                        bid = bid+Machine.oldprices[index-1]-
Machine.oldreservationprices [index-1];
                                }

                                else
                                {
                                        bid = bid;
                                }
                        }

                        bid = bid+process*Machine.INCREMENT;
        }

        public void ReviseUtilities2()
        {
                for (int index = release+1; index <= MAXTIMESLOTS-process+1; index++)
                {
                        if (Machine.status2[index-1] == 0)
                        {
                                utilities[index-1] = Machine.normalizer-costs[index-1];

                                for (int subindex = index; subindex < index+process;
subindex++)
                                {
                                        if (Machine.status2[subindex-1] != 0)
                                        {
                                                utilities[index-1] = 0;
                                                subindex = MAXTIMESLOTS+2;
                                        }
                                        else
                                        {
                                                utilities[index-1] = utilities[index-1]-
Machine.prices [subindex-1];
                                        }
                                }
                        }
                }
        }

        public int getDueDate(){
                return duedate;
        }

        public int getProcTime(){
```

```
		return process;
	}

	public int getReleaseTime(){
		return release;
	}

	public double getEarly(){
		return early;
	}

	public double getTardy(){
		return tardy;
	}



	public void reportJob(){
		String message = "";
		message += "Release Time:\t" +release +" Due Date:\t"+ duedate+"\n";
		message += "Process Time:\t" +process+" Early:\t" +early+"\n";
		message += "Tardy:\t"+ tardy;
		System.out.println(message);
	}

	public void reportMaxcost()
	{
		System.out.println(maxcost);
	}

	public void reportUtility()
	{
		for (int index = 1; index <= MAXTIMESLOTS; index ++)
		{
			System.out.print(utilities[index-1] + " ");
		}

		System.out.println();
	}

}
```

Machine.java

```
public class Machine
{
```

```
	Job[] jobs;
	public static double[] reservationprices = new double[Job.MAXTIMESLOTS];
	public static final double INCREMENT = 2.5;
	public static int MAXJOBS;
	public static int[] status = new int[Job.MAXTIMESLOTS];
	public static int[] oldstatus = new int[Job.MAXTIMESLOTS];
	public static double[] prices = new double[Job.MAXTIMESLOTS];
	public static double normalizer;
	public static double[] oldprices = new double[Job.MAXTIMESLOTS];
	public static double[] oldreservationprices = new double[Job.MAXTIMESLOTS];
	public static int[] status2 = new int[Job.MAXTIMESLOTS];

	public Machine (Job[] jobs_)
	{
		jobs = jobs_;
		for(int i = 0; i<status.length; i++){
			status[i] = 0;
		}
		MAXJOBS = jobs.length;
	}

	public double[] GenerateReservationPrices ()
	{
		int[] load = new int[Job.MAXTIMESLOTS];
		for(int i = 0; i < load.length; i++){
			load[i] = 0;
		}

		for (int index = 1; index <= Job.MAXTIMESLOTS; index++)
		{
			for (int subindex = 1; subindex <= MAXJOBS; subindex++)
			{
				if (jobs[subindex-1].getDueDate() == index-1+jobs[subindex-
1].getProcTime())
				{
					for (int subsubindex = index; subsubindex <
index+jobs[subindex-1].getProcTime(); subsubindex++)
						load[subsubindex-1] = load[subsubindex-1]+1;
				}
			}

			reservationprices[index-1] = INCREMENT*load[index-1];//dikkat
et,incrementle çarpiyordun, artik çarpmiyorsun
		}
		return reservationprices;

	}
```

```
	public double FindNormalizer()
	{
		normalizer = jobs[0].maxcost;

		for (int index = 2; index <= MAXJOBS; index++)
		{
			if (jobs[index-1].maxcost > normalizer)
			{
				normalizer = jobs[index-1].maxcost;
			}
		}

		return normalizer;
	}

	public void AssignTimeSlots (int index)
	{
		int target = jobs[index-1].argmaxutility;

		for (int subindex = target; subindex < target+jobs[index-1].getProcTime();
subindex++)
		{
			status[subindex-1] = index;
		}

		for (int subindex = jobs[index-1].getReleaseTime()+1; subindex < target;
subindex++)
		{
			if (oldstatus[target-1] != 0 && oldstatus[subindex-1] ==
oldstatus[target-1])
			{
				status[subindex-1] = 0;
			}

		}

		for (int subindex = target+jobs[index-1].getProcTime(); subindex <=
Job.MAXTIMESLOTS-jobs[index-1].getProcTime()+1; subindex++)
		{
			if (oldstatus [target+jobs[index-1].getProcTime()-1-1] != 0 &&
			oldstatus[subindex-1] == oldstatus [target+jobs[index-
1].getProcTime()-1-1])
			{
				status[subindex-1] = 0;
			}
		}
```

