COMBINATORIAL AUCTION BASED RESOURCE CO-ALLOCATION MODEL FOR GRIDS

by

Ali Haydar Özer

B.S. in Computer Engineering, Boğaziçi University, 2002

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University

2004

COMBINATORIAL AUCTION BASED RESOURCE CO-ALLOCATION MODEL FOR GRIDS

APPROVED BY:

Assoc. Prof. Can Özturan
(Thesis Supervisor)

Prof. Cem Ersoy

Assist. Prof. Erdoğan Sevilgen

DATE OF APPROVAL: 22.07.2004

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor Assoc. Prof. Can Özturan, for his kindness, guidance, encouragement and support throughout the whole study. I thank Prof. Cem Ersoy and Assist. Prof. Erdoğan Sevilgen for participating in my thesis jury.

I would like to thank my dear friends for being with me whenever I need. Without the beautiful times we spent together, this thesis would never be complete.

Finally, I would like to thank my beloveds, my family for their endless love and support. They taught me love as the meaning of existence.

ABSTRACT

COMBINATORIAL AUCTION BASED RESOURCE CO-ALLOCATION MODEL FOR GRIDS

Resource co-allocation problem is one of the challenging problems in grids. In order to model this problem, a new combinatorial auction based resource co-allocation (CABRC) approach is proposed. This economy based model provides efficient allocation of resources in a grid environment by allowing bidders to submit bids on the combinations of different resource types. In order to solve the model, CABRC problem is defined and formulated using integer programming. It is proved that CABRC problem is NP-hard and since optimum solutions may take tremendous amount of time to be found, two new greedy heuristics based on price per unit criteria are proposed. A software package that consists of an artificial test case generator, an optimum solver, an upper bound estimator and three greedy heuristic solvers for CABRC problem is coded. Since there is no real world data for testing the solvers, performance of algorithms are compared using a comprehensive test suite which is produced by the test case generator. Proposed two polynomial time heuristic solvers produce promising results of 97.3 per cent and 99.2 per cent average performance relative to the optimum solution respectively.

ÖZET

ŞEBEKELER İÇİN BİRLEŞİMSEL MÜZAYEDE TABANLI KAYNAK TAHSİS MODELLERİ

Kaynak tahsis problemi, bilişim şebekelerinin etkin olarak çözülememiş problemlerinden biridir. Bu problemin modellenmesi doğrultusunda bilişim şebekeleri için yeni bir birleşimsel müzayede tabanlı kaynak tahsis (BMTKT) yaklaşımı önerilmiştir. Bu ekonomi tabanlı model müşterilere ayrım yapmaksızın, istedikleri kaynak tiplerine birleşimsel teklif vermelerine imkan vererek, bilişim şebeklerine ait kaynakların etkili ve ekonomik açıdan verimli tahsisine olanak sağlamaktadır. Bu modeli çözmek için öncelikle BMTKT problemi tanımlanmış ve problem tamsayı programlama metodu kullanılarak formüle edilmiştir. Bu problem NP-hard sınıfına ait bir problem olduğundan dolayı optimum çözümü bulmak çok fazla zaman alabileceğinden, birim fiyat kriteri tabanlı iki yaklaşık sonuç algoritması önerilmiştir. BMTKT modeli için, içinde suni test üreticisi, optimum çözücüsü, bir üst sınır hesaplayıcısı ve üç adet yaklaşık sonuç çözücüsü bulunduran bir yazılım paketi hazırlanmıştır. Elimizde bu modele ait gerçek hayat verileri olmadığından, algoritmaların performansları test üreticisi tarafından oluşturulan kapsamlı testler ile sınanmıştır. Önerilmiş olan iki polinom zamanlı yaklaşık sonuç algoritması optimum sonuçlara göre yüzde 97,3 ve yüzde 99,2'lik ortalama sonuçlarla ümit verici performans değerleri vermiştir.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii
ABSTRACT iv
ÖZET v
LIST OF FIGURES viii
LIST OF TABLES x
LIST OF SYMBOLS/ABBREVIATIONS xii
1. INTRODUCTION 1
1.1. Contributions in the Thesis 3
2. AUCTION MODELS 5
2.1. Sequential Auction Model and Common Auction Forms 5
2.2. Combinatorial Auction Model (Single Unit) 7
2.2.1. Definition of CA Problem 8
2.2.2. CA Example 9
2.3. Multi-Unit Combinatorial Auction Model 10
2.3.1. Definition of MUCA Problem 10
2.3.2. MUCA Example 11
2.4. Multi-Unit Nondiscriminatory Combinatorial Auction Model 12
3. CA BASED RESOURCE CO-ALLOCATION MODEL 13
3.1. Definition of CABRC Problem 13
3.2. Formulation of CABRC Problem 14
3.3. Theorems Related to CABRC Problem 15
3.4. Applying CABRC Model to Grids 16
3.4.1. Economy Based Approach 16
3.4.2. Priority Based Approach 17
3.4.3. Identifying Resources in a Grid Environment 17
3.5. CABRC Problem Example 18
3.5.1. Exclusive-OR Bids 20
4. CABRC PROBLEM SOLVER 21
4.1. Optimum Solver 21

4.2. Linear Relaxation Based Upper Bound Estimator 21
4.3. Linear Relaxation Based Greedy Heuristic Solver 22
4.3.1. Checking Feasibility of Given Bid Set 23
4.4. Greedy Heuristic Solver Based on Price Per Unit Criteria 25
4.5. Enhanced Heuristic Solver Based on Price Per Unit Criteria 26
5. CABRC PROBLEM TEST CASE GENERATOR 29
5.1. Determining Requested Resource Types in a Subbid 30
5.2. Determining the Price of a Bid 30
5.3. Distributions Used in the Generator 31
6. EXPERIMENTAL RESULTS 32
6.1. Phase 1: Effect of Parameters 33
6.1.1. Number of Resource Types (m) 33
6.1.2. Set of Units of Resources (u) 34
6.1.3. Number of Bids (n) 36
6.1.4. Number of Subbids for Bids (t) 37
6.1.5. Size of Requested Subset of Resources (s) 38
6.1.6. Method for Determining Elements of s_{jk} ($s_{jk}method$) 39
6.1.7. Requested Amount of Resources for Each Subbid (q) 40
6.1.8. Price Factor for ORed Requests Inside a Subbid (or_factor) . . 40
6.1.9. Price Factor for ANDed Subbids Inside a Bid (and_factor) . . . 41
6.1.10. Variance for Price ($price_variance$) 42
6.1.11. Conclusion of Phase 1 43
6.2. Phase 2: General Performance 44
7. CONCLUSION 46
APPENDIX A: DISTRIBUTIONS USED IN THE GENERATOR 47
APPENDIX B: TABLES OF TEST RESULTS 49
APPENDIX C: CONFIGURATIONS OF TESTS 75
REFERENCES 78

LIST OF FIGURES

Figure 4.1. The pseudocode for LRS algorithm 23

Figure 4.2. Network flow diagram for checking feasibility of bids 24

Figure 4.3. The pseudocode for PS algorithm 26

Figure 4.4. The pseudocode for EPS algorithm 28

Figure 5.1. The pseudocode for determining the price of a bid 31

Figure 6.1. Goodness of solutions and running times: (m) 34

Figure 6.2. Goodness of solutions and running times: (u) 35

Figure 6.3. Goodness of solutions and running times: (n) 36

Figure 6.4. Goodness of solutions and running times: (t) 37

Figure 6.5. Goodness of solutions and running times: (s) 38

Figure 6.6. Goodness of solutions and running times: $(s_{jk}method)$ 39

Figure 6.7. Goodness of solutions and running times: (q) 40

Figure 6.8. Goodness of solutions and running times: (or_factor) 41

Figure 6.9. Goodness of solutions and running times: (and_factor) 42

Figure 6.10. Goodness of solutions and running times: $(price_variance)$ 43

Figure 6.11. Overall goodness of solutions 45

Figure 6.12. Overall running times of solutions 45

LIST OF TABLES

Table B.1. Goodness of solutions (%): m 49

Table B.2. Running times (s): m 50

Table B.3. Goodness of solutions (%): u 51

Table B.4. Running times (s): u 52

Table B.5. Goodness of solutions (%): n 53

Table B.6. Running times (s): n 54

Table B.7. Goodness of solutions (%): t 55

Table B.8. Running times (s): t 56

Table B.9. Goodness of solutions (%): s 57

Table B.10. Running times (s): s 58

Table B.11. Goodness of solutions (%): $s_{jk}method$ 59

Table B.12. Running times (s): $s_{jk}method$ 60

Table B.13. Goodness of solutions (%): q 61

Table B.14. Running times (s): q 62

Table B.15. Goodness of solutions (%): or_factor 63

Table B.16. Running times (s): *or_factor* 64

Table B.17. Goodness of solutions (%): *and_factor* 65

Table B.18. Running times (s): *and_factor* 66

Table B.19. Goodness of solutions (%): *price_variance* 67

Table B.20. Running times (s): *price_variance* 68

Table B.21. Goodness of solutions (%): general-1 69

Table B.22. Running times (s): general-1 70

Table B.23. Goodness of solutions (%): general-2 71

Table B.24. Running times (s): general-2 72

Table B.25. Goodness of solutions (%): general-3 73

Table B.26. Running times (s): general-3 74

Table C.1. Base configuration for uniformly distributed test files 75

Table C.2. Base configuration for normally distributed test files 76

Table C.3. Base configuration for exponentially distributed test files 77

LIST OF SYMBOLS/ABBREVIATIONS

B	Set of bids
b_i	Bid i
m	Number of resource types
n	Number of bids
p_j	Offered price for bid j
q_{jk}	Requested quantity of resources for the set s_{jk}
R	Set of resource types
r_i	Resource type i
s_{jk}	Set of requested substitutable resource types
t_j	Number of subbids in bid j
U	Set of units of resources
u_i	Number of available units of resource type i
v	Maximum number of subbids among all bids
α	Price factor for ORed resource requests inside a subbid
β	Price factor for ANDed subbids inside a bid
CA	Combinatorial Auction
CABRC	Combinatorial Auction Based Resource Co-allocation
EPS	Enhanced Heuristic Solver Based on Price Per Unit Criteria
FCC	Federal Communications Commission
GSL	GNU Scientific Library
IP	Integer Programming
LRE	Linear Relaxation Based Upper Bound Estimator
LRS	Linear Relaxation Based Greedy Heuristic Solver
MUCA	Multi-Unit Combinatorial Auction
MUNCA	Multi-Unit Nondiscriminatory Combinatorial Auction
OGSA	Open Grid Services Architecture
OPT	Optimum Solver

PS	Greedy Heuristic Solver Based on Price Per Unit Criteria
SMR	Simultaneous Multiple-Round

1. INTRODUCTION

Recently much research has been conducted on conventional distributed computing. By the time wide area network technology has evolved, this brought attention to distributed computing across wide area networks. A new paradigm *grid* emerged in mid1990s as a new branch of distributed computing [1]. The intention of this paradigm is to interconnect computational resources all over the world like electrical power grids and to make each application draw computational power from the grid by plugging into the grid. Technically the term *grid* refers to an infrastructure that offers multiple computation, data, or service resources owned by different organizations and spread over a geographical region. The main purpose of this infrastructure is to enable collaborative use of these resources through *virtual organizations* which consist of rules that control accesses for shared resources. Although various distributed computing technologies exist, they are not suitable for coordinated resource sharing among geographically separated organizations. Grid technologies are aimed at finding a solution for the problem of coordinated resource sharing and problem solving in virtual organizations [2].

Grid systems can be grouped into three major category [3]. *Computational grids* consist of interconnected computational resources such as supercomputers, clusters of computers and are mainly used for solving computationally difficult problems. *Data grids* consist of interconnected data repositories such as data warehouses and digital libraries in order to process and mine large amount of data that cannot be handled by local data repositories and computers. Finally, *service grids* are constructed for providing services that any single machine cannot provide alone.

Most attention has been paid to system level design and implementation of grid software. One of the well known grid technology solution is *Globus* [4] which is based on *Open Grid Services Architecture* (OGSA) [5]. It is a solution based on open source, open architecture protocols, services and software for grids and applications. As of May 2004, current version of Globus provides services and software libraries for security, information infrastructure, resource management, data management, communication,

fault detection and portability.

Although academic and non-profit institutions may support grids by sharing their resources for the common good, this is not the case for companies which hold large amount of computational resources. So in order to make grid concept pervasive in the capitalist world we live in, market models for grids must be researched in correspondence with grid services and software. Wolski *et al.* introduced the term *G-commerce* as the problem of dynamic resource allocation on the grid in terms of computational market economies [6]. Both commodities market and auction based models are discussed and it is claimed that commodities market based models are more suitable for grids. However, the auction mechanism described is a form of sequential auction, in which each item or indivisible bundle of items in the auction is auctioned one at a time. Since sequential auction models do not support combinatorial bidding, synergies between resources cannot be expressed. Therefore sequential auctions may not be suitable for grids, but this is not the case for combinatorial auctions.

In grid environments, *resource co-allocation* problem is defined as the provision of allocation, configuration, and monitoring/control functions for the resource ensemble required by a single application [7]. As a part of OGSA, several studies have been conducted for resource co-allocation problem at system level [7, 8].

One of the challenging problems related to resource co-allocation problem is the discovery of resources and considering their allocation policies in especially dynamic grid environments. Raman *et al.* proposed a matchmaking framework for addressing these problems [9, 10]. In this framework, both resource suppliers and customers advertise characteristics of their resources and resource requests respectively to an advertising agent. These advertisements are called classified advertisements. Advertising agents match the classified advertisements of resource requests with the classified advertisements of available resources using a matchmaking algorithm called *Gangmatching*. This algorithm is basically a searching algorithm in which resource requests are tried to be satisfied by searching classified advertisements of resources. Since the requests are processed one at a time, this algorithm may provide inefficient allocation of resources

when multiple resource types are requested.

In this study, we consider resource co-allocation problem at user level instead of system level and define the problem as efficient allocation of available resources among multiple users when the users need multiple units of different resources types in a grid environment. We are mainly interested in economic perspective of resource co-allocation problem in order to encourage resource holders to supply their resources to users, therefore efficiency criteria will be based on the profit gained by resource suppliers. However the model is also applicable for non-profit grid environments such as academic grids.

In this thesis, we propose a new combinatorial auction (CA) based resource co-allocation model for grids. In Chapter 2, we go over currently available auction models and introduce a new combinatorial auction model called multi-unit nondiscriminatory combinatorial auction (MUNCA) model. This model provides economically efficient allocation of resources by allowing bidders to submit bids on the combinations of available resource types. The model also supports declaring substitutable resource types inside the bids. In Chapter 3, we apply MUNCA model to grid environment. We first define combinatorial auction based resource co-allocation (CABRC) problem, give integer programming (IP) formulation and prove that CABRC problem is NP-hard. In Chapter 4 and 5, we describe the CABRC problem solver and test case generator in detail. Our solver package consists of an optimum solver, a linear relaxation based upper bound estimator, a linear relaxation based greedy heuristic solver and two new greedy heuristic solvers based on price per unit criteria. In Chapter 6, we present the results of the tests conducted for measuring performance of the solvers especially the two new greedy heuristic solvers and discuss the result afterwards.

1.1. Contributions in the Thesis

Main contributions in this thesis are given in the following list:

- A new combinatorial auction model MUNCA is presented. This model allows

bidders to bid on multiple instances of items and to declare their preferences of substitutable items. This results in economically efficient allocation of items from both bidders' and sellers' perspective.

- We present CABRC model which is the application of MUNCA model to resource co-allocation problem in grids. We define CABRC problem and its IP formulation. We discuss both economy and priority based applications of the model.
- We prove that CABRC problem is NP-hard.
- We propose two polynomial time greedy heuristic solvers based on price per unit criteria and code solver package for CABRC model. Besides these heuristic solvers, this package also contains an optimum solver based on CPLEX [11] mixed integer solver, a linear relaxation based upper bound estimator and a linear relaxation based greedy heuristic solver.
- We code CABRC problem test case generator. The generator is fully flexible and all ten parameters of CABRC model can be configured separately. The generator is also capable of producing test cases for full-factorial testing.
- In order to measure the performance of solvers, we conduct comprehensive tests based on the artificial test suite produced by the generator. In the first phase of the tests, we examine the effect of each parameter of the model on the solvers separately and in the second phase, we measure the general performance of solvers based on more than one thousand instances of CABRC problem.
- According to the test results, we conclude that the two proposed polynomial-time greedy heuristics run quite fast with respect to optimum solver and produce results of 97.3 per cent and 99.2 per cent performance on average relative to the optimum solution.

2. AUCTION MODELS

An auction is a market institution with an explicit set of rules determining resource allocation and prices on the basis of bids from the market participants. Although precise origin of auctions and auctioning is not known, it is believed that first auction began with the abuse of woman more than 2500 years ago. The Greek historian, Herodotus, described Babylonian annual auctions of young women to be sold for the purpose of marriage. However in these auctions, instead of starting with low prices and going up like well-known contemporary auctions, the offers for women started from high prices and lowered until a bidder accepted the woman for marriage. After Babylonians, auctions have been used in the times of the Romans as a system of commercial trade. After a battle, captured prisoners and goods were being auctioned by agents. In 193 AD, Roman Empire collapsed and probably most interesting auction in the history occurred. Entire Roman Empire was auctioned as one lot. Diduis Julianus, the winner of the auction, was proclaimed Emperor. However after a short period of time, a terrible instance of *winner's curse* occurred and he was beheaded [12]. Although history of auctions goes back to 500 BC, most of the research related to auction models have been done in the twentieth century. With the increase in computational powers of computers, more complex and efficient auction models such as combinatorial auctions have been introduced.

In order to solve resource co-allocation problem in grids, we constructed a new combinatorial auction model called MUNCA model and adapted the new model to grid environment. In the remaining parts of this chapter, we will first describe previous auction models and then present MUNCA model briefly which is the basis of our CABRC model.

2.1. Sequential Auction Model and Common Auction Forms

In the sequential auction model, each item or indivisible bundle of items in the auction is auctioned one at a time. Winner determination is done simply by picking

the highest bidder for each item. Because of its simplicity, this model has been widely used throughout the history and most of the auctions being done in the world are of this type. There are four major auction forms [13]:

- **The English Auction:** In this well-known open auction form, auctioneer begins with *reserve price*, lowest acceptable price, and proceeds to higher bids from bidders until no further increase in price occurs. Highest bidder gets the item by paying the amount of his highest bid. Since the auction is *open*, all the bids are known by the bidders during the auction. In the English auction, the auctioneer has also right to keep reserve price secret. Because of high competition level among the bidders and inexperienced bidders who bid up the price, *winner's curse*, paying more than item's real value, is common in this form of auction. Currently, most of the auctions being done in the world are of this form.
- **The Dutch Auction:** Although the English auction is the most common auction form, the Dutch auction is the first auction known in the history. As explained in the beginning of this chapter, Babylonians used this auction form for selling young women to rich men. In this open auction type bidding starts at an extreme price and is lowered until a buyer buys the item by calling "mine". The buyer pays the exact price at the time he calls "mine".
- **The First-Price, Sealed-Bid Auction:** The most important property of this auction is that bids are sealed and hence are hidden from other bidders. In this form of auction, generally each bidder can only submit one bid so preparation of a bid is extremely important. There are two phases of this auction form. A bidding phase in which bidders submit their bids and a resolution phase in which winner of the auction is determined. In this auction the winner pays the exact amount of money he offered.
- **The Vickrey Auction:** This form of auction is also known as *uniform second-price*. Like the first-price, sealed-bid auction, bids are sealed and hidden from other bidders. However in this auction form, winner of the auction pays only the amount of second highest bid. Although it seems that the first-price auction should be favored with respect to benefit of seller, practically this is not true. In Vickrey auctions, bidders do not fear for submitting high prices and this returns

to seller as profit.

Although we only considered single unit auctions, the discussed forms of auctions can be extended to multi-unit case in which more than one identical or equivalent objects are auctioned. For detailed discussion on multi-unit sequential auctions see [12].

2.2. Combinatorial Auction Model (Single Unit)

Sequential auctions are suitable when the value of each item is unrelated to the values of other items for every bidder. However, there may be complementarities and substitutabilities between items [14]. Assume that in an electronic equipment auction, several different brands of televisions and video recorders are to be auctioned. A bidder who does not have a television and a video recorder may request both equipments together because the gain of getting equipments together is more than sum of the gains of getting them separately. The reason is that although the gain of getting a television alone can be higher but gain of getting a video recorder without a television is much lower. Therefore getting both equipments is much more preferable. So we can say that there is a complementarity between televisions and video recorders for this bidder. Formally, complementarity between items i and j exists if $g_b(\{i,j\}) > g_b(\{i\}) + g_b(\{j\})$ where $g_b(S)$ is the gain of getting set of items S for a bidder b. If a bidder is interested only in one television without discriminating its brand, gain of getting second television would be useless for him and gain of getting both televisions would be lower than getting them separately. So we can say that there is a substitutability between televisions of different brands for this bidder. Formally, substitutability between items i and j exists if $g_b(\{i,j\}) < g_b(\{i\}) + g_b(\{j\})$ where $g_b(S)$ is the gain of getting set of items S for a bidder b.

If there are complementarities between different items, sequential auctions may provide inefficient allocation and a new type of auction model is necessary in order to increase economic efficiency. *Combinatorial auction* model solves the *complementarity* problem, having complementarities between items, by allowing bidders to submit bids

on combinations of different items [15, 16, 17]. In this model, all items are available to bidders and bidders are free to express their own valuations to any combination of items. Also in CA model it is possible to solve the *substitutability* problem, having substitutabilities between items, by introducing dummy items for each substitutable item. The role of dummy items are allowing bidder to express exclusive-or relationship between bids. For instance, if a bidder wants to get one television of brand A or B substitutably, one dummy item should be introduced and two separate bids, one bid for combination of dummy item and television of brand A and other bid for combination of dummy item and television of brand B, must be submitted by the bidder. Although this trick helps solving substitutability problem, number of bids increases combinatorially with the number of substitutable items.

Because of the rapid growth of electronic business, combinatorial auctions have become popular in the past years. Federal Communications Commission (FCC) spectrum auctions and auctions for airport time-slots, railroad segments and delivery routes can be cited as examples of real world combinatorial auctions [18]. Among these auctions, the most well-known auctions are the FCC spectrum auctions. Since 1994, FCC has conducted auctions of licenses for electromagnetic spectrum. These auctions are in the form of simultaneous multiple-round (SMR) auction where all licenses are available for bidding throughout the entire auction. SMR auctions are conducted in rounds, with the length of each round announced by FCC. After each round, results are processed and made public. Until the next round, bidders go over their bid strategies and adjust their bids if necessary. The FCC auctions also support combinatorial bidding so bidders may place bids on groups of licenses. The benefit of this approach is allowing bidders to express the value of any complementarities that may exist among licenses and to avoid the risk of winning only part of a desired set of licenses [19, 20].

2.2.1. Definition of CA Problem

Definition 1. Let $M = \{I_1, I_2, ..., I_m\}$ be the set of m items for sale. Let $B = \{b_1, b_2, \ldots, b_n\}$ be the set of n submitted bids where a bid is defined as $b_j =<$

$(\lambda_j^1, \lambda_j^2, \ldots, \lambda_j^m), p_j >$, with

$$\lambda_j^i = \begin{cases} 1 & \text{if item } k \text{ is requested in bid } j \\ 0 & \text{otherwise} \end{cases}$$

and $p_j > 0$ being the offered price. Then CA problem can be formulated as:

$$\text{maximize} \quad \sum_{j=1}^{n} p_j x_j \tag{2.1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} \lambda_j^i x_j \leq 1 \qquad (1 \leq i \leq m) \tag{2.2}$$

$$x_j \in \{0,1\} \; (1 \leq j \leq n) \tag{2.3}$$

In this formulation, x_j is a 0-1 variable that represents whether a bid is satisfied (1) or not (0). Objective line in Equation 2.1, maximizes the revenue gained from the auction. Finally, Equation 2.2 ensures that the bids that have at least one common item cannot be satisfied together.

2.2.2. CA Example

In this subsection, we will present a simple combinatorial auction example with four items. The items to be auctioned are one wired keyboard (*kyb*), one wired mouse (*mouse*), one wireless keyboard (*kyb_w*) and one wireless mouse (*mouse_w*). So $M = \{kyb, mouse, kyb_w, mouse_w\}$ with $m = 4$. Assume that following six bids are submitted:

- In bid 1, wired keyboard and mouse set is requested for 20 Turkish Liras (TL). So $b_1 = (< 1, 1, 0, 0 >, 20)$.
- In bid 2, wireless keyboard and mouse set is requested for 80 TL. So $b_2 = (< 0, 0, 1, 1 >, 80)$.
- In bid 3, two keyboards are requested for 40 TL. So $b_3 = (< 1, 0, 1, 0 >, 40)$.
- In bid 4, two mice are requested for 50 TL. So $b_4 = (< 0, 1, 0, 1 >, 50)$.

- In bid 5, only wireless keyboard is requested for 40 TL. So $b_5 = (< 0, 0, 1, 0 >, 40)$.
- In bid 6, only wireless mouse is requested for 50 TL. So $b_6 = (< 0, 0, 0, 1 >, 50)$.

This example can be formulated as:

$$
\begin{array}{lrcrcrcrcl}
\text{maximize} & 20x_1 & + & 80x_2 & + & 40x_3 & + & 50x_4 & + & 40x_5 + 50x_6 \\
\text{subject to} & x_1 & & & + & x_3 & & & & \leq 1 \\
& x_1 & & & & & + & x_4 & & \leq 1 \\
& & & x_2 & + & x_3 & & & + & x_5 \leq 1 \\
& & & x_2 & & & + & x_4 & + & x_6 \leq 1 \\
& & & & & & & & & x_j \in \{0,1\}\ (1 \leq j \leq 6)
\end{array}
$$

2.3. Multi-Unit Combinatorial Auction Model

Although single unit combinatorial auction model provides economically efficient allocations when the bidders are interested in bundles of items, they are inappropriate for situations where multiple instances of items are auctioned. Since the bidder is not interested in a special unit, he must bid separately to all combinations of items he wants. For instance, if the bidder wants to get 100 keyboards and 100 mice out of 200 keyboards and 300 mice in a single unit combinatorial auction, he must bid $\binom{200}{100} \cdot \binom{300}{100}$ times. Multi-unit combinatorial auction (MUCA) model solves this problem by representing identical items as multiple units of single item and allowing bidders to bid on units of items [21, 22, 23, 24, 25].

2.3.1. Definition of MUCA Problem

Definition 2. Let $M = \{I_1, I_2, ..., I_m\}$ be the set of m items and u_i be the number of units of item I_i for sale $(1 \leq i \leq m)$. Let $B = \{b_1, b_2, \ldots, b_n\}$ be the set of n submitted bids where a bid is defined as $b_j =< (\lambda_j^1, \lambda_j^2, \ldots, \lambda_j^m), p_j >$, with $\lambda_j^k \geq 0$ being the requested amount of item k, and $p_j > 0$ being the offered price. Then MUCA problem

can be formulated as:

$$\text{maximize} \quad \sum_{j=1}^{n} p_j x_j \tag{2.4}$$

$$\text{subject to} \quad \sum_{j=1}^{n} \lambda_j^i x_j \leq u_i \qquad (1 \leq i \leq m) \tag{2.5}$$

$$x_j \in \{0,1\} \; (1 \leq j \leq n) \tag{2.6}$$

In this formulation, x_j is a 0-1 variable that represents whether a bid is satisfied (1) or not (0). Objective line in Equation 2.4, maximizes the revenue gained from the auction. Equation 2.5 constrains the sum of the requested amount of resource type i by all bids with the number of units of resource type i, so it is ensured that for each resource type i, the resource limit u_i is preserved.

2.3.2. MUCA Example

We will extend the example in Section 2.2.2. Let the items to be auctioned be two hundred wired keyboards (*kyb*), four hundred wired mice (*mouse*), three hundred wireless keyboards (*kyb_w*) and one hundred wireless mice (*mouse_w*) . So $M = \{kyb, mouse, kyb_w, mouse_w\}$ with $m = 4$ and $u_1 = 200, u_2 = 400, u_3 = 300$, and $u_4 = 100$. Assume that following six bids are submitted:

- In bid 1, 200 wired keyboard and mouse sets are requested for 400 TL. So $b_1 = (< 200, 200, 0, 0 >, 400)$.
- In bid 2, 100 wireless keyboard and mouse set is requested for 800 TL. So $b_2 = (< 0, 0, 100, 100 >, 800)$.
- In bid 3, 100 wired keyboards and 300 wireless keyboards are requested for 950 TL. So $b_3 = (< 100, 0, 300, 0 >, 950)$.
- In bid 4, 200 wired mice and 100 wireless mice are requested for 900 TL. So $b_4 = (< 0, 200, 0, 100 >, 900)$.
- In bid 5, only 200 wireless keyboards are requested for 600 TL. So $b_5 = (< 0, 0, 200, 0 >, 600)$.

- In bid 6, only 100 wireless mouse is requested for 500 TL. So $b_6 = (< 0, 0, 0, 100 >, 500)$.

This example can be formulated as:

$$
\begin{array}{lrcl}
\text{maximize} & 400x_1 + 800x_2 + 950x_3 + 900x_4 + 600x_5 + 500x_6 & & \\
\text{subject to} & 200x_1 + 100x_3 & \leq & 200 \\
& 200x_1 + 200x_4 & \leq & 400 \\
& 100x_2 + 300x_3 + 200x_5 & \leq & 300 \\
& 100x_2 + 100x_4 + 100x_6 & \leq & 100 \\
& x_j \in \{0,1\}\ (1 \leq j \leq 6) & &
\end{array}
$$

2.4. Multi-Unit Nondiscriminatory Combinatorial Auction Model

In MUCA model, bidders are allowed to bid on instances of items. Therefore if multiple instances of items are to be auctioned, MUCA model provides efficient bid representation. Although substitutability problem between identical units of items is solved with this model, substitutability between different items is not considered. If the bidder does not differentiate two or more different items, MUCA model becomes insufficient for representing such preferences. Assume that in the example MUCA given in Section 2.2.2, a bidder wants to buy 100 keyboards without differentiating wired and wireless models. For this preference, one dummy item must be introduced and he must bid 101 times $((< 0, 0, 100, 0, 1 >, p), (< 1, 0, 99, 0, 1 >, p), \ldots, (< 100, 0, 0, 0, 1 >, p))$. Likewise if he wants to buy 100 instances of items without differentiating any four items, one dummy item must be introduced and he must bid 176,851 times $((< 0, 0, 0, 100, 1 >, p)(< 0, 0, 1, 99, 1 >, p), \ldots, (< 100, 0, 0, 0, 1 >, p))$. In order to overcome this inefficiency, we propose a new auction model called *MUNCA* model. In this model, bidders may encode their preferences of substitutability to bids easily by declaring list of nondiscriminatory items. Details of this model will be given in the next chapter.

3. CA BASED RESOURCE CO-ALLOCATION MODEL

In this chapter we will introduce a new economy based model for co-allocating resources in a grid environment called *CABRC* model. The goals of this model are:

- maximizing the revenue of resource holders and encouraging them to supply their idle resources,
- maximizing the utility and the value of resources by allowing buyers to express their preferences of complementarities and substitutabilities between resources,
- being easily adaptable to existing grid infrastructures.

This model is based on MUNCA model defined in Section 2.4. We will first give the definition and formulation of CABRC problem and prove the related theorems, then explain how to apply this model to grids and finally give an example of the model.

3.1. Definition of CABRC Problem

Definition 3. Let $R = \{r_1, r_2, \ldots, r_m\}$ be the set of m different resource types in a grid environment and let $U = \{u_1, u_2, \ldots, u_m\}$ be the set of units of resources where u_i is the number of available identical units of resource r_i $(1 \leq i \leq m, u_i \in \mathbb{Z}^+)$. Let $B = \{b_1, b_2, \ldots, b_n\}$ be the set of submitted bids. A bid consists of two parts, a list of subbids and an offered price to be paid if the bid is satisfied. There is logical AND relationship between subbids because a bid is satisfiable if *all* of the subbids are satisfiable. Similarly a subbid consists of two parts, a set of substitutable resource types and requested quantity of resources from this set. There is logical OR relationship between resource types in this set because a subbid is satisfiable if requested amount of resources can be supplied from the resource types in *any* subset of this set. So if the size of this set is more than one, it means that the bidder does not discriminate the listed resource types and treats them equivalently.

Let v be the maximum number of subbids among all bids. Then formally a bid

$b_j = \{(< s_{j1}, q_{j1} >, < s_{j2}, q_{j2} >, \ldots, < s_{jt_j}, q_{jt} >), p_j\}$ is defined as a combination of set of subbids and an offered price p_j where $s_{jk} \subseteq R$ is the set of requested substitutable resource types and q_{jk} is the requested quantity of resources for the set s_{jk} $(1 \leq j \leq n, 1 \leq k \leq t_j \leq v, p_j \in \mathbb{Z}^+)$. *CABRC* problem is defined as finding the subset b_s of bid set B and corresponding allocation of resources that maximizes the sum of the offered prices of bids in b_s while preserving the resource limits in U.