```
		for (int subindex = 1; subindex < target; subindex++)
		{
			if (oldstatus[subindex-1] == index)
			{
				status[subindex-1] = 0;
			}
		}

		for (int subindex = target+jobs[index-1].getProcTime(); subindex <=
Job.MAXTIMESLOTS; subindex++)
		{
			if (oldstatus[subindex-1] == index)
			{
				status[subindex-1] = 0;
			}
		}
	}

	public void DeterminePrices(int indice)
	{
		int target = jobs[indice-1].argmaxutility;
		int numtarget = jobs[indice-1].getProcTime();
		double addvalue = jobs[indice-1].bid;

		for (int subindex = target; subindex < target+jobs[indice-1].getProcTime();
subindex++)
		{
			addvalue = addvalue-oldprices[subindex-1];
		}

		for (int subindex = target; subindex < target+jobs[indice-1].getProcTime();
subindex++)
		{
			prices[subindex-1] = oldprices[subindex-1] + (addvalue/numtarget);
		}

		for (int subindex = 1; subindex <= target-1; subindex++)
		{
			if (oldstatus[subindex-1] != status[subindex-1])
			{
				if (oldstatus[subindex-1] == indice)
				{
					reservationprices[subindex-1] = oldprices[subindex-1];
					prices[subindex-1] = reservationprices[subindex-1];
				}
```

```
                                else
                                {
                                        reservationprices[subindex-1] =
oldreservationprices[subindex-1];
                                        prices[subindex-1] = reservationprices[subindex-1];
                                }
                        }

                        else
                        {
                                reservationprices[subindex-1] = oldreservationprices[subindex-
1];
                                prices[subindex-1] = oldprices[subindex-1];
                        }
                }

                for (int subindex = target+jobs[indice-1].getProcTime(); subindex <=
Job.MAXTIMESLOTS; subindex++)
                {
                        if (oldstatus[subindex-1] != status[subindex-1])
                        {
                                if (oldstatus[subindex-1] == indice)
                                {
                                        reservationprices[subindex-1] = oldprices[subindex-1];
                                        prices[subindex-1] = reservationprices[subindex-1];
                                }

                                else
                                {
                                        reservationprices[subindex-1] =
oldreservationprices[subindex-1];
                                        prices[subindex-1] = reservationprices[subindex-1];
                                }
                        }

                        else
                        {
                                reservationprices[subindex-1] = oldreservationprices[subindex-
1];
                                prices[subindex-1] = oldprices[subindex-1];
                        }
                }
        }

        public double[] GenerateReservationPrices2 (boolean[] scheduled2)
        {
                int[] load = new int[Job.MAXTIMESLOTS];
```

```
		for(int i = 0; i < load.length; i++){
			load[i] = 0;
		}

		for (int index = 1; index <= Job.MAXTIMESLOTS; index++)
		{
			for (int subindex = 1; subindex <= MAXJOBS; subindex++)
			{
				if (scheduled2[subindex-1] == false)
				{
					if (jobs[subindex-1].getDueDate() == index-
1+jobs[subindex-1].getProcTime())
					{
						for (int subsubindex = index; subsubindex <
index+jobs[subindex-1].getProcTime(); subsubindex++)
							load[subsubindex-1] =
load[subsubindex-1]+1;
					}
				}
			}

			reservationprices[index-1] = INCREMENT*load[index-1];//dikkat
et,incrementle çarpiyordun, artik çarpmiyorsun
		}
		return reservationprices;

	}

	public double FindNormalizer2(boolean[] scheduled2)
	{
		normalizer = 0;

		for (int index = 1; index <= MAXJOBS; index++)
		{
			if (scheduled2[index-1] == false)
			{
				if (jobs[index-1].maxcost > normalizer)
				{
					normalizer = jobs[index-1].maxcost;
				}
			}
		}

		return normalizer;
	}

	public void AssignTimeSlots2(int index)
```

```
	{
		int target = jobs[index-1].argmaxutility;

		for (int subindex = target; subindex < target+jobs[index-1].getProcTime();
subindex++)
		{
			status[subindex-1] = index;
		}

		for (int subindex = jobs[index-1].getReleaseTime()+1; subindex < target;
subindex++)
		{
			if (status2[subindex-1] == 0)
			{
				if (oldstatus[target-1] != 0 && oldstatus[subindex-1] ==
oldstatus[target-1])
				{
					status[subindex-1] = 0;
				}
			}