3.2. Formulation of CABRC Problem

In order to formulate this problem, two new variables x and y must be introduced. In this formulation x_j is a 0-1 variable that represents whether a bid is satisfied (1) or not (0) and $y_{jk}^{(i)}$ is a natural number that represents how many number of units of resource i are taken by kth subbid of bid j if it is satisfied $(1 \leq i \leq m, 1 \leq j \leq n, 1 \leq k \leq v)$.

$$\text{maximize} \quad \sum_{j=1}^{n} p_j x_j \tag{3.1}$$

$$\text{subject to} \quad \sum_{j=1}^{n}\sum_{k=1}^{v} y_{jk}^{(i)} \leq u_i \quad (1 \leq i \leq m) \tag{3.2}$$

$$(\sum_{i=1}^{m} y_{jk}^{(i)}) - q_{jk}x_j = 0 \quad (1 \leq j \leq n, 1 \leq k \leq v) \tag{3.3}$$

$$y_{jk}^{(i)} = 0 \quad (1 \leq i \leq m, 1 \leq j \leq n, 1 \leq k \leq v, r_i \notin s_{jk}) \tag{3.4}$$

$$x_j \in \{0,1\} \; (1 \leq j \leq n) \tag{3.5}$$

$$y_{jk}^{(i)} \in \mathbb{N} \quad (1 \leq i \leq m, 1 \leq j \leq n, 1 \leq k \leq v) \tag{3.6}$$

In this formulation, the objective line in Equation 3.1 ensures that the maximum revenue is gained from the auction. Equation 3.2 constrains the sum of the requested amount of resource type i by all subbids with the number of units of resource type i, so it is ensured that for each resource type i, the resource limit u_i for that resource type is preserved. Equation 3.3 constrains the sum of quantities for each item type inside a subbid to be equal with the requested quantity in that subbid if the bid j is satisfied and x_j is 1, otherwise $y_{jk}^{(i)}$ values are cleared to 0. In this equation, it is also ensured

that if a bid is satisfied, all the subbids inside that bid is also satisfied. In Equation 3.4, $y_{jk}^{(i)}$ value is set to 0 if resource i is not requested by kth subbid of jth bid.

3.3. Theorems Related to CABRC Problem

Proposition 1. *CABRC problem is NP-hard.*

Proof. Let Π be decision version of CABRC problem: Given set of resource types $R = \{r_1, r_2, ..., r_m\}$, set of units of resources $U = \{u_1, u_2, ..., u_m\}$, set of bids $B = \{b_1, b_2, \ldots, b_n\}$, and an integer P, is there a satisfiable subset of bids b_s such that the sum of the prices of bids in b_s is greater than or equal to P?

First we will show that Π is in NP. If we have a certificate that consists of z bids, and allocation values $y_{jk}^{(i)}$ for these bids, we can verify this certificate in polynomial time by checking the following equalities:

$$\sum_{j=1}^{z}\sum_{k=1}^{t_j} y_{jk}^{(i)} \leq u_i \quad (1 \leq i \leq m)$$

$$(\sum_{i=1}^{m} y_{jk}^{(i)}) = q_{jk} \quad (1 \leq j \leq z, 1 \leq k \leq t_j)$$

where t_j is the number of subbids in bid j. Therefore Π is in NP.

Next we will present a polynomial transformation from independent set problem (given G and positive integer P, does G contain an independent set of size P) to Π. Let $G = (V, E)$ be an instance of independent set problem where V is the set of n vertices, $V = \{v_1, v_2, \ldots, v_n\}$, and E is the set of m edges, $E = \{e_1, e_2, \ldots, e_m\}$. Assume that for each edge e_i in G we have a separate item type r_i in Π with only one available unit ($R = \{r_1, r_2, \ldots, r_m\}, U = \{1, 1, ..., 1\}$). Also for each vertex v_j in G we define a bid b_j with price 1 in Π and for each edge e_i connected to v_j we add a subbid to bid b_j with requested subset of r_i and quantity of 1 ($b_j = \{(< s_{j1}, q_{j1} >, < s_{j2}, q_{j2} > , \ldots, < s_{jt_j}, q_{jt_j} >), p_j\}$ where $q_{jk} = 1$, $s_{jk} = \{r_i\}$ if e_i is connected to v_j, $p_j = 1$,

$t_j = degree(v_j)$, $1 \leq j \leq m$, $1 \leq k \leq t_j$). Then if G has an independent set with vertices v_o and size P, then by the definition of independent set, there can be no edge connecting vertices v_o of this independent set. Since edges represents a common item between bids in Π, we can conclude that subset of bids b_o corresponding to vertices v_o, does not share a common item and all bids in b_o can be satisfied. As the price of all bids are set to 1, P will be the total price of bids in b_o. Similarly if Π has a solution b_o with total price P, since there is only one unit of each item type, bids in b_o cannot share a common item. As a result, vertices v_o that correspond to bids b_o cannot have edges in common. As the price of all bids are set to 1, G must have an independent set of size P. Therefore we can conclude that a solution to independent set problem will be a solution to Π and vice versa.

Since Π is in NP and independent set problem was proved to be NP-complete [26], Π is NP-complete and therefore CABRC problem is NP-hard. □

3.4. Applying CABRC Model to Grids

In this section, we will first briefly discuss two different approaches of applying CABRC model to grids and then comment on identifying resources in a typical grid environment.

3.4.1. Economy Based Approach

In this approach, resource suppliers declare their available resources for hiring and buyers submit bids on these resources. Rental period can be either fixed or variable as decided by the supplier. In the fixed period scheme, the resources are allocated to winner for a specific amount of time. After the end of this time, the resources get released and a new auction can be conducted. In the variable length period scheme, the winner pays an extra amount of money beside the bid price for the time he is using the resources. New auctions can be conducted after the winner releases the resources.

In the auction, all resources are available to all bidders and auction can be held either in one session as in the first-price sealed bid auction model or in successive sessions as in the SMR auction model.

3.4.2. Priority Based Approach

For non-profit organizations like academic institutions, priority based approach can be used for co-allocating resources. In this approach, priority values are used instead of price values for resource requests. Then CABRC model is used for maximizing sum of priorities instead of prices. Although no mathematical changes in the model are necessary for this approach, defining priority policy, what priority value to be assigned for a request, can be difficult. However, the simplest policy, assigning 1 to priorities of all requests, can be sufficient for most cases. In order to prevent starvation of requests, *aging* technique can be used. In this technique, priorities of waiting requests are gradually increased in order to get higher chance of winning in the next auction [27].

3.4.3. Identifying Resources in a Grid Environment

Any resource in a grid environment that can be reserved to a user for a specific amount of time can be used as resource type in CABRC model. To be more specific, if advanced reservations of resources are possible in a grid environment [28], following resources can be used in CABRC model:

- **Processors:** Since most of the current operating systems provide preemptive multitasking, percentage time slots of different processors can be used as a resource type in the model. However, along with the processor time slot, a fixed memory and fixed hard disk space must also be allocated for every processor request.
- **Memory and Secondary Storage:** Each process requires memory and storage space in order to be executed by the processor. However, some processes require more memory and storage space than others. In order to provide better allocation

of resources, remaining memory and storage space can be defined as separate resource types. Then users of applications that require more memory and/or storage space can bid for more memory and storage space beside processor time.

- **Network Bandwidth:** A predefined amount of bandwidth is required for applications running in parallel. Like in memory and secondary storage case, some parallel applications may require more bandwidth then other parallel applications. So we can also model excess bandwidth inside and between clusters as separate resource types.

Although these described resources can be used as resource types in CABRC model, in practice bidding for these resources may get overcomplicated. So instead of dividing resources inside a computer, we can threat each computer configuration as one resource type. If only one application runs on a single computer, then it may use all available resources of the computer and share the network bandwidth with other applications running on other computers. If more than one application is permitted to run on a single machine, then applications running on a computer at the same time share available memory, secondary storage space and available network bandwidth equally. This simplifies both the bidding process and the co-allocation process at system level. Beside computers and network bandwidth, external storages, databases, licenses and other reservable resources such as input/ouput units can also be used as resource types in this model.

3.5. CABRC Problem Example

In this section, we will present a simple CABRC model example. Assume that in a small computational grid following reservable resources are available:

- 10 Intel x86 workstations (*intel*)
- 10 AMD x86 workstations (*amd*)
- 20 Sun workstations (*sun*)
- A license server with 5 MATLAB licenses (*matlab*) and 5 CPLEX licenses (*cplex*)
- 10 GB (in 1GB chunks) storage space in an external storage server (*storage*)

Among these resources, first bidder requests 10 Intel x86 workstations, 5 MATLAB licenses and 4 GB storage space for 1,000 TL. Second bidder requests 10 x86 workstations and 5 CPLEX licenses for 600 TL and the last bidder requests 30 workstations and 5 GB storage space for 1,500 TL. So for this environment, $R = \{intel, amd, sun, matlab, cplex, storage\}$, $U = \{10, 10, 20, 5, 5, 10\}$ and submitted bids are:

- $b_1 = \{(< \{intel\}, 10 >, < \{matlab\}, 5 >, < \{storage\}, 4 >), 1000\}$
- $b_2 = \{(< \{intel, amd\}, 10 >, < \{cplex\}, 5 >), 600\}$
- $b_3 = \{(< \{intel, amd, sun\}, 30 >, < \{storage\}, 5 >), 1500\}$

This example can be formulated as (for simplicity we omitted variables and constraints if $y_{jk}^{(i)} = 0$):

$$
\begin{aligned}
\text{maximize} \quad & 1000x_1 + 600x_2 + 1500x_3 \\
\text{subject to} \quad & y_{11}^{(1)} + y_{21}^{(1)} + y_{31}^{(1)} \leq 10 \\
& y_{21}^{(2)} + y_{31}^{(2)} \leq 10 \\
& y_{31}^{(3)} \leq 20 \\
& y_{12}^{(4)} \leq 5 \\
& y_{22}^{(5)} \leq 5 \\
& y_{13}^{(6)} + y_{32}^{(6)} \leq 10 \\
& y_{11}^{(1)} - 10x_1 = 0 \\
& y_{12}^{(4)} - 5x_1 = 0 \\
& y_{13}^{(6)} - 4x_1 = 0 \\
& y_{21}^{(1)} + y_{21}^{(2)} - 10x_2 = 0 \\
& y_{22}^{(5)} - 5x_2 = 0 \\
& y_{31}^{(1)} + y_{31}^{(2)} + y_{31}^{(3)} - 30x_3 = 0 \\
& y_{32}^{(6)} - 5x_3 = 0 \\
& x_1, x_2, x_3 \in \{0, 1\} \\
& y_{jk}^{(i)} \in \mathbb{N} \text{ (for all } i, j, k)
\end{aligned}
$$

3.5.1. Exclusive-OR Bids

Although CABRC model does not directly support exclusive-OR bids, there is an indirect method for handling this problem. Assume that bids b_1 and b_2 are exclusive-ORed ($b_1 \oplus b_2$). We introduce a dummy resource $d1$ with only one instance. Then we add one subbid of the same dummy resource with one unit of quantity for each exclusive-ORed bids. So

- $b_1 = \{(< \{intel\}, 10 >, < \{matlab\}, 5 >, < \{storage\}, 4 >, < \{d1\}, 1 >), 1000\}$
- $b_2 = \{(< \{intel, amd\}, 10 >, < \{cplex\}, 5 >, < \{d1\}, 1 >), 600\}$

Since there is only one unit of $d1$, b_1 and b_2 cannot be satisfied together.

4. CABRC PROBLEM SOLVER

CABRC problem solver package consists of the following solvers and estimators:

- Optimum solver (OPT)
- Linear relaxation based upper bound estimator (LRE)
- Linear relaxation based greedy heuristic solver (LRS)
- Greedy heuristic solver based on price per unit criteria (PS)
- Enhanced heuristic solver based on price per unit criteria (EPS)

Linear relaxation based methods are widely used for bounding NP-hard problems because of their simplicity and effectiveness. Although our main contributions in this study are OPT, PS and EPS, we included LRE and LRS in order to present how tight the linear relaxation based bounds for CABRC problem are. Solver package is coded in ANSI C++ and compiled using Microsoft Visual C++ .NET 2003 compiler. CPLEX solver version 8.1.1 [11] is used for solving integer and linear programming problems and network flow problems.

4.1. Optimum Solver

Optimum solver solves CABRC problem by calling CPLEX mixed integer programming solver (CPXmipopt function) for the IP problem described in Section 3.2.

4.2. Linear Relaxation Based Upper Bound Estimator

This trivial estimator finds an upper bound for the optimum solution of CABRC problem by solving the linear relaxation of the IP formulation described in Section 3.2.

Linear relaxation of the formulation is,

$$\text{maximize} \quad \sum_{j=1}^{n} p_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n}\sum_{k=1}^{v} y_{jk}^{i} \leq u_i \quad (1 \leq i \leq m)$$

$$(\sum_{i=1}^{m} y_{jk}^{i}) - q_{jk}x_j = 0 \quad (1 \leq j \leq n, 1 \leq k \leq v)$$

$$y_{jk}^{i} = 0 \quad (1 \leq i \leq m, 1 \leq j \leq n, 1 \leq k \leq v, r_i \notin s_{jk})$$

$$x_j \geq 0 \quad (1 \leq j \leq n)$$

$$x_j \leq 1 \quad (1 \leq j \leq n)$$

$$y_{jk}^{i} \geq 0 \quad (1 \leq i \leq m, 1 \leq j \leq n, 1 \leq k \leq v)$$

LRE uses CPLEX linear programming solver (CPXlpopt function) for solving this linear programming problem.

4.3. Linear Relaxation Based Greedy Heuristic Solver

LRS finds an approximate feasible solution to CABRC problem by solving the linear relaxation of the IP formulation and running a greedy heuristic based on the results of linear relaxation solution. LRS algorithm is based on the assumption that after solving the linear relaxation of the problem, bids with highest x_j values are more likely to be in the optimum solution. This algorithm consists of two phases. In the ranking phase, linear relaxation of the problem is solved and then bids are sorted according to x_j values in descending order. In the allocation phase, we start by adding the first bid in the sorted list to empty winning bids set and check whether the bid in the set is feasible or not. If it is feasible, we continue adding the second bid in the list to the winning bids set. If it is not feasible, we remove the bid from the set and then add the second bid. Then we check the feasibility of the set again. The procedure continues in this manner until all the bids in the sorted list are traversed. After the end of procedure, winning bid set is returned. The pseudocode for LRS algorithm is presented in Figure 4.1. *checkFeasibility* function will be explained in the following

subsection.

Algorithm LRS

Input: CABRC problem instance

Output: $winningBids$

/* Ranking Phase */

solve LP relaxation of CABRC problem and get x_j values for each bid

sort bids according to x_j in descending order and store in $orderedBids$

/* Allocation Phase */

$winningBids = \emptyset$

for $j = 0$ **to** $n - 1$

    add $orderedBids_j$ to $winningBids$

    if checkFeasibility($winningBids$) = false **then**

        remove $orderedBids_j$ from $winningBids$

end for

return $winningBids$

Figure 4.1. The pseudocode for LRS algorithm

4.3.1. Checking Feasibility of Given Bid Set

For all presented greedy heuristics, checking feasibility of a given bid set is an essential step. Unlike MUCA model, checking feasibility in CABRC model is not trivial because of ORed resource types in the subbids. In order to check feasibility of given

bid set $B = \{b_1, b_2, \ldots, b_n\}$, following set of equations must be solved,

$$\sum_{j=1}^{n}\sum_{k=1}^{v} y_{jk}^{i} \leq u_i \quad (1 \leq i \leq m)$$

$$(\sum_{i=1}^{m} y_{jk}^{i}) - q_{jk} = 0 \quad (1 \leq j \leq n, 1 \leq k \leq v)$$

$$y_{jk}^{i} = 0 \quad (1 \leq i \leq m, 1 \leq j \leq n, 1 \leq k \leq v, r_i \notin s_{jk})$$

$$y_{jk}^{i} \in \mathbb{N} \quad (1 \leq i \leq m, 1 \leq j \leq n, 1 \leq k \leq v)$$

We modeled this problem as a feasible network flow problem (for details on network flow problems, see [29]). The model is presented in Figure 4.2.

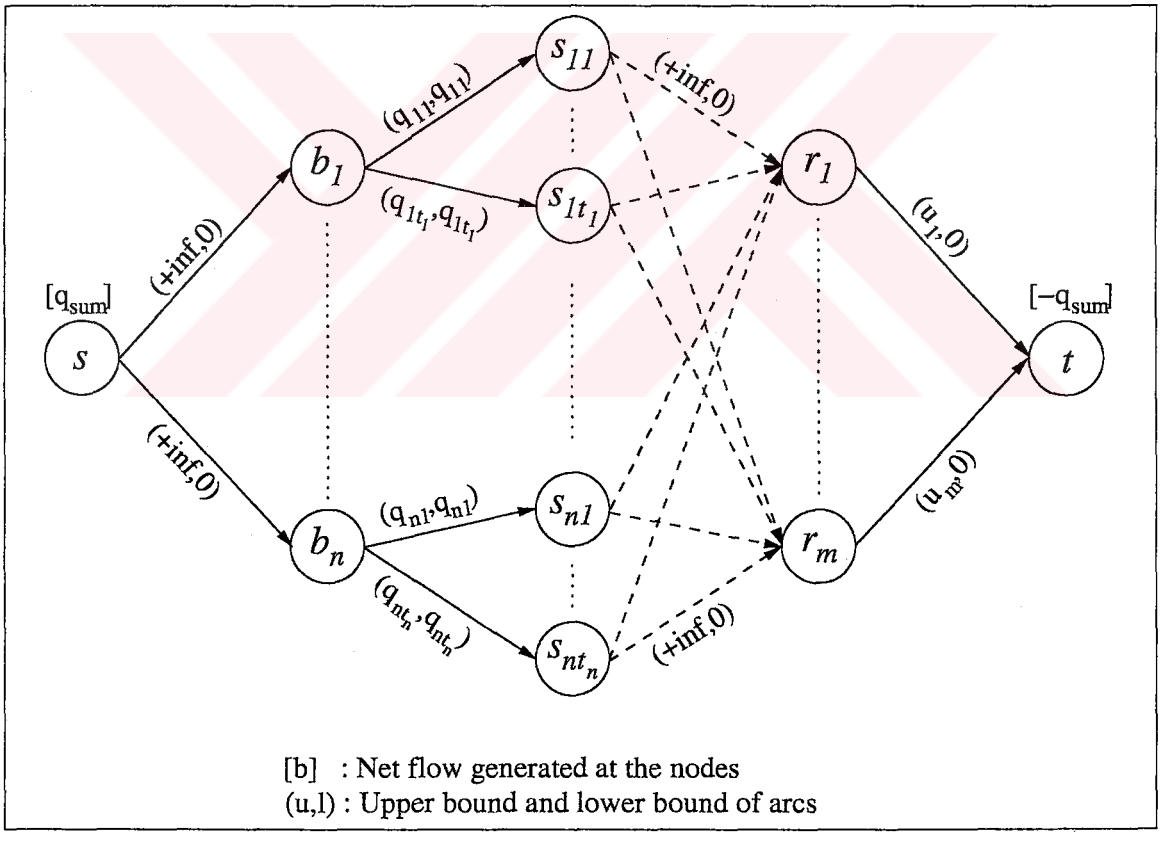


Figure 4.2. Network flow diagram for checking feasibility of bids

Let $N(V, A, l, u, b)$ denote the network. We begin constructing the network by adding one node, b_j, for each bid and drawing an arc from source node s, to b_j with infinite upper bound and zero lower bound $((u, l) = (+inf, 0), 1 \leq j \leq n)$. Then for each node b_j, we add t_j nodes where t_j is the number of subbids inside bid j, and

draw an arc from b_j to each newly added node s_{jk} with fixed flow requirement of q_{jk} $((u,l) = (q_{jk}, q_{jk}), 1 \leq j \leq n, 1 \leq k \leq t_j)$. These arcs with fixed flow ensure that each subbid gets the requested amount of resources. After that we add one node, r_i, for each resource type in R and draw an arc from s_{jk} if resource i is requested in subbid k of bid j $(1 \leq i \leq m)$. There are no flow constraints (except being positive) on these arcs $((u,l) = (+inf, 0))$. These arcs ensure that only requested resource types for each subbid are allocated by that subbid. Finally, we add one arc from each resource type node r_i to sink node t with upper bound of u_i and lower bound of zero $((u,l) = (u_i, 0)$. These last set of arcs limits the number of units resource types. Supply/demand values for internal nodes are set to zero, source node supplies $[q_{sum} = \sum_{j=1}^{n} \sum_{k=1}^{t_j} q_{jk}]$ amount of flow and sink node demands the same amount as the source node $[-q_{sum}]$.

If this network model is feasible, then we can conclude that given set of bids is also feasible. After solving the model, amount of flow between nodes s_{jk} and r_i gives $y_{jk}^{(i)}$ values $(x_{s_{jk},r_i} = y_{jk}^{(i)})$.

4.4. Greedy Heuristic Solver Based on Price Per Unit Criteria

Like LRS, PS is also based on greedy allocation of bids however for sorting bids, price per unit criteria is used instead of linear relaxation. In the ranking phase of the algorithm, we first calculate heuristic value of each bid j using the following formula:

$$h_j = \frac{p_j}{\sum_{k=1}^{t_j} q_{jk}} \tag{4.1}$$

Then we sort bids according to these heuristic values in descending order. The rest of the algorithm, the allocation phase, is the same as LRS algorithm. The pseudocode for PS algorithm is presented in Figure 4.3.

Algorithm PS

Input: CABRC problem instance

Output: $winningBids$

/* Ranking Phase */

for each bid j

  $qSum_j = 0$

  for each subbid k of bid j

    $qSum_j = qSum_j + q_{jk}$

  end for

$h_j = \frac{p_j}{qSum_j}$

end for

sort bids according to h_j in descending order and store in $orderedBids$

/* Allocation Phase */

$winningBids = \emptyset$

for $j = 0$ **to** $n - 1$

  add $orderedBids_j$ to $winningBids$

  if checkFeasibility($winningBids$) = false **then**

    remove $orderedBids_j$ from $winningBids$

end for

return $winningBids$

Figure 4.3. The pseudocode for PS algorithm

4.5. Enhanced Heuristic Solver Based on Price Per Unit Criteria

In EPS, we change the ranking scheme by introducing two new factors, or_factor (α) and and_factor (β). The new ranking formula is:

$$h_j = \frac{p_j}{\sum_{k=1}^{t_j} (q_{jk} \cdot \alpha^{|s_{jk}|-1}) \cdot \beta^{t_j-1}} \tag{4.2}$$

The allocation phase is the same with PS algorithm however different from PS, this algorithm increases the chance of finding better solution by changing or_factor and and_factor in the range of $[0.9, 1.1]$ with 0.05 increments and returns the best solution found. The original idea behind this ranking scheme was to favor the number of ORed resource types inside the subbids and to disfavor the number of ANDed subbids inside the bid. In order to realize this idea, the range of or_factor was originally set to $[0.9, 1]$ and the range of and_factor was set to $[1, 1.1]$. However, we observed that extended ranges gave slightly better solutions because of the exceptional instances of the problem.

When or_factor and and_factor are set to 1, this algorithm gives same result as PS, therefore EPS is guaranteed to give better solutions than PS. However, this also makes EPS to run up to approximately twenty five times slower than PS. The pseudocode for EPS algorithm is presented in Figure 4.4.

Algorithm EPS

Input: CABRC problem instance

Output: $maxBids$

$maxPrice = 0$;

for $\beta = 0.9$ **to** 1.1 **step** 0.05 /* **and_factor** */

  for $\alpha = 0.9$ **to** 1.1 **step** 0.05 /* **or_factor** */

    /* **Ranking Phase** */

    for each bid j

      $qSum_j = 0$

      for each subbid k of bid j

        $qSum_j = qSum_j + q_{jk} \cdot \alpha^{|s_{jk}|-1}$

      end for

      $h_j = \frac{p_j}{qSum_j \cdot \beta^{t_j - 1}}$

    end for

    sort bids according to h_j in descending order and store in $orderedBids$

    /* **Allocation Phase** */

    $winningBids = \emptyset$

    for $j = 0$ **to** $n - 1$

      add $orderedBids_j$ to $winningBids$

      if checkFeasibility($winningBids$) = false **then**

        remove $orderedBids_j$ from $winningBids$

    end for

    if $total_price(winningBids) > maxPrice$ **then**

      $maxPrice = total_price(winningBids)$

      $maxBids = winningBids$

    end if

  end for

end for

return $maxBids$

Figure 4.4. The pseudocode for EPS algorithm

5. CABRC PROBLEM TEST CASE GENERATOR

Since there is no real world data for CABRC problem, we code an artificial CABRC problem test case generator in order to generate a test suite for observing the performance of our solvers. The generator is capable of producing test cases for full-factorial testing in which all possible combinations of all factors can be tested.

Generator module is coded in ANSI C++ and compiled using Microsoft Visual C++ .NET 2003 compiler. It uses GNU Scientific Library (GSL) [30] for generating pseudo-random numbers.

CABRC model generator requires settings for eleven different configuration parameters related to CABRC model. The list of parameters are given below.

- *number_of_instances* defines how many set of instances will be generated.
- m defines how many types of resource will be generated.
- u defines number of units for each resource type.
- n defines how many bids will be generated for each instance.
- t defines how many subbids will be generated for each bid in the model.
- s defines size of requested subset of resources for each subbid inside all bids.
- *s_jk_method* defines method for generating requested resources for subbids.
- q defines requested amount of resources for each subbid inside all bids.
- *or_factor* defines price factor for ORed resource requests inside a subbid.
- *and_factor* defines price factor for ANDed subbids inside a bid.
- *price_variance* defines variance for calculating price.

The algorithm of generator is relatively simple: For each parameter, the generator draws pseudo random numbers according to a given distribution type and the parameters, and construct CABRC problem instances based on these numbers. Although the algorithm is simple in general, the method for determining the resource types inside a subbid and determining the price of a bid needs more explanation. In the following

sections, we will describe these methods in detail.

5.1. Determining Requested Resource Types in a Subbid

There are two methods for generating requested resource types for subbids. First method (0) is called *uniform random* method. In this method, resource types of subbids are chosen randomly from all available resource types. Second method (1) is called *neighborhood* method. In this method, first resource type of subbid is chosen randomly among all resource types and the remaining resource types are chosen from the neighbors (in terms of resource type index) of chosen resource type. For instance let $m = 10$, and for a subbid let $s = 5$ and $index\ of\ chosen\ resource = 8$, then list of requested resources are {6,7,8,9,0}. In grids, it is more likely that clusters that are connected with fast networks or geographically closer to be chosen inside a subbid. This method is proposed in order to model this property simplistically.

5.2. Determining the Price of a Bid

Assignment of proper prices for bids has important role for generating realistic test cases. In the generator, after number of resources are determined, a uniform random number between 0 and 1 is assigned as the price of one unit of each resource. In order to determine the price of a bid, we first find the price of each subbid in the bid. Raw price of a subbid is determined by multiplying requested quantity of resources with the *weighted average price* of ORed resources inside the subbid. In order to favor substitutability between resources, we multiply the raw subbid price with $\alpha^{|s_{jk}|-1}$ where α is the or_factor and $|s_{jk}|$ is the number of resource types inside the subbid. This produces the price of the subbid. Then we sum up the prices of subbids inside a bid and in order to disfavor complementarities between subbids, we multiply this value with β^{t_j-1} and produce the raw bid price where β is the and_factor and t_j is the number of subbids inside the bid. Finally, we draw a normally distributed random number with mean raw bid price, and standard deviation $price_variance$. We assign this number as the price of the bid. The pseudocode for determining the price of a bid is presented in Figure 5.1.

Price Determination Algorithm

Input: or_factor, and_factor, price_variance

Output: price for given bid

for each resource type i

$unitPrice_i = uniform(0, 1)$

end for

for each bid j

$bidPrice_j = 0$

for each subbid k of bid j

$subbidPrice_k = 0$

for each resource type l in subbid k

$subbidPrice_k = subbidPrice_k + q_{jk} \cdot unitPrice_l \cdot (u_l / \sum_{z \in s_{jk}} u_z)$

end for

$subbidPrice_k = subbidPrice_k \cdot \alpha^{|s_{jk}|-1}$

$bidPrice_j = bidPrice_j + subbidPrice_k$

end for

$bidPrice_j = bidPrice_j \cdot \beta^{t_j - 1}$

$bidPrice_j = normal(\mu = bidPrice_j, \sigma = (bidPrice_j \cdot price_variance/100))$

end for

Figure 5.1. The pseudocode for determining the price of a bid

5.3. Distributions Used in the Generator

The generator currently supports uniform, normal, exponential distributions and fixed value parameters for generating CABRC problem instances. Details of these distributions are given in Appendix A.

6. EXPERIMENTAL RESULTS

In order to measure the performance of CABRC problem solvers, we prepare a comprehensive test suite generated by CABRC problem test case generator and conduct tests in two phases. In the first phase, we examine the effect of each parameter separately while holding the other parameters fixed and in the second phase, we check the general performance of solvers by combining the results in the first phase and the results of several general tests.

We run the tests on the following platform: AMD Athlon 64 3200+ based workstation with 1GB RAM. The operating system used is Microsoft Windows XP Service Pack 1.

Goodness values of solutions for each configuration are calculated using the following formula:

$$\frac{Solution\ Of\ Solver\ s\ for\ Distribution\ d}{Optimum\ Solution\ for\ Distribution\ d} \cdot 100 \tag{6.1}$$

where s is either LRE, LRS, PS or EPS and d is either uniform, normal, or exponential. Running time values are recorded using wall clock time in seconds. Maximum running time for optimum solver is set to 1000 seconds for all configurations except "Number of Bids" configurations which are set to 2000 seconds. The reason for increasing the maximum running time is that in "Number Of Bids" configuration, tests are conducted for larger n values which are exactly the twice of the n values in other configurations. The results of all solvers are omitted if the optimum solution cannot be found.

Also note that since goodness values of the upper bound estimator and greedy heuristic solvers are calculated according to the same formula, they should be interpreted differently. For greedy heuristic solvers, higher goodness values are better, whereas for upper bound estimator, lower goodness values are better.

6.1. Phase 1: Effect of Parameters

In this part, our aim is to find effects of each parameter on the results separately while holding other parameters fixed. We will also examine the effects of different distributions on the results. For each parameter, we generate three sets of CABRC problem instances based on uniform, normal and exponential distributions. Base configuration files for each set can be found in Appendix C. Graphs presented in this part are based on mean values of results for configurations based on these three distributions.

In this phase of the tests, if the mean of a parameter is set to $mean$, then parameter(s) of

- uniform distribution are $(a = 1, b = (2 \cdot mean) - 1)$,
- normal distribution are $(\mu = mean, \sigma = mean/4)$,
- exponential distribution is $(\mu = mean)$.

Since EPS algorithm is an enhanced version of PS algorithm and is guaranteed to produce better results than PS algorithm, goodness results of PS and EPS would not be compared. Also since it is guaranteed that PS runs faster than EPS, running times performances would not be compared too. The results of PS are given in order to consider the trade-off between speed of algorithms and goodness of results.

In this phase of the tests, we mention the results of greedy heuristics as lower bounds of the optimum solution in order to be confident with the upper bound estimator and therefore the word *tightness* refers to how close the approximate solutions are to the optimum solution.

6.1.1. Number of Resource Types (m)

We conducted tests for $m = \{20, 40, 60, 80, 100\}$. The results of each distribution are given in Table B.1 and Table B.2. The mean results can be seen in Figure 6.1. Note that logarithmic scale is used for the y axis of the running time plot.

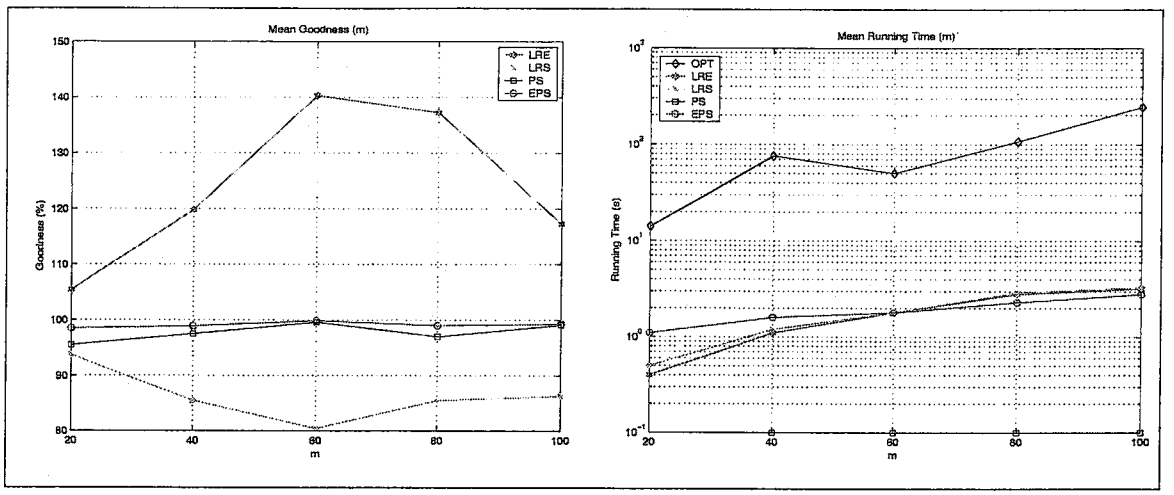


Figure 6.1. Goodness of solutions and running times: (m)

For uniform and normal distributions, LRE produces tight upper bounds with a maximum mean of 104 per cent and corresponding standard deviation of 2 per cent. However, for exponential distribution, the upper bound is far away from tightness with a maximum mean of approximately 215 per cent. For LRS, the results are nearly the same as the results of LRE. For uniform and normal distributions, the results of LRS ,although not so tight, are tighter than results of LRS for the exponential distribution. Results of EPS are the tightest among all greedy heuristic solutions with a minimum mean of 97.5 per cent and corresponding standard deviation of 3.3 per cent.