		}

		for (int subindex = target+jobs[index-1].getProcTime(); subindex <=
Job.MAXTIMESLOTS-jobs[index-1].getProcTime()+1; subindex++)
		{
			if (status2[subindex-1] == 0)
			{
				if (oldstatus [target+jobs[index-1].getProcTime()-1-1] != 0 &&
				oldstatus[subindex-1] == oldstatus [target+jobs[index-
1].getProcTime()-1-1])
				{
					status[subindex-1] = 0;
				}
			}
		}

		for (int subindex = 1; subindex < target; subindex++)
		{
			if (status2[subindex-1] == 0)
			{
				if (oldstatus[subindex-1] == index)
				{
					status[subindex-1] = 0;
				}
			}
		}
```

```
		for (int subindex = target+jobs[index-1].getProcTime(); subindex <=
Job.MAXTIMESLOTS; subindex++)
		{
			if (status2[subindex-1] == 0)
			{
				if (oldstatus[subindex-1] == index)
				{
					status[subindex-1] = 0;
				}
			}
		}
	}

	public void DeterminePrices2(int indice)
	{
		int target = jobs[indice-1].argmaxutility;
		int numtarget = jobs[indice-1].getProcTime();
		double addvalue = jobs[indice-1].bid;

		for (int subindex = target; subindex < target+jobs[indice-1].getProcTime();
subindex++)
		{
			addvalue = addvalue-oldprices[subindex-1];
		}

		for (int subindex = target; subindex < target+jobs[indice-1].getProcTime();
subindex++)
		{
			prices[subindex-1] = oldprices[subindex-1] + (addvalue/numtarget);
		}

		for (int subindex = 1; subindex <= target-1; subindex++)
		{
			if (status2[subindex-1] == 0)
			{
				if (oldstatus[subindex-1] != status[subindex-1])
				{
					if (oldstatus[subindex-1] == indice)
					{
						reservationprices[subindex-1] =
oldprices[subindex-1];
						prices[subindex-1] =
reservationprices[subindex-1];
					}

					else
```

```
                                        {
                                                reservationprices[subindex-1] =
oldreservationprices[subindex-1];
                                                prices[subindex-1] =
reservationprices[subindex-1];
                                        }
                                }

                                else
                                {
                                        reservationprices[subindex-1] =
oldreservationprices[subindex-1];
                                        prices[subindex-1] = oldprices[subindex-1];
                                }
                        }
                }

                for (int subindex = target+jobs[indice-1].getProcTime(); subindex <=
Job.MAXTIMESLOTS; subindex++)
                {
                        if (status2[subindex-1] == 0)
                        {
                                if (oldstatus[subindex-1] != status[subindex-1])
                                {
                                        if (oldstatus[subindex-1] == indice)
                                        {
                                                reservationprices[subindex-1] =
oldprices[subindex-1];
                                                prices[subindex-1] =
reservationprices[subindex-1];
                                        }

                                        else
                                        {
                                                reservationprices[subindex-1] =
oldreservationprices[subindex-1];
                                                prices[subindex-1] =
reservationprices[subindex-1];
                                        }
                                }

                                else
                                {
                                        reservationprices[subindex-1] =
oldreservationprices[subindex-1];
                                        prices[subindex-1] = oldprices[subindex-1];
                                }
```

```
                }
        }
}

public boolean isChanged(){

        boolean result = false;
        int count = 0;

        for(int i = 0; i < status.length; i++)
        {
                if(status[i] == oldstatus[i])
                {
                        count++;
                }
        }

        if(count == status.length)
        {
                result = true;
        }

        return result;
}

public void assignstatus()
{
        for (int index = 1; index <= Job.MAXTIMESLOTS; index++)
        {
                status2[index-1] = status[index-1];
        }
}

public void reportStatus()
{
        for (int index = 1; index <= Job.MAXTIMESLOTS; index++)
        {
                System.out.print(status[index-1] + " ");
        }

        System.out.println();
}

public void reportPrices()
{
        for (int index = 1; index <= Job.MAXTIMESLOTS; index++)
        {
```

```
                System.out.print(prices[index-1] + " ");
            }

            System.out.println();
        }

        public void reportReservationPrices()
        {
            for (int index = 1; index <= Job.MAXTIMESLOTS; index++)
            {
                System.out.print(reservationprices[index-1] + " ");
            }