In terms of running times, the optimum solver is much slower than heuristic solvers as expected. PS is the fastest among all and EPS is faster than linear relaxation based heuristic. In general running times of all solvers are increased with m. According to the mean running times of the optimum solver, we can also conclude that in general, the problem gets harder if we increase the number of resource types. However it should also be noted that the *size of requested subset of resources (s)* is also based on the value of m.

6.1.2. Set of Units of Resources (u)

We conducted tests for $mean(u) = \{2, 4, 8, 16, 32\}$. The results of each distribution are given in Table B.3 and Table B.4. The mean results can be seen in Figure

6.2.

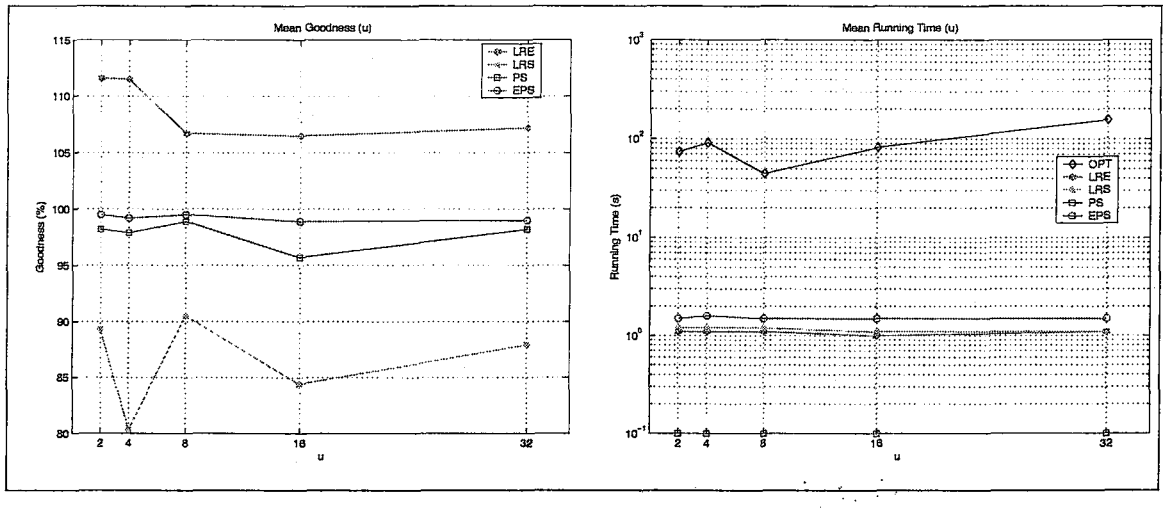


Figure 6.2. Goodness of solutions and running times: (u)

Like results of configuration m, for uniform and normal distributions, LRE produces tighter upper bounds than for exponential distribution with a maximum mean of 106 per cent and corresponding standard deviation of 2.2 per cent. However for exponential distribution, upper bound is tighter than results of configuration m, with a maximum mean of approximately 126.5 per cent. For LRS, the results for normal distribution are better than results for uniform distribution. The results for exponential distribution are again the worst in terms of tightness. So linear relaxation methods produce relatively better bounds for uniform and normal distributions but fail for exponential distribution. Results of EPS are the tightest among all greedy heuristic solutions with a minimum mean of 97.7 per cent and corresponding standard deviation of 2.3 per cent.

The speed of the optimum solver for exponential distribution is interestingly better than those of uniform and normal distributions, although in general it is much more slower than heuristic solvers. PS is the fastest among all heuristics and the speed of other heuristics are nearly the same. On average, running times of the optimum solver tend to increase with u, however, speed of heuristics are not affected by changes in this parameter.

6.1.3. Number of Bids (n)

We conducted tests for $n = \{100, 200, 300, 400, 500\}$. The results of each distribution are given in Table B.5 and Table B.6. The mean results can be seen in Figure 6.3.

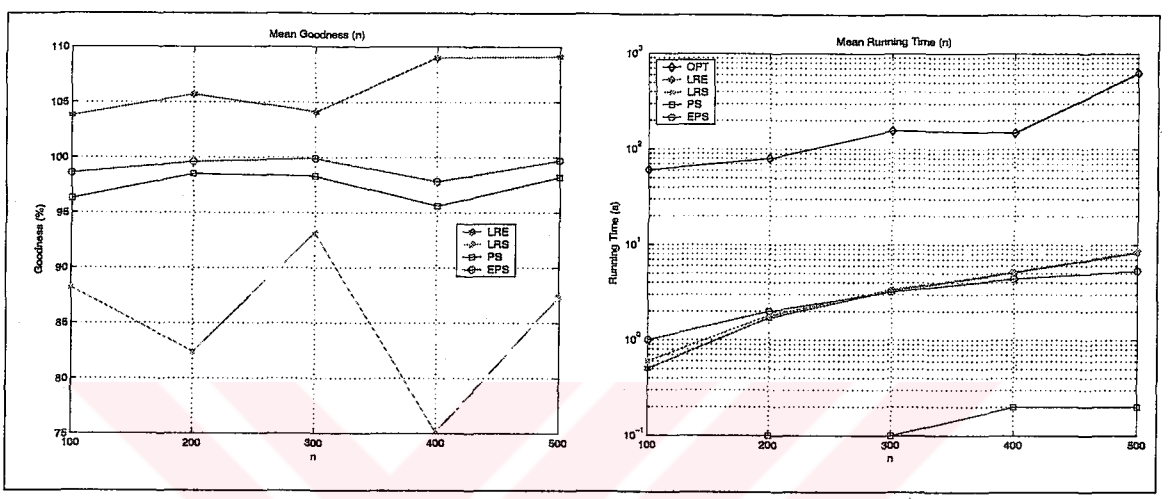


Figure 6.3. Goodness of solutions and running times: (n)

The tightness of upper bounds produced by LRE are less than 110 per cent except for the exponential distribution with $n = 400, 500$ which are less than 120 per cent. So, it can be concluded that results of LRE do not change dramatically with the type of distribution unlike in configuration m. For LRS, the results for normal distribution which oscillate between 90 per cent and 98 per cent, are better than results for uniform distribution which oscillate between 77.3 per cent and 92.3 per cent. The results for exponential distribution are very discouraging with a minimum of 50.6 per cent. Results of EPS are the tightest among all greedy heuristic solutions with a minimum mean of 96.6 per cent and corresponding standard deviation of 4.8 per cent.

According to mean running times of the optimum solver, we can conclude that in general, the problem gets harder if we increase the *number of bids*. Also the running times of heuristics increase with n as expected.

6.1.4. Number of Subbids for Bids (t)

We conducted tests for $mean(t) = \{3, 7, 11, 15, 19\}$. The results of each distribution are given in Table B.7 and Table B.8. The mean results can be seen in Figure 6.4.

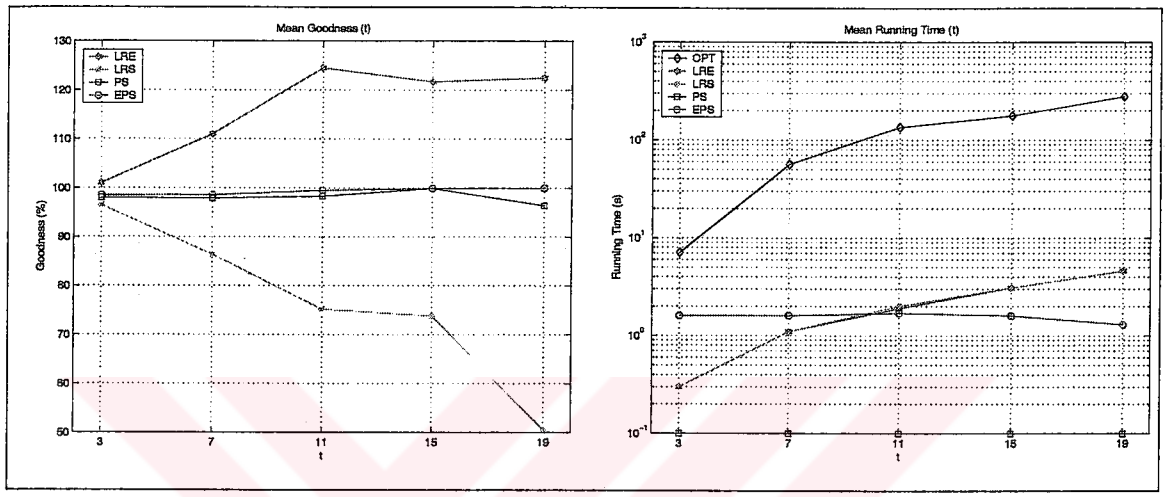


Figure 6.4. Goodness of solutions and running times: (t)

For $mean(t) = 3$, LRE produces tighter bounds of less than 102 per cent and with an increase in t, mean upper bound gets relaxed to between 120-130 per cent. LRE gives best bounds of maximum 112.4 per cent for normal distribution, and gives worst results of maximum 159.7 per cent for exponential distribution. Like LRE, LRS tends to give worse results as t increases. On average, goodness of LRS starts with 96.5 per cent for $mean(t) = 3$ and decreases to 50.6 per cent for $mean(t) = 19$. The results of PS is close to the results of EPS. EPS again gives the best results in all tests generated for this configuration.

Mean running times of the optimum solver strictly increase as t increases. So, *number of bids (n)*, and *number of subbids for bids (t)* parameters are important factors for determining the difficulty of CABRC problem instance. However, the speed of heuristic solvers are not affected from the change in t.

6.1.5. Size of Requested Subset of Resources (s)

We conducted tests for $mean(t) = \{10, 20, 30, 40, 50\ per\ cent\}$ where percentage defines the relativity to *number of resource types (m)*. The results of each distribution are given in Table B.9 and Table B.10. The mean results can be seen in Figure 6.5.

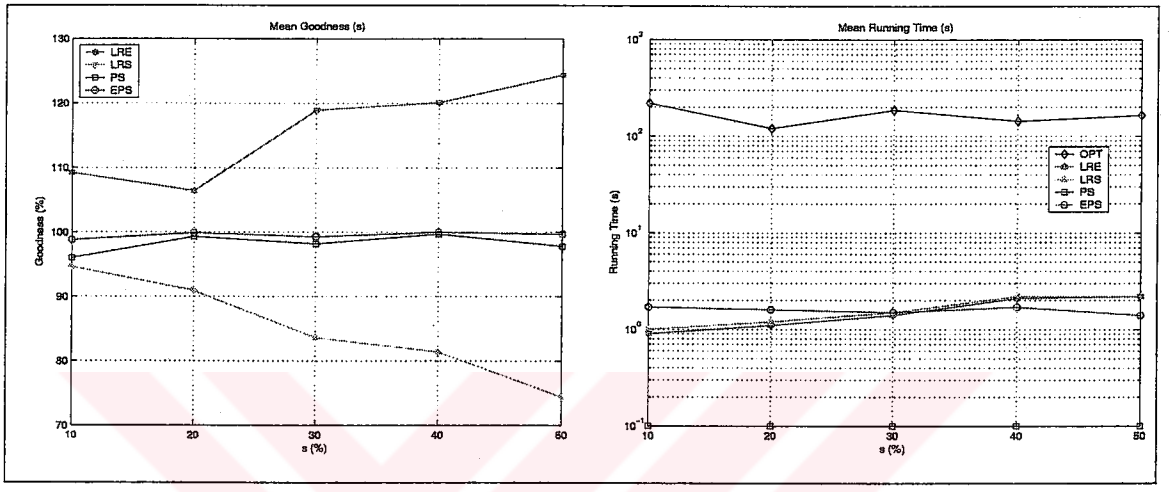


Figure 6.5. Goodness of solutions and running times: (s)

The goodness pattern of LRE and LRS resembles the pattern in configuration t. Lower t values produce better results. Mean goodness of LRE starts from approximately 110 per cent when $mean(t) = 10\ per\ cent$ and increases to 125 per cent when $mean(t) = 50\ per\ cent$ with little oscillations. Also mean goodness of LRS starts from 94.6 per cent and decreases to 74.4 per cent. Performances of PS and EPS are closer and EPS gives the best results in all tests generated for this configuration.

Mean running times of the optimum solver are about 160 seconds. Interestingly, the optimum solutions for normal distribution take four to nine times more time than those for the exponential distribution. In general, there is no pattern of variation between running times of all solvers according to s so it is more likely that the parameter s does not affect the difficulty of the model.

6.1.6. Method for Determining Elements of s_{jk} ($s_{jk}method$)

We conducted tests for $s_{jk}method = \{0, 1\}$ where zero means uniform random method and one means neighborhood method. The results of each distribution are given in Table B.11 and Table B.12. The mean results can be seen in Figure 6.6.

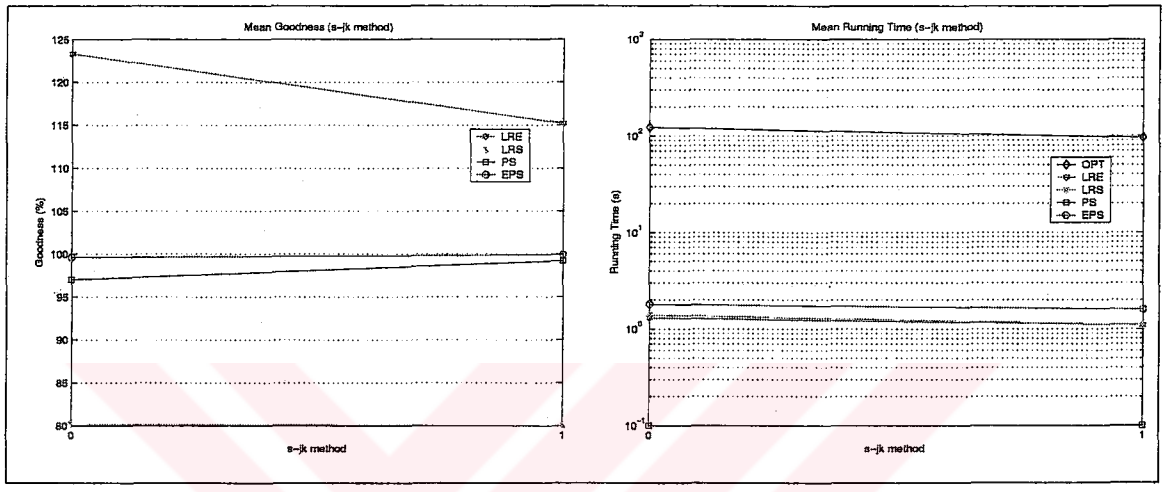


Figure 6.6. Goodness of solutions and running times: ($s_{jk}method$)

For uniform and normal distributions, LRE produces tight upper bounds with a maximum mean of approximately 105 per cent. However, for the exponential distribution, the upper bound is looser with a maximum mean of approximately 165 per cent. For LRS, results for uniform distribution provide tighter lower bound of approximately 97 per cent than for normal distribution. Like results of LRE, results for exponential distribution is far from being tightness with a mean of approximately 55 per cent. While PS results provide mean goodness of 98.1 per cent, EPS results provide a mean result of 99.8 per cent and EPS wins the round again.

In terms of running times, the pattern resembles to the s case. The optimum solutions for the normal distribution take four to thirty times more time than for the exponential distribution and again the running times of algorithms do not vary dramatically with the $s_{jk}method$.

6.1.7. Requested Amount of Resources for Each Subbid (q)

We conducted tests for $mean(q) = \{10, 20, 30, 40, 50\ per\ cent\}$ where percentage defines relativity to $\sum_{k \in s_{jk}} u_k$. The results of each distribution are given in Table B.13 and Table B.14. The mean results can be seen in Figure 6.7.

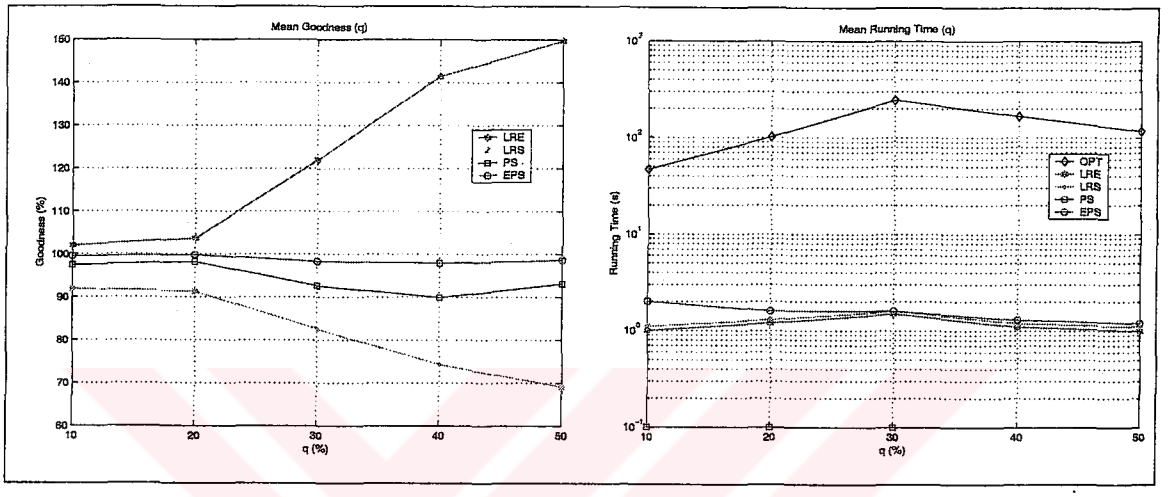


Figure 6.7. Goodness of solutions and running times: (q)

As the ratio of q increases, the results of LRE get worse in terms of goodness for all three distributions. The worst bounds are obtained from instances based on exponential distribution. Same condition is also true for LRS. When $mean(q) = 10\ per\ cent$ average goodness is approximately 92 per cent and when $mean(q) = 50\ per\ cent$ average goodness reduces to 69 per cent. Again, PS performs better than LRS, and EPS is the clear winner in terms of goodness among all greedy heuristics.

Running times of the optimum solver do not show an increasing or decreasing pattern along with q and therefore the effect of q to the difficulty of CABRC problem instance cannot be determined with these results. The speed of heuristic solvers are not affected by q directly.

6.1.8. Price Factor for ORed Requests Inside a Subbid (or_factor)

We conducted tests for $or_factor = \{0.92, 0.94, 0.96, 0.98, 1\}$. The results of each distribution are given in Table B.15 and Table B.16. The mean results can be seen in

Figure 6.8.

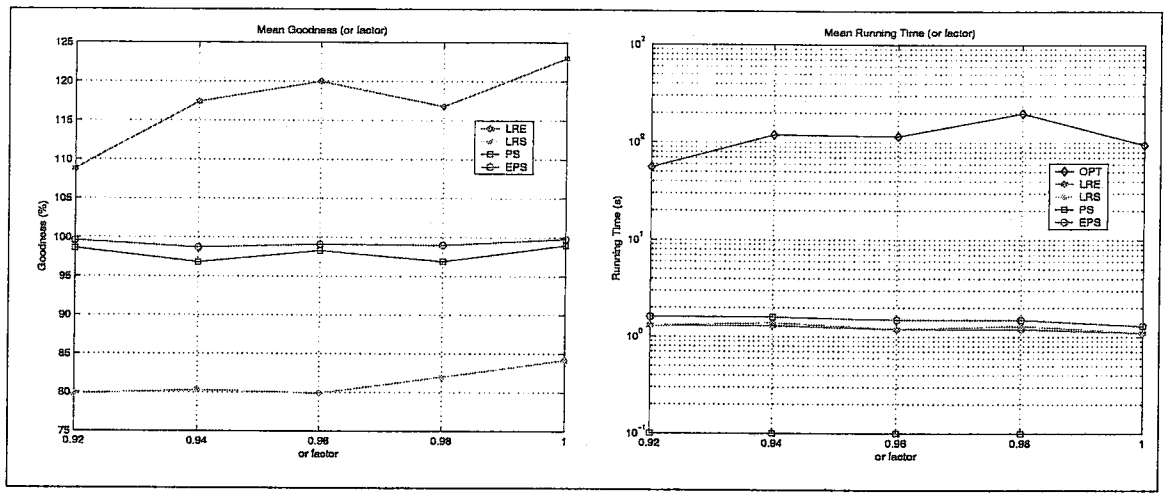


Figure 6.8. Goodness of solutions and running times: (or_factor)

Both LRE and LRS produce relatively tight upper and lower bounds respectively for uniform and normal distributions. However, both heuristics fail when parameters are distributed by an exponential distribution. The difference between results of LRE and the optimum solver and also the difference between results of LRS and the optimum are approximately 50 per cent in the worst case. Again performance of PS is much greater than LRS in terms of goodness and EPS outperforms both solvers with a mean goodness of 99.2 per cent.

The maximum average running time of the optimum solver occur when $or_factor = 0.98$, and running times of other greedy heuristics do not differ with the changes in or_factor.

6.1.9. Price Factor for ANDed Subbids Inside a Bid (and_factor)

We conducted tests for $and_factor = \{1, 1.02, 1.04, 1.06, 1.08\}$. The results of each distribution are given in Table B.17 and Table B.18. The mean results can be seen in Figure 6.9.

For $and_factor = 1$, LRE produces tighter average bounds of approximately 102 per cent and with the increase of and_factor, the mean upper bound gets relaxed to

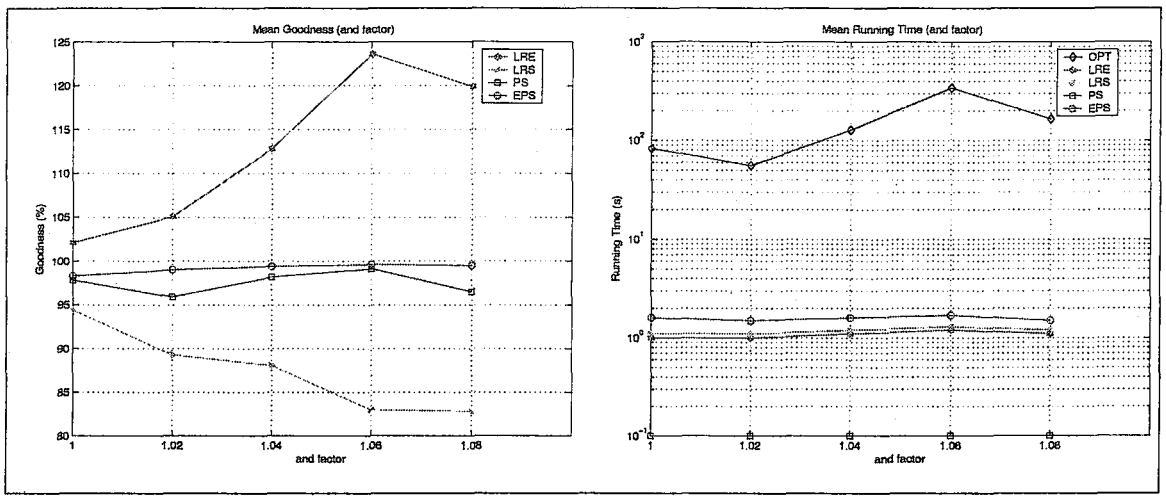


Figure 6.9. Goodness of solutions and running times: (*and_factor*)

between 120-125 per cent on the average. LRE gives best bounds with an average of 104.8 per cent for normal distribution, and gives worst results with an average of 127.9 per cent for exponential distribution. Like LRE, LRS tends to give worse results as *and_factor* increases. On the average, goodness of LRS starts with 94.4 per cent for *and_factor* = 1 and decreases to 82.8 per cent for *and_factor* = 1.08. The results of PS is 1.6 per cent lower than the results of EPS and EPS again gives the best results with the minimum mean of 97.7 per cent and corresponding standard deviation of 3.3 per cent in all tests generated for this configuration.

The maximum average running time of the optimum solver occur when *and_factor* = 1.06, and running times of other greedy heuristics do not differ with the changes in *and_factor* like the *or_factor* case.

6.1.10. Variance for Price (*price_variance*)

We conducted tests for *price_variance* = $\{10, 20, 30, 40, 50$ *per cent*$\}$ where percentage defines relativity to *raw price* of each bid. The results of each distribution are given in Table B.19 and Table B.20. The mean results can be seen in Figure 6.10.

For uniform and normal distributions, LRE produces tight upper bounds with a maximum mean of 106.7 per cent and corresponding standard deviation of 1.7 per

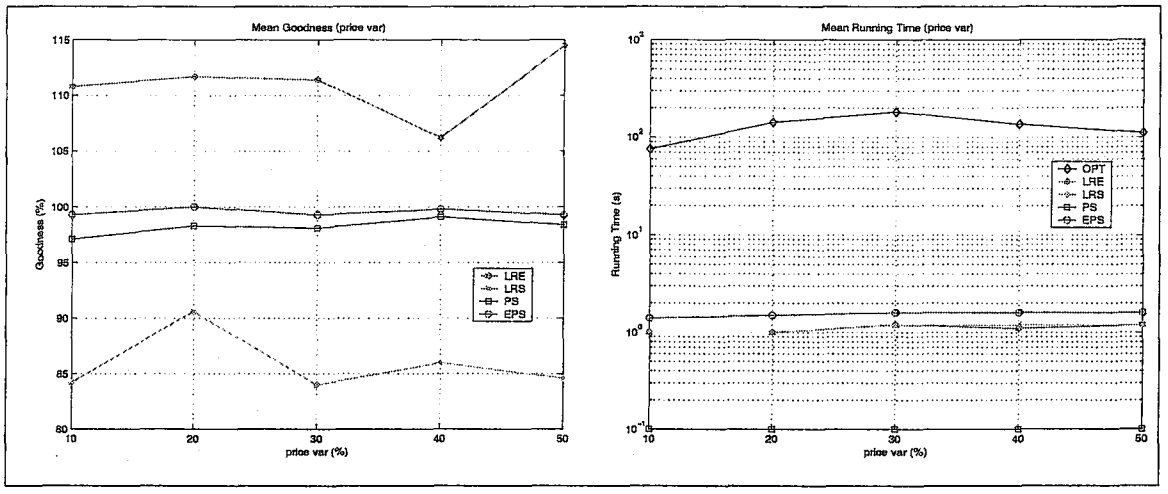


Figure 6.10. Goodness of solutions and running times: (*price_variance*)

cent. However for exponential distribution, upper bound is looser on average with a maximum mean of approximately 132 per cent. For LRS, the results are nearly same as the results of LRE. For uniform and normal distributions, the bounds of LRS are tighter than the bounds of LRS for exponential distribution. Results of EPS are the tightest among all greedy heuristic solutions with a minimum mean of 98.2 per cent and corresponding standard deviation of 3.5 per cent.

The maximum average running time of optimum solver occur when *price_variance* = 30 *per cent*, and running times of other greedy heuristics do not differ with the changes in *price_variance*.

6.1.11. Conclusion of Phase 1

In this phase of the tests, we tested the effect of ten different parameters of CABRC model using three different configurations based on uniform, normal and exponential distributions. We also discussed the general behavior of solvers under different configurations. From the results, it can be concluded that in general, an increase in parameters m,n or s make the instance of model more difficult to solve in terms of running times of the optimum solver. Secondly, it can be seen that on average, running times of optimum solver for normal distribution are more than for other two distributions. So instances of the problem which are based on normal distribution are

more difficult instances. Finally, in general, linear relaxation based heuristics, LRE and LRS, provide tighter bounds on the tests based on uniform and normal distributions and looser bounds on the tests based on exponential distribution. Also, the results obtained in this part give some information about general characteristics of solvers but in order to be more confident, more general tests are needed. Therefore, comparison of solvers will be left to the next phase of the tests.

6.2. Phase 2: General Performance

In this phase of the tests, we prepare three sets of general test configurations namely *general-1*,*general-2* and *general-3* for uniform, normal and exponential distributions separately (total of nine configurations). We fix the maximum time for the optimum solver to 1000 seconds and omit the results of all solvers if the optimum solution cannot be found. The results of these tests are given in Tables B.21-B.26. After conducting general tests, we combined the results of general tests with the results of tests explained in the first phase. After that the average goodness and running time of all 1478 solutions are calculated. The results can be seen in Figure 6.11 and Figure 6.12.

In terms of goodness, LRE gives results approximately 15 per cent higher than the optimum solutions and LRS gives results approximately 15 per cent lower than the optimum solutions on average. While PS gives a very high goodness ratio of 97.34 per cent, EPS outperforms other two greedy heuristics and gives an excellent goodness ratio of 99.2 per cent on average. In terms of running times, PS is the clear winner among all greedy heuristic solvers. However, EPS is also not very slow compared to PS but very fast compared to the optimum solver. So, overall, we can conclude that performance of EPS is excellent both in term of goodness and speed. However, if the application is time critical, PS can also be preferred with approximately 2 per cent loss on average.

By achieving 99.2 per cent goodness on average with a fast polynomial heuristic EPS, one of the main objectives of this study is accomplished.