            System.out.println();
        }


}
```

APPENDIX B

Gams Code of Lagrangean Relaxation Algorithm

```
$title Lagrangian Relaxation
$stitle original model definition

sets  i   time slots
      j   jobs
      dataset data sets /dataset1 * dataset20/
      resultset /totalutilitymain, totalcostmain, optimalutility, optimalcost, averageupper,
avgsolvetimedata, avgsolvetimeutil, avgsolvetimecost/
alias (i, k, ii)
alias (j, jj)

binary variable x(i,j) assignment of i to j
       variable z      total negative utility of assignment
       variable zcost  optimal total cost of assignment
       variable zutility optimal total utility of assignment

equations capacity(i)  resource availability
          choice(j)    assignment constraint.. one resource per item
          defz         definition of negative total utility
          defzcost     definition of total cost
          defzutility  definition of total utility;

parameters  a(i,j)  utilization of resource i by item j
            f(i,j)  utility of assigning item j to resource i
            c(i,j)  cost of assigning item j to resource i
            b(i)    available resources
            process(j)    processing time of job j
            fp(i,j,dataset)  cost of assigning item j to resource i dataset
            processp(j,dataset)    processing time of job j dataset
            utilities1(i,j)
            utilities2(i,j)
```

utilities3(i,j)
utilities4(i,j)
utilities5(i,j)
utilities6(i,j)
utilities7(i,j)
utilities8(i,j)
utilities9(i,j)
utilities10(i,j)
utilities11(i,j)
utilities12(i,j)
utilities13(i,j)
utilities14(i,j)
utilities15(i,j)
utilities16(i,j)
utilities17(i,j)
utilities18(i,j)
utilities19(i,j)
utilities20(i,j)
costs1(i,j)
costs2(i,j)
costs3(i,j)
costs4(i,j)
costs5(i,j)
costs6(i,j)
costs7(i,j)
costs8(i,j)
costs9(i,j)
costs10(i,j)
costs11(i,j)
costs12(i,j)
costs13(i,j)
costs14(i,j)
costs15(i,j)
costs16(i,j)
costs17(i,j)
costs18(i,j)
costs19(i,j)
costs20(i,j)
process1(j)
process2(j)
process3(j)
process4(j)
process5(j)
process6(j)
process7(j)
process8(j)
process9(j)

```
        process10(j)
        process11(j)
        process12(j)
        process13(j)
        process14(j)
        process15(j)
        process16(j)
        process17(j)
        process18(j)
        process19(j)
        process20(j);


capacity(i)..  sum(j, sum(k$((ord(k) ge (max(ord(i)-process(j)+1, 1)))$(ord(k) le (ord(i)))),
a(k,j)*x(k,j))) =l= b(i);

choice(j)..    sum(i, x(i,j)) =e= 1;

defz..         z =e= -sum((i,j), f(i,j)*x(i,j));
defzcost..     zcost =e= sum((i,j), c(i,j)*x(i,j));
defzutility..  zutility =e= sum((i,j), f(i,j)*x(i,j));

model assign original assignment model / capacity, choice, defz /;
model costproblem cost optimization / capacity, choice, defzcost /;
model utilityproblem utility optimization / capacity, choice, defzutility /;

sets  i   time slots / r1 *r250   /
      j   jobs     / i1 * i8 /;

$Offlisting
table  a(i,j)  utilization of resource i by item j
$include "tableaij.gms"
table  utilities1(i,j) utility of job j starting at time slot i
$include "utilities1.gms"
table  utilities2(i,j) utility of job j starting at time slot i
$include "utilities2.gms"
table  utilities3(i,j) utility of job j starting at time slot i
$include "utilities3.gms"
table  utilities4(i,j) utility of job j starting at time slot i
$include "utilities4.gms"
table  utilities5(i,j) utility of job j starting at time slot i
$include "utilities5.gms"
table  utilities6(i,j) utility of job j starting at time slot i
$include "utilities6.gms"
table  utilities7(i,j) utility of job j starting at time slot i
$include "utilities7.gms"
table  utilities8(i,j) utility of job j starting at time slot i
```

```
$include "utilities8.gms"
table  utilities9(i,j) utility of job j starting at time slot i
$include "utilities9.gms"
table  utilities10(i,j) utility of job j starting at time slot i
$include "utilities10.gms"
table  utilities11(i,j) utility of job j starting at time slot i
$include "utilities11.gms"
table  utilities12(i,j) utility of job j starting at time slot i
$include "utilities12.gms"
table  utilities13(i,j) utility of job j starting at time slot i
$include "utilities13.gms"
table  utilities14(i,j) utility of job j starting at time slot i
$include "utilities14.gms"
table  utilities15(i,j) utility of job j starting at time slot i
$include "utilities15.gms"
table  utilities16(i,j) utility of job j starting at time slot i
$include "utilities16.gms"
table  utilities17(i,j) utility of job j starting at time slot i
$include "utilities17.gms"
table  utilities18(i,j) utility of job j starting at time slot i
$include "utilities18.gms"
table  utilities19(i,j) utility of job j starting at time slot i
$include "utilities19.gms"
table  utilities20(i,j) utility of job j starting at time slot i
$include "utilities20.gms"