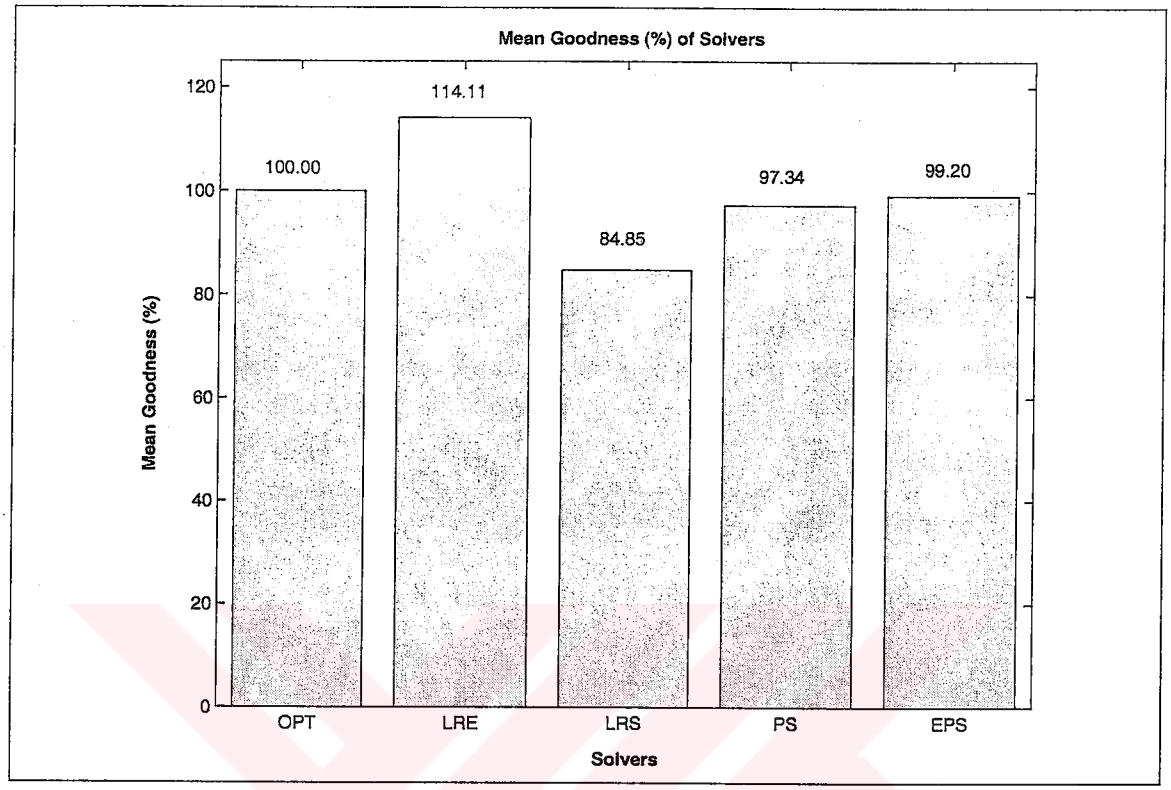


Figure 6.11. Overall goodness of solutions

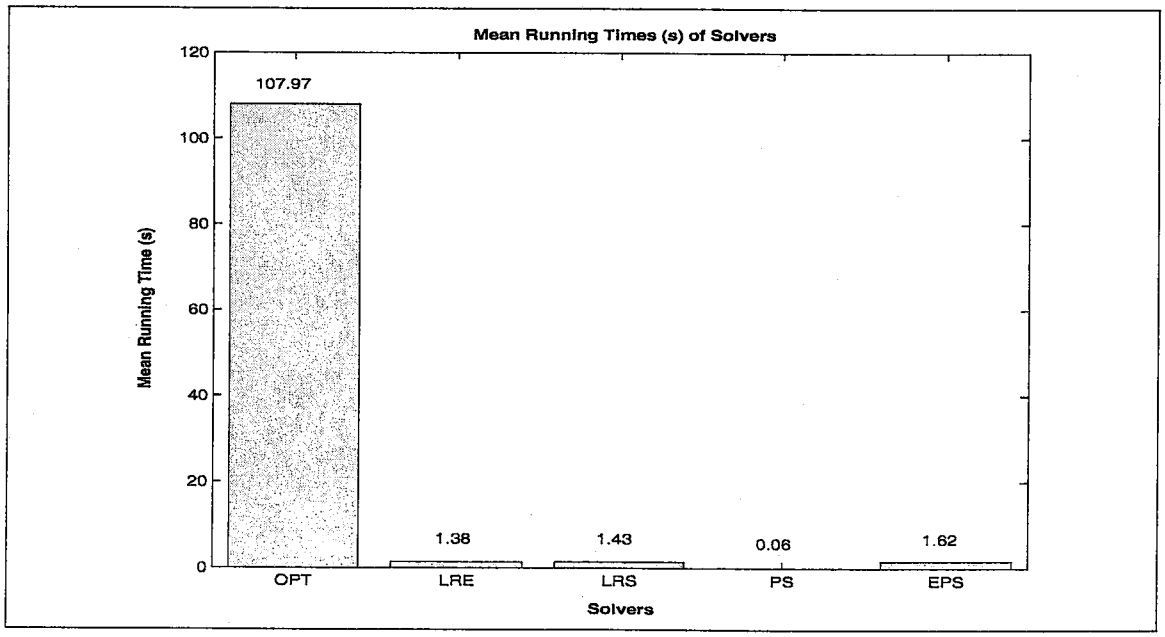


Figure 6.12. Overall running times of solutions

7. CONCLUSION

In this study, we proposed a new combinatorial auction based model for resource co-allocation problem in grids. The intention behind this economical model is to encourage resource holders to supply their idle resources in grid environment and to allow users value any combination of resources according to their preferences. Powerful bid representation of this auction based model provides higher revenues to resource suppliers and higher satisfaction to users. In this study, we examined CABRC model and its application to grids in detail. We formulated CABRC problem using IP and proved that it is NP-hard. Although it can be difficult to solve large instances of the problem, for small sized grids with relatively small number of resource types and users, optimum solution of the problem can be found quickly. However, for large grids, the optimum solution may take tremendous amount of time. Therefore, we proposed two new polynomial-time greedy heuristic solvers in order to solve large instances of the problem. We coded a solver package that consists of the optimum solver, linear relaxation based upper bound estimator, linear relaxation based greedy heuristic solver and the two mentioned solvers. Since there is no real world data, we also coded a comprehensive test case generator for CABRC problem and measured the performances of these solvers under different test conditions. Proposed two heuristic solvers, PS and EPS, obtained quite good results in the tests with 97.3 per cent and 99.2 per cent average performance relative to the optimum solution respectively.

As a feature work, CABRC model can be integrated into grid softwares such as Globus and Condor as a resource co-allocation submodule. However, before using this model commercially, if the proposed heuristics are to be used as solvers, performance of heuristics must also be confirmed under real life conditions. Therefore, before going public, it would be better to test this model in a grid testbed and evaluate the performances of heuristics in real grid environments.

APPENDIX A: DISTRIBUTIONS USED IN THE GENERATOR

The generator currently supports three different distributions and fixed value parameters in order to generate models. Accepted distributions are:

- **Uniform Distribution:** This distribution gets two parameters a and b, and generates random integers in the range $[a, b]$ that are equally likely.
  The generator uses *gsl_rng_uniform_int(const gsl_rng * r, unsigned long int n)* function of GSL library in order to generate uniform random integers. This function returns a random integer from 0 to $n-1$ inclusive such that all integers in the range $[0, n-1]$ are equally likely. The distribution is,

$$p(x) = \frac{1}{(b-a)} \tag{A.1}$$

  if $a \leq x < b$ and 0 otherwise. In order to get integers in the range $[a, b]$, the function is called with parameter $n = abs(b-a)+1$ and $min(a, b)$ is added to the return value.
- **Normal Distribution:** This distribution gets two parameters μ and σ, and generates normally distributed random real or integer numbers with *mean* $= \mu$ and *standard deviation* $= \sigma$.
  The generator uses *gsl_ran_gaussian(const gsl_rng * r, double sigma)* function of GSL library in order to generate normally distributed real numbers. This function returns a Gaussian random variate, with mean zero and standard deviation *sigma*. The probability distribution for Gaussian random variates is,

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-x^2/2\sigma^2) \tag{A.2}$$

  for x in the range $-\infty$ to $+\infty$. In order to get numbers with *mean* μ and *standard deviation* σ, the transformation $z = \mu + x$ is applied to the numbers

returned by this function. If integer numbers are required by the generator instead of real numbers, real numbers are rounded to the nearest integers.

- **Exponential Distribution:** This distribution gets one parameter μ and generates exponentially distributed random real or integer numbers with $mean = \mu$. The generator uses *gsl_ran_exponential(const gsl_rng * r, double* μ) function of GSL library in order to generate exponentially distributed real numbers. This function returns a random variate from the exponential distribution with mean μ. The distribution is,

$$p(x) = \frac{1}{\mu} \exp(-x/\mu) \tag{A.3}$$

for $x \geq 0$. If integer numbers are required by the generator instead of real numbers, first mean is shifted to left by 0.5 and gsl_ran_gaussian function is called with shifted mean, then returned numbers are shifted to right by 0.5, and finally numbers are rounded to the nearest integers.

APPENDIX B: TABLES OF TEST RESULTS

Table B.1. Goodness of solutions (%): m

	m	20		40		60		80		100	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
LRE	**U**	103.1	2.2	103.2	2.0	102.7	1.2	102.1	0.8	103.2	1.9
	N	103.4	0.8	104.2	0.7	103.6	0.9	103.6	0.8	104.2	1.8
	E	109.7	6.5	152.0	26.6	214.8	45.5	205.8	113.4	144.4	39.6
LRS	**U**	98.2	1.3	91.7	6.2	91.0	6.4	95.2	3.1	89.6	5.9
	N	95.0	4.3	90.6	4.5	97.4	3.6	92.4	1.8	94.1	6.1
	E	87.9	7.8	74.2	9.8	52.7	9.0	68.9	26.3	75.1	17.8
PS	**U**	98.7	0.9	99.2	0.8	99.0	0.3	99.0	0.9	99.5	0.4
	N	96.8	3.6	95.1	3.4	100.0	0.0	95.3	3.5	98.4	1.4
	E	90.8	11.7	98.4	2.3	99.9	0.1	96.7	3.8	99.5	0.4
EPS	**U**	99.6	0.6	99.4	0.4	99.7	0.5	99.0	0.9	99.5	0.4
	N	97.5	3.3	97.6	3.1	100.0	0.0	99.6	0.7	98.6	1.4
	E	98.5	2.6	99.8	0.2	99.9	0.1	98.4	2.2	99.6	0.3

Table B.2. Running times (s): m

	m	20		40		60		80		100	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
OPT	**U**	5.6	5.1	93.2	118.7	61.5	60.8	91.3	97.3	175.3	45.5
	N	31.9	32.8	117.3	157.5	45.0	23.9	112.5	41.5	380.7	295.0
	E	4.9	2.9	18.0	9.5	44.1	18.4	118.8	113.3	177.8	218.8
LRE	**U**	0.4	0.2	1.0	0.6	1.8	1.0	2.2	1.3	2.7	1.3
	N	0.4	0.2	0.9	0.5	0.6	0.1	1.7	1.2	2.3	1.7
	E	0.4	0.2	1.6	0.7	2.8	1.3	4.5	3.3	4.7	2.0
LRS	**U**	0.5	0.2	1.1	0.6	1.9	1.0	2.3	1.4	2.8	1.4
	N	0.5	0.2	0.9	0.5	0.7	0.1	1.8	1.2	2.4	1.7
	E	0.4	0.2	1.6	0.8	2.9	1.3	4.6	3.3	4.9	2.1
PS	**U**	0.0	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0
	N	0.1	0.0	0.1	0.0	0.0	0.0	0.1	0.0	0.1	0.0
	E	0.0	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.1
EPS	**U**	1.1	0.4	1.6	0.6	2.2	0.8	2.4	0.9	2.8	1.1
	N	1.2	0.4	1.5	0.6	1.2	0.0	2.0	0.9	2.2	0.9
	E	1.2	0.5	1.6	0.7	1.9	0.9	2.5	1.0	3.4	1.5

Table B.3. Goodness of solutions (%): u

	u	2		4		8		16		32	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
	U	105.8	2.5	104.6	2.1	101.5	1.1	103.4	1.4	106.3	2.2
LRE	**N**	103.5	2.0	103.5	0.0	104.1	1.7	104.7	3.4	103.9	1.6
	E	125.4	22.7	126.5	22.4	114.3	22.7	111.5	3.6	111.2	12.9
	U	94.5	4.1	91.7	7.9	96.2	4.9	86.5	9.3	87.5	8.2
LRS	**N**	95.0	3.8	78.1	0.0	94.1	4.9	91.9	5.4	96.3	5.0
	E	78.4	20.5	71.0	24.8	81.2	16.2	74.7	23.2	79.9	24.3
	U	99.1	1.3	97.5	3.3	99.2	1.6	97.6	4.8	97.0	2.3
PS	**N**	95.8	7.4	96.7	0.0	97.9	2.0	91.5	6.1	98.8	2.0
	E	99.7	0.4	99.3	0.7	99.6	0.6	98.2	1.4	98.7	2.4
	U	99.4	1.1	98.7	2.2	99.9	0.2	100.0	0.1	99.4	0.8
EPS	**N**	99.2	1.4	98.9	0.0	98.8	2.3	97.7	2.3	98.9	1.9
	E	99.7	0.4	100.0	0.0	99.9	0.2	99.2	0.7	98.8	2.4

Table B.4. Running times (s): u

	m	2		4		8		16		32	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
OPT	**U**	59.3	58.0	80.9	102.8	51.5	97.2	35.1	36.0	178.2	223.5
	N	98.5	133.9	172.5	0.0	65.0	63.6	183.9	262.1	263.0	245.4
	E	64.8	85.4	20.1	18.3	17.1	12.7	28.3	15.2	29.1	37.7
LRE	**U**	0.9	0.4	0.7	0.3	0.9	0.5	0.9	0.5	0.8	0.6
	N	1.0	0.7	1.2	0.0	0.9	0.6	0.7	0.5	0.9	0.6
	E	1.5	0.8	1.5	1.0	1.6	0.9	1.5	0.7	1.5	0.5
LRS	**U**	0.9	0.5	0.7	0.4	1.0	0.5	1.0	0.5	0.9	0.6
	N	1.1	0.8	1.3	0.0	1.0	0.6	0.8	0.5	1.0	0.6
	E	1.5	0.8	1.5	1.0	1.6	1.0	1.6	0.7	1.5	0.5
PS	**U**	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0
	N	0.1	0.0	0.1	0.0	0.1	0.0	0.0	0.0	0.1	0.0
	E	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0
EPS	**U**	1.5	0.6	1.4	0.6	1.5	0.6	1.5	0.6	1.4	0.6
	N	1.4	0.6	2.0	0.0	1.5	0.6	1.3	0.6	1.4	0.6
	E	1.6	0.6	1.6	0.6	1.6	0.7	1.6	0.6	1.6	0.7

Table B.5. Goodness of solutions (%): n

	n	100		200		300		400		500	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
LRE	**U**	105.8	0.7	107.3	1.8	102.5	1.0	108.9	0.0	101.9	0.2
	N	102.4	0.5	103.9	0.9	103.0	2.1	103.3	0.0	104.0	0.0
	E	103.2	1.0	105.8	2.1	106.9	0.3	114.9	7.9	121.5	27.2
LRS	**U**	77.3	7.2	88.3	2.6	92.3	7.9	83.8	0.0	92.3	1.8
	N	91.7	6.3	97.3	2.6	95.7	6.1	90.8	0.0	97.8	0.0
	E	95.7	4.4	61.7	41.1	91.5	5.4	50.6	6.7	72.2	34.7
PS	**U**	94.9	4.7	98.5	2.1	98.5	2.1	92.4	0.0	97.9	3.0
	N	96.6	0.7	97.3	2.6	100.0	0.0	97.9	0.0	97.8	0.0
	E	97.4	1.1	99.6	0.6	96.2	5.4	96.6	4.8	98.9	0.9
EPS	**U**	99.4	0.9	99.2	1.2	99.8	0.2	98.5	0.0	100.0	0.0
	N	98.6	2.0	99.6	0.6	100.0	0.0	98.2	0.0	100.0	0.0
	E	97.9	1.8	99.9	0.1	100.0	0.0	96.6	4.8	99.1	1.0

Table B.6. Running times (s): n

	n	100		200		300		400		500	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
OPT	**U**	148.4	140.5	111.1	81.7	109.4	73.4	269.0	0.0	233.8	241.9
	N	25.7	3.8	102.1	72.1	128.4	164.2	139.3	0.0	1266.5	0.0
	E	6.7	0.8	25.8	20.7	230.9	235.6	38.2	9.1	377.1	237.6
LRE	**U**	0.5	0.1	1.4	0.0	3.0	0.3	5.1	0.0	7.0	0.8
	N	0.4	0.0	1.3	0.1	3.2	0.4	4.7	0.0	8.5	0.0
	E	0.7	0.0	2.6	0.1	3.7	0.0	5.4	0.2	9.5	0.8
LRS	**U**	0.6	0.1	1.5	0.1	3.1	0.3	5.3	0.0	7.1	0.8
	N	0.4	0.0	1.3	0.1	3.3	0.4	4.9	0.0	8.6	0.0
	E	0.7	0.0	2.6	0.1	3.9	0.1	5.6	0.1	9.7	0.7
PS	**U**	0.0	0.0	0.1	0.0	0.1	0.0	0.2	0.0	0.2	0.0
	N	0.0	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.2	0.0
	E	0.0	0.0	0.1	0.0	0.1	0.0	0.2	0.0	0.2	0.0
EPS	**U**	1.0	0.1	2.0	0.1	3.3	0.1	4.5	0.0	5.5	0.0
	N	1.0	0.0	1.9	0.1	2.9	0.1	3.8	0.0	4.7	0.0
	E	1.1	0.0	2.2	0.1	3.5	0.2	4.9	0.1	5.8	0.3

Table B.7. Goodness of solutions (%): t

	t	3		7		11		15		19	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
LRE	**U**	100.4	0.3	104.6	2.5	107.1	4.2	115.3	18.6	132.4	24.7
	N	100.8	0.5	104.3	1.2	106.7	3.0	107.6	2.9	112.4	0.0
	E	101.8	0.9	124.0	17.5	159.7	22.8	142.2	27.9	122.6	0.2
LRS	**U**	96.0	5.1	97.5	1.7	82.4	24.5	71.3	23.9	67.8	6.4
	N	98.5	1.3	97.4	3.2	89.6	14.1	100.0	0.0	40.3	0.0
	E	95.0	3.8	64.4	15.1	53.5	9.1	50.1	19.4	43.6	9.4
PS	**U**	99.1	0.9	98.7	1.2	100.0	0.0	100.0	0.0	97.0	4.3
	N	98.0	2.1	98.1	2.1	96.9	5.3	100.0	0.0	92.4	0.0
	E	97.0	2.4	96.9	4.9	98.0	2.3	99.8	0.3	100.0	0.1
EPS	**U**	99.2	0.8	99.5	0.6	100.0	0.0	100.0	0.0	100.0	0.0
	N	98.8	1.7	99.0	0.9	100.0	0.0	100.0	0.0	100.0	0.0
	E	97.6	2.4	97.3	5.1	98.4	1.6	99.9	0.2	100.0	0.1