table  costs1(i,j) cost of job j starting at time slot i
$include "costs1.gms"
table  costs2(i,j) cost of job j starting at time slot i
$include "costs2.gms"
table  costs3(i,j) cost of job j starting at time slot i
$include "costs3.gms"
table  costs4(i,j) cost of job j starting at time slot i
$include "costs4.gms"
table  costs5(i,j) cost of job j starting at time slot i
$include "costs5.gms"
table  costs6(i,j) cost of job j starting at time slot i
$include "costs6.gms"
table  costs7(i,j) cost of job j starting at time slot i
$include "costs7.gms"
table  costs8(i,j) cost of job j starting at time slot i
$include "costs8.gms"
table  costs9(i,j) cost of job j starting at time slot i
$include "costs9.gms"
table  costs10(i,j) cost of job j starting at time slot i
$include "costs10.gms"
table  costs11(i,j) cost of job j starting at time slot i
```

```
$include "costs11.gms"
table  costs12(i,j) cost of job j starting at time slot i
$include "costs12.gms"
table  costs13(i,j) cost of job j starting at time slot i
$include "costs13.gms"
table  costs14(i,j) cost of job j starting at time slot i
$include "costs14.gms"
table  costs15(i,j) cost of job j starting at time slot i
$include "costs15.gms"
table  costs16(i,j) cost of job j starting at time slot i
$include "costs16.gms"
table  costs17(i,j) cost of job j starting at time slot i
$include "costs17.gms"
table  costs18(i,j) cost of job j starting at time slot i
$include "costs18.gms"
table  costs19(i,j) cost of job j starting at time slot i
$include "costs19.gms"
table  costs20(i,j) cost of job j starting at time slot i
$include "costs20.gms"

parameters process1(j) process times of jobs
$include "process1.gms"
parameters process2(j) process times of jobs
$include "process2.gms"
parameters process3(j) process times of jobs
$include "process3.gms"
parameters process4(j) process times of jobs
$include "process4.gms"
parameters process5(j) process times of jobs
$include "process5.gms"
parameters process6(j) process times of jobs
$include "process6.gms"
parameters process7(j) process times of jobs
$include "process7.gms"
parameters process8(j) process times of jobs
$include "process8.gms"
parameters process9(j) process times of jobs
$include "process9.gms"
parameters process10(j) process times of jobs
$include "process10.gms"
parameters process11(j) process times of jobs
$include "process11.gms"
parameters process12(j) process times of jobs
$include "process12.gms"
parameters process13(j) process times of jobs
$include "process13.gms"
parameters process14(j) process times of jobs
```

```
$include "process14.gms"
parameters process15(j) process times of jobs
$include "process15.gms"
parameters process16(j) process times of jobs
$include "process16.gms"
parameters process17(j) process times of jobs
$include "process17.gms"
parameters process18(j) process times of jobs
$include "process18.gms"
parameters process19(j) process times of jobs
$include "process19.gms"
parameters process20(j) process times of jobs
$include "process20.gms"



parameters b(i) available resources
$include "tablebi.gms"

$stitle Relaxed Problem Definition and Subgradient Optimization
* Lagrangian subproblem definition
* uses dynamic set to define WHICH knapsack to solve

sets iter   subgradient iteration index / iter1 * iter20  /


parameters  w(j)   Lagrangian multipliers
        improv has the Lagrangian bound improved over the previous iterations
        zbest(dataset) value of best feasible solution
        costutility(dataset) cost corresponding to best feasible solution
        upperzlbest(dataset) best upper value
        optutility(dataset) optimal utility for each dataset
        optcost(dataset) optimal cost for each dataset
        summary(resultset) summary of the results
        solvetimeiter(iter) solution time at each iteration
        solvetimedata(dataset) solution time of each data set
        solvetimeutil(dataset) solution time of each optimal utility
        solvetimecost(dataset) solution time of each optimal cost

variable  zlrx    relaxed objective

equations knapsack(i) capacity with dynamic sets
       defzlrx     definition of zlrx;

knapsack(i)..  sum(j, sum(k$((ord(k) ge (max(ord(i)-process(j)+1, 1)))$(ord(k) le (ord(i)))),
a(k,j)*x(k,j))) =l= b(i);
```

```
defzlrx..       zlrx =e= -sum((i,j), (f(i,j)+w(j))*x(i,j));

model pknap / knapsack, defzlrx /;

scalars target target objective function value
        alpha  step adjuster / 1 /
        norm   norm of slacks
        step   step size for subgradient / na /
        zfeas  value for best known solution or valid upper bound
        zlr    Lagrangian objective value
        zl     Lagrangian objective value
        zlbest current best Lagrangian lower bound
        count  count of iterations without improvement
        reset  reset count counter / 5 /
        tol    termination tolerance / 1e-5 /
        status outer loop status /0/
        counter count of infeasible constraints
        zfes   value of feasible solution
        zbestdummy dummy zbest
        counter1 count of repeating assignments
        counter2 count of infeasible capacity constraints
        bool   boolean variable
        totalutility average total utility
        totalcost average total cost corresponding to totalutility
        totaloptutility average optimal total utility
        totaloptcost average optimal total cost
        totalupper average of Lagrangean upper bound
        totalsolvetimedata average time to solve Lagrangean problem
        totalsolvetimeutil average time to solve utilization optimization problem
        totalsolvetimecost average time to solve cost optimization problem

parameters
   s(j)             slack variable
   report(iter,*)   iteration log
   xrep(j,i,*)      x iteration report
   srep(iter,j)     slack report
   wrep(iter,j)     w iteration report
   xx(i,j)          binary storage to show the starting time of job j
   xstart(j)        starting time of job j;

option   mip = default
        rmip = default;

file results writes iteration report / solution /;
loop (dataset,

f(i,j)$(ord(dataset)=1) = utilities1(i,j);
```

```
f(i,j)$(ord(dataset)=2) = utilities2(i,j);
f(i,j)$(ord(dataset)=3) = utilities3(i,j);
f(i,j)$(ord(dataset)=4) = utilities4(i,j);
f(i,j)$(ord(dataset)=5) = utilities5(i,j);
f(i,j)$(ord(dataset)=6) = utilities6(i,j);
f(i,j)$(ord(dataset)=7) = utilities7(i,j);
f(i,j)$(ord(dataset)=8) = utilities8(i,j);
f(i,j)$(ord(dataset)=9) = utilities9(i,j);
f(i,j)$(ord(dataset)=10) = utilities10(i,j);
f(i,j)$(ord(dataset)=11) = utilities11(i,j);
f(i,j)$(ord(dataset)=12) = utilities12(i,j);
f(i,j)$(ord(dataset)=13) = utilities13(i,j);
f(i,j)$(ord(dataset)=14) = utilities14(i,j);
f(i,j)$(ord(dataset)=15) = utilities15(i,j);
f(i,j)$(ord(dataset)=16) = utilities16(i,j);
f(i,j)$(ord(dataset)=17) = utilities17(i,j);
f(i,j)$(ord(dataset)=18) = utilities18(i,j);
f(i,j)$(ord(dataset)=19) = utilities19(i,j);
f(i,j)$(ord(dataset)=20) = utilities20(i,j);

c(i,j)$(ord(dataset)=1) = costs1(i,j);
c(i,j)$(ord(dataset)=2) = costs2(i,j);
c(i,j)$(ord(dataset)=3) = costs3(i,j);
c(i,j)$(ord(dataset)=4) = costs4(i,j);
c(i,j)$(ord(dataset)=5) = costs5(i,j);
c(i,j)$(ord(dataset)=6) = costs6(i,j);
c(i,j)$(ord(dataset)=7) = costs7(i,j);
c(i,j)$(ord(dataset)=8) = costs8(i,j);
c(i,j)$(ord(dataset)=9) = costs9(i,j);
c(i,j)$(ord(dataset)=10) = costs10(i,j);
c(i,j)$(ord(dataset)=11) = costs11(i,j);
c(i,j)$(ord(dataset)=12) = costs12(i,j);
c(i,j)$(ord(dataset)=13) = costs13(i,j);
c(i,j)$(ord(dataset)=14) = costs14(i,j);
c(i,j)$(ord(dataset)=15) = costs15(i,j);
c(i,j)$(ord(dataset)=16) = costs16(i,j);
c(i,j)$(ord(dataset)=17) = costs17(i,j);
c(i,j)$(ord(dataset)=18) = costs18(i,j);
c(i,j)$(ord(dataset)=19) = costs19(i,j);
c(i,j)$(ord(dataset)=20) = costs20(i,j);

process(j)$(ord(dataset)=1) = process1(j);
process(j)$(ord(dataset)=2) = process2(j);
process(j)$(ord(dataset)=3) = process3(j);
process(j)$(ord(dataset)=4) = process4(j);
process(j)$(ord(dataset)=5) = process5(j);
process(j)$(ord(dataset)=6) = process6(j);
```

```
process(j)$(ord(dataset)=7) = process7(j);
process(j)$(ord(dataset)=8) = process8(j);
process(j)$(ord(dataset)=9) = process9(j);
process(j)$(ord(dataset)=10) = process10(j);
process(j)$(ord(dataset)=11) = process11(j);
process(j)$(ord(dataset)=12) = process12(j);
process(j)$(ord(dataset)=13) = process13(j);
process(j)$(ord(dataset)=14) = process14(j);
process(j)$(ord(dataset)=15) = process15(j);
process(j)$(ord(dataset)=16) = process16(j);
process(j)$(ord(dataset)=17) = process17(j);
process(j)$(ord(dataset)=18) = process18(j);
process(j)$(ord(dataset)=19) = process19(j);
process(j)$(ord(dataset)=20) = process20(j);


* --- calculate initial Lagrangian multipliers
*     There are many possibl ways to find initial multipliers.
*     The choice of initial multipliers is very important for the
*     overall performance. The marginals of the relaxed problem
*     are often used to initialize the multipliers. Another choice
*     is simply to start with zero multipliers.


* replace 'default' with solver of your choice.




put results 'solvers used: RMIP = ' system.rmip /
          '              MIP = ' system.mip  /;

* --- solve relaxed problem to get initial multipliers
*     Note that different solvers get different dual solutions
*     which are not as good as a zero set of initila multipliers.

solve assign minimizing z using rmip;
option solprint = off;
put / 'RMIP objective value = ', z.l:12:6/;

if(assign.modelstat = 1,
  status = 1                    ! everything ok
else
  abort '*** relaxed MIP not optimal',
        '    no subgradient iterations', x.l );
```

```
xrep(j,i,'initial') = x.l(i,j);

zlbest = z.l;

* --- use RMIP duals
w(j)   = choice.m(j);

* --- use optimal duals
*w(j) = wopt(j);

* --- use zero starting point
*w(j) = 0;
*zlbest=0;

put // 'zlbest                  objective value  = ', zlbest:12:6;
put // "Dual values on assignment constraint"/ ;
loop(j, put / "w('",j.tl,"') =  ", w(j):16:6 ";" );

*  one needs a value for zfeas
*  one can compute a valid upper bound as follows:
$ontext
zfeas  = sum(j, smax(i, f(i,j)));
put // 'zfeas quick and dirty bound obj value      = ', zfeas:12:6;
display 'a priori upper bound',zfeas;
$offtext

*  another alternative to compute a value for zfeas is
*  to solve gapmin by B-B and stop
*  at first 0-1 feasible solution found
*  using gapmin.optcr = 1, as follows

assign.optcr=1;assign.solprint=2;

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

solve assign minimizing z using mip;
option solprint = off;
zfeas=z.l;
display 'final zfeas',zfeas;
display 'heuristic solution by B-B ',x.l,z.l;
put  / 'zfeas IP solution bound objective value    = ', zfeas:12:6;


put /// 'Iteration         New Bound   Previous Bound           norm      abs(zl-zf)'/;


*  then keep the smaller of the two values as zfeas
```

```
pknap.optcr = 0;     ! ask for global solution
pknap.solprint = 2;  ! turn off all solution output

*=====================================================================
===========*
*                                                    *
*  beginning of subgradient loop                                 *
*                                                    *
*=====================================================================
===========*

count  = 1;
alpha  = 1;

display status;
zbest(dataset) = 0;

loop(iter$(status = 1),    ! i.e., repeat while status is 1

*  solve Lagrangian subproblems by solving nonoverlapping knapsack
*  poblems. Note the use of the dynamic set id(i) which will
*  contain the current knapsack descriptor.

     solve pknap using mip minimizing zlrx;
     option solprint = off;
     zlr = zlrx.l;
     xx(i,j) = x.l(i,j);
     xstart(j) = sum(i, xx(i,j)*ord(i))-1;
     solvetimeiter(iter) = pknap.resusd;

counter = 0;
loop(j, if(sum(i, x.l(i,j)) = 1, counter = counter +1));
if((counter = 0), zfes = sum((i,j), f(i,j)*x.l(i,j)));

loop(j,
    counter1 = 0;
    loop(i, if(xx(i,j)=1,
          counter1 = counter1+1;
          if(counter1>1, xx(i,j) = 0))));

loop(j, if(xstart(j)=-1,
       bool = 0;
       loop(i$(bool=0),
          counter2 = 0;
          bool = 0;
          xx(i,j)=1;
```

```
          loop(ii, if(sum(jj, sum(k$((ord(k) ge (max(ord(ii)-process(jj)+1, 1)))$(ord(k) le
(ord(ii)))), a(k,jj)*xx(k,jj))) > b(ii), counter2 = counter2+1));
          if(counter2>0, xx(i,j)=0);
          if(counter2=0, bool=1);));));

xstart(j) = sum(i, xx(i,j)*ord(i))-1;
zfes = sum((i,j), f(i,j)*xx(i,j));

zbestdummy = zbest(dataset);
zbest(dataset) = max(zbest(dataset), zfes);

costutility(dataset)$(zbest(dataset)<>zbestdummy) = sum((i,j), c(i,j)*xx(i,j));

$Ontext
    id(ii) = no );                  ! make set empty again
$Offtext

  improv = 0;
  zl     = zlr + sum(j, w(j));
  improv$(zl > zlbest) = 1;             ! is zl better than zlbest?
  zlbest = max(zlbest,zl);
  s(j)   = 1 - sum(i, x.l(i,j));        ! subgradient
  norm   = sum(j, sqr(s(j)));

  status$(norm < tol)             = 2;
  status$(abs(zlbest-zfeas) < 1e-4) = 3;
  status$(pknap.modelstat <> 1)     = 4;
  put results / iter.tl ,zl:16:6,zlbest:16:6,norm:16:6,abs(zlbest-zfeas):16:6;
  if((status = 2),
    put //"subgr. method has converged, status = ",status:5:0//;
    put //"last solution found is optimal for IP problem"//;
   );    ! end if
  if((status = 3),
    put //"subgr. method has converged, status = ",status:5:0//;
    put //"no duality gap, best sol. found is optimal "//;
  );     ! end if
  if ((status = 4),
    put //"something wrong with last Lag. subproblem"//;
    put //"status = ",status:5:0//;
  );     ! end if

  report(iter,'zlr')    = zlr;
  report(iter,'zl')     = zl;
  report(iter,'zlbest') = zlbest;
  report(iter,'norm')   = norm;
  report(iter,'step')   = step;
*   display zfes;
```

```
*   display zbest;
*   display xx;
*   display xstart;

   wrep(iter,j) = w(j);
   srep(iter,j) = s(j);
   xrep(j,i,iter) = x.l(i,j);

   if(status=1,
      target = (zlbest+zfeas)/2;
      step   = (alpha*(target-zl)/norm)$(norm > tol);
      w(j)   = w(j)+step*s(j);
      if(count>reset,          ! too many iterations w/o improvement
        alpha = alpha/2;
        count = 1
      else if(improv,          ! reset count if improvement
            count = 1
          else
            count = count +1 ! update count if no improvement
          )
       )
    )
);                                  ! end loop iter

*display report, wrep, srep, xrep;
display zfeas;
display zbest;
upperzlbest(dataset) = zlbest;
put results // "Dual values on assignment constraint" /;
loop(j, put /  "w('",j.tl,"') =  ", w(j):16:6  ";" )
put //"best Lagrangian bound   =   ",zlbest:10:5;

solve utilityproblem maximizing zutility using mip;
option solprint = off;
solve costproblem minimizing zcost using mip;
option solprint = off;
optutility(dataset) = zutility.l;
optcost(dataset) = zcost.l;

solvetimedata(dataset) = sum(iter, solvetimeiter(iter));
solvetimeutil(dataset) = utilityproblem.resusd;
solvetimecost(dataset) = costproblem.resusd;
);

totalutility = (sum(dataset, zbest(dataset)))/(card(dataset));
totalcost = (sum(dataset, costutility(dataset)))/(card(dataset));
totaloptutility = (sum(dataset, optutility(dataset)))/(card(dataset));
```

```
totaloptcost = (sum(dataset, optcost(dataset)))/(card(dataset));
totalupper = -(sum(dataset, upperzlbest(dataset)))/(card(dataset));

totalsolvetimedata = (sum(dataset, solvetimedata(dataset)))/(card(dataset));
totalsolvetimeutil = (sum(dataset, solvetimeutil(dataset)))/(card(dataset));
totalsolvetimecost = (sum(dataset, solvetimecost(dataset)))/(card(dataset));

summary('totalutilitymain') = totalutility;
summary('totalcostmain') = totalcost;
summary('optimalutility') = totaloptutility;
summary('optimalcost') = totaloptcost;
summary('averageupper') = totalupper;

summary('avgsolvetimedata') = totalsolvetimedata;
summary('avgsolvetimeutil') = totalsolvetimeutil;
summary('avgsolvetimecost') = totalsolvetimecost;

display summary;
```