Table B.8. Running times (s): t

	t	3		7		11		15		19	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
OPT	**U**	1.8	1.9	87.2	101.6	42.4	18.9	289.3	376.5	64.8	40.7
	N	18.1	20.9	65.0	33.4	294.0	188.4	43.5	18.0	741.5	0.0
	E	1.4	0.4	17.2	11.9	69.0	40.0	200.0	259.9	32.6	9.9
LRE	**U**	0.2	0.1	0.7	0.3	1.7	1.1	2.3	1.7	2.1	0.1
	N	0.2	0.1	0.9	0.5	1.9	1.0	5.2	1.4	10.3	0.0
	E	0.4	0.2	1.5	0.9	2.2	0.9	1.8	1.0	1.3	0.0
LRS	**U**	0.3	0.1	0.7	0.3	1.7	1.2	2.3	1.7	2.1	0.1
	N	0.3	0.1	1.0	0.5	2.0	1.0	5.2	1.4	10.5	0.0
	E	0.5	0.2	1.6	0.9	2.3	0.9	1.9	1.0	1.4	0.0
PS	**U**	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.0	0.0
	N	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0
	E	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.0	0.0
EPS	**U**	1.6	0.6	1.5	0.6	1.5	0.8	1.3	0.6	1.0	0.0
	N	1.4	0.5	1.6	0.5	1.6	0.6	1.9	0.1	1.9	0.0
	E	1.7	0.7	1.7	0.6	1.9	0.6	1.5	0.7	1.1	0.0

Table B.9. Goodness of solutions (%): s

	s	10%		20%		30%		40%		50%	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
LRE	**U**	103.9	1.8	104.4	2.3	105.5	3.8	110.5	10.5	115.4	3.8
	N	102.9	2.0	104.3	2.9	105.9	3.6	114.0	0.0	110.3	3.8
	E	120.7	17.3	110.4	9.4	145.2	45.5	135.9	21.2	147.5	26.1
LRS	**U**	95.4	3.2	91.5	3.3	85.8	17.6	78.9	21.1	72.1	17.7
	N	95.9	3.2	96.6	4.8	88.8	11.3	100.0	0.0	86.9	17.1
	E	92.5	2.8	84.5	21.6	76.1	19.5	65.2	17.4	64.1	7.1
PS	**U**	95.4	8.3	98.9	1.3	98.6	1.9	99.2	1.6	98.1	2.1
	N	95.3	5.4	99.6	0.5	97.5	2.2	100.0	0.0	96.2	6.2
	E	97.3	0.6	99.4	0.6	98.5	1.9	99.9	0.1	99.2	1.4
EPS	**U**	99.1	1.3	99.9	0.1	100.0	0.0	100.0	0.0	99.5	0.8
	N	98.2	2.7	100.0	0.0	98.3	1.5	100.0	0.0	99.4	1.1
	E	99.3	0.6	99.7	0.3	99.6	0.8	99.9	0.1	100.0	0.0

Table B.10. Running times (s): s

	s	10%		20%		30%		40%		50%	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
OPT	**U**	53.1	27.7	47.5	40.5	251.3	363.5	117.2	130.1	61.1	29.9
	N	540.5	382.3	279.0	427.0	235.1	267.2	271.9	0.0	349.6	345.1
	E	62.3	64.7	34.3	46.5	71.5	57.0	40.0	24.8	80.1	20.9
LRE	**U**	0.7	0.3	1.0	0.6	1.5	0.9	2.0	1.3	2.1	1.9
	N	0.7	0.3	0.9	0.5	1.1	0.9	2.6	0.0	2.6	2.0
	E	1.3	0.6	1.5	0.7	1.8	0.9	1.9	0.7	1.8	1.2
LRS	**U**	0.8	0.3	1.1	0.7	1.5	1.0	2.0	1.3	2.2	1.9
	N	0.7	0.4	1.0	0.6	1.1	0.9	2.7	0.0	2.7	2.1
	E	1.4	0.6	1.6	0.7	1.8	0.9	1.9	0.8	1.8	1.2
PS	**U**	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0
	N	0.1	0.0	0.1	0.0	0.0	0.0	0.1	0.0	0.1	0.0
	E	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0
EPS	**U**	1.7	0.7	1.6	0.8	1.5	0.5	1.5	0.5	1.3	0.5
	N	1.6	0.6	1.5	0.6	1.3	0.6	1.9	0.0	1.4	0.5
	E	1.7	0.7	1.7	0.7	1.6	0.6	1.6	0.7	1.4	0.7

Table B.11. Goodness of solutions (%): $s_{jk}method$

	s_jk_method	1		2	
		μ	σ	μ	σ
LRE	**U**	102.3	0.9	103.3	1.8
	N	103.2	0.0	105.2	0.5
	E	164.4	10.8	137.1	24.3
LRS	**U**	96.4	1.9	98.0	1.7
	N	86.6	0.0	90.8	5.7
	E	57.6	4.1	51.1	2.5
PS	**U**	99.9	0.2	99.7	0.4
	N	95.2	0.0	99.6	0.6
	E	96.1	5.6	98.3	2.1
EPS	**U**	100.0	0.0	99.7	0.4
	N	99.1	0.0	100.0	0.0
	E	99.9	0.2	99.9	0.1

Table B.12. Running times (s): $s_{jk}method$

	s_jk_method	1		2	
		μ	σ	μ	σ
OPT	**U**	48.8	46.2	73.8	97.6
	N	309.1	0.0	171.5	161.9
	E	10.7	1.5	41.5	34.5
LRE	**U**	0.9	0.5	0.8	0.5
	N	1.5	0.0	1.0	0.7
	E	1.6	1.2	1.5	1.1
LRS	**U**	0.9	0.6	0.9	0.5
	N	1.5	0.0	1.0	0.8
	E	1.6	1.2	1.6	1.1
PS	**U**	0.1	0.0	0.1	0.0
	N	0.1	0.0	0.1	0.0
	E	0.1	0.0	0.1	0.0
EPS	**U**	1.6	0.9	1.6	0.8
	N	2.0	0.0	1.5	0.7
	E	1.7	0.9	1.8	0.8

Table B.13. Goodness of solutions (%): q

	q	10%		20%		30%		40%		50%	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
LRE	**U**	101.1	0.8	103.2	0.8	105.8	1.4	127.2	11.9	151.0	3.1
	N	101.0	0.2	103.3	1.0	105.2	0.0	117.4	6.6	123.0	12.3
	E	104.0	3.6	104.6	5.1	154.8	23.7	179.5	46.4	175.3	43.2
LRS	**U**	90.2	0.7	94.5	3.0	84.9	17.7	74.9	23.9	73.1	2.1
	N	95.6	3.9	94.2	3.5	88.8	0.0	82.6	24.6	71.6	17.1
	E	89.8	7.3	85.1	26.0	74.3	11.6	65.6	9.0	62.6	10.2
PS	**U**	98.4	0.9	96.8	1.2	95.7	6.9	97.1	5.1	94.0	2.1
	N	98.6	0.7	97.8	4.4	88.8	0.0	82.6	24.6	88.7	7.4
	E	95.2	7.6	99.9	0.2	93.2	5.2	90.1	5.3	96.6	2.7
EPS	**U**	98.8	1.2	99.4	0.7	99.3	1.1	99.8	0.5	97.9	1.3
	N	99.7	0.2	100.0	0.0	97.5	0.0	98.6	2.0	100.0	0.0
	E	100.0	0.1	99.9	0.2	98.2	1.5	95.4	3.7	97.7	3.0

Table B.14. Running times (s): q

	q	10%		20%		30%		40%		50%	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
OPT	**U**	43.7	61.7	166.3	218.7	251.8	320.0	281.8	282.7	116.0	5.3
	N	91.5	92.3	131.3	155.0	401.5	0.0	72.8	59.9	164.4	259.5
	E	4.8	4.7	9.1	4.3	84.5	101.4	147.4	227.9	70.5	81.6
LRE	**U**	0.9	0.5	1.1	0.7	1.0	0.5	1.0	0.6	0.5	0.0
	N	0.8	0.5	1.0	0.7	1.8	0.0	1.0	0.8	1.0	0.7
	E	1.5	0.7	1.6	0.8	1.7	0.9	1.4	0.8	1.6	0.8
LRS	**U**	1.0	0.6	1.1	0.7	1.1	0.6	1.1	0.6	0.6	0.0
	N	0.8	0.5	1.0	0.7	1.8	0.0	1.0	0.8	1.0	0.7
	E	1.6	0.7	1.7	0.9	1.7	0.9	1.4	0.9	1.7	0.8
PS	**U**	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.0	0.0
	N	0.1	0.0	0.1	0.0	0.1	0.0	0.0	0.0	0.0	0.0
	E	0.1	0.0	0.1	0.0	0.1	0.0	0.0	0.0	0.1	0.0
EPS	**U**	2.0	0.8	1.6	0.6	1.4	0.6	1.3	0.5	0.9	0.0
	N	1.9	0.7	1.5	0.6	1.7	0.0	1.2	0.6	1.2	0.5
	E	2.2	0.9	1.7	0.7	1.6	0.7	1.3	0.6	1.5	0.6

Table B.15. Goodness of solutions (%): *or_factor*

	or	0.92		0.94		0.96		0.98		1	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
LRE	**U**	104.6	2.1	104.7	5.1	103.6	2.1	104.4	3.2	105.9	1.0
	N	103.3	2.2	102.9	1.0	104.4	2.5	104.1	1.7	103.9	0.0
	E	118.4	15.8	144.6	30.6	152.0	30.9	141.8	34.8	159.4	24.6
LRS	**U**	87.5	3.8	94.0	6.9	93.6	3.5	91.3	7.4	89.9	6.3
	N	94.7	9.1	96.2	1.7	95.7	3.7	93.4	6.1	94.8	0.0
	E	57.3	17.3	50.8	7.2	50.4	8.8	61.4	7.7	68.0	4.4
PS	**U**	97.6	2.5	97.1	4.8	99.5	0.7	96.6	3.7	99.2	0.7
	N	99.4	1.2	95.2	4.9	97.2	2.1	94.8	2.9	98.2	0.0
	E	98.7	2.4	98.0	1.7	98.2	2.0	99.3	0.7	99.7	0.4
EPS	**U**	99.1	1.2	99.3	0.7	99.5	0.7	98.7	1.7	99.6	0.5
	N	99.8	0.4	96.9	3.4	99.1	1.6	98.4	1.5	100.0	0.0
	E	99.9	0.1	100.0	0.1	98.7	1.7	99.9	0.2	99.7	0.4

Table B.16. Running times (s): *or_factor*

	or	0.92		0.94		0.96		0.98		1	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
OPT	**U**	85.7	107.9	64.1	33.7	104.1	111.6	154.2	163.2	80.3	38.8
	N	66.1	89.3	272.0	205.1	72.0	60.9	410.2	442.4	42.3	0.0
	E	14.5	13.8	21.9	23.3	170.6	196.1	33.7	51.5	161.5	178.6
LRE	**U**	1.1	0.6	1.1	0.7	1.0	0.6	1.1	0.7	1.0	0.8
	N	0.9	0.5	1.1	0.6	0.7	0.5	1.0	0.7	0.4	0.0
	E	1.8	0.9	1.7	0.8	1.8	0.8	1.5	0.9	1.8	0.9
LRS	**U**	1.1	0.7	1.2	0.7	1.1	0.6	1.1	0.7	1.0	0.8
	N	1.0	0.5	1.2	0.6	0.8	0.5	1.1	0.7	0.4	0.0
	E	1.8	0.9	1.8	0.8	1.8	0.9	1.6	0.9	1.9	1.0
PS	**U**	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.0	0.0
	N	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.0	0.0
	E	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0
EPS	**U**	1.6	0.6	1.6	0.6	1.6	0.6	1.6	0.6	1.4	0.6
	N	1.4	0.6	1.6	0.6	1.3	0.6	1.5	0.6	1.0	0.0
	E	1.8	0.6	1.8	0.6	1.8	0.7	1.5	0.6	1.7	0.6

Table B.17. Goodness of solutions (%): *and_factor*

	and	1		1.02		1.04		1.06		1.08	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
LRE	**U**	102.1	1.2	104.2	2.2	104.7	2.3	108.4	5.2	107.9	3.2
	N	102.7	0.7	103.4	0.9	105.6	4.3	106.5	1.5	105.8	3.3
	E	101.6	1.4	107.9	3.8	128.3	21.6	155.8	42.0	145.9	59.2
LRS	**U**	95.3	1.7	93.9	5.0	92.6	5.0	89.3	7.6	84.7	4.9
	N	95.5	5.5	91.9	5.7	99.3	1.2	95.6	4.2	90.9	10.8
	E	92.4	5.0	82.3	16.2	72.5	29.4	64.2	36.9	72.8	40.4
PS	**U**	98.9	1.3	96.6	3.9	98.6	1.8	100.0	0.0	94.0	4.5
	N	96.6	2.5	94.3	5.4	97.2	4.9	98.6	1.3	96.2	5.3
	E	97.8	1.7	96.8	2.7	98.8	1.9	98.6	1.7	99.2	0.7
EPS	**U**	99.1	0.9	99.9	0.2	98.6	1.8	100.0	0.0	98.5	2.7
	N	97.7	3.3	99.0	1.3	100.0	0.0	99.9	0.2	100.0	0.0
	E	98.0	2.0	98.0	2.8	99.5	0.6	98.9	1.9	100.0	0.0

Table B.18. Running times (s): *and_factor*

	and	1		1.02		1.04		1.06		1.08	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
OPT	**U**	71.4	94.9	47.0	33.9	51.0	6.3	162.2	270.9	185.4	189.2
	N	144.5	224.9	106.2	125.9	115.5	84.0	375.2	258.7	239.2	331.8
	E	32.1	53.4	13.9	10.2	217.5	412.4	493.4	446.9	67.3	105.9
LRE	**U**	0.9	0.5	0.9	0.5	0.8	0.6	0.9	0.5	1.0	0.6
	N	0.7	0.6	1.0	0.6	1.1	0.5	1.0	0.6	1.1	0.7
	E	1.5	0.9	1.2	0.9	1.5	0.9	1.7	0.9	1.2	0.9
LRS	**U**	1.0	0.5	1.0	0.5	0.9	0.6	1.0	0.6	1.1	0.6
	N	0.8	0.6	1.0	0.6	1.2	0.5	1.0	0.6	1.1	0.7
	E	1.6	1.0	1.3	0.9	1.5	1.0	1.8	0.9	1.3	0.9
PS	**U**	0.1	0.0	0.1	0.0	0.0	0.0	0.1	0.0	0.1	0.0
	N	0.0	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0
	E	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0
EPS	**U**	1.6	0.6	1.6	0.6	1.4	0.7	1.6	0.6	1.6	0.6
	N	1.4	0.6	1.5	0.6	1.7	0.6	1.5	0.6	1.5	0.6
	E	1.8	0.7	1.5	0.7	1.8	0.8	2.0	0.7	1.6	0.8

Table B.19. Goodness of solutions (%): *price_variance*

	pv	10%		20%		30%		40%		50%	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
LRE	**U**	103.5	0.8	102.7	1.9	108.8	1.7	105.0	1.3	103.8	2.2
	N	103.7	2.0	104.4	0.7	104.8	3.0	104.5	2.2	106.7	1.7
	E	125.2	25.5	127.9	46.8	120.7	19.3	109.1	5.9	132.9	31.3
LRS	**U**	87.3	7.7	93.0	8.0	82.8	9.8	94.0	5.1	87.2	8.9
	N	92.8	5.2	96.2	4.6	91.0	5.1	93.6	7.3	97.6	0.4
	E	72.5	24.9	82.5	28.3	78.1	21.6	70.4	29.7	69.1	32.2
PS	**U**	97.5	1.0	98.0	4.0	97.6	2.8	98.8	0.8	97.9	3.5
	N	94.5	7.7	97.0	3.5	97.0	1.9	98.9	2.1	98.1	1.3
	E	99.3	0.2	99.9	0.2	99.7	0.5	99.6	0.4	99.2	0.8
EPS	**U**	98.7	0.8	100.0	0.0	99.3	0.7	99.9	0.3	98.2	3.5
	N	100.0	0.0	100.0	0.0	98.8	1.0	99.6	0.8	100.0	0.0
	E	99.3	0.3	99.9	0.2	99.7	0.5	99.9	0.2	99.6	0.3

Table B.20. Running times (s): *price_variance*

	pv	10%		20%		30%		40%		50%	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
OPT	**U**	89.4	88.9	45.3	71.6	171.6	143.8	126.5	197.3	46.7	71.1
	N	53.9	17.9	359.7	313.9	349.1	403.4	85.1	103.4	258.8	337.8
	E	83.0	102.2	22.2	34.2	25.4	17.3	195.2	353.5	28.7	35.6
LRE	**U**	1.0	0.7	1.0	0.7	1.1	0.7	0.9	0.5	1.0	0.6
	N	0.4	0.0	0.8	0.6	0.9	0.6	0.9	0.6	0.9	0.6
	E	1.5	1.1	1.2	1.2	1.5	1.1	1.6	1.2	1.5	1.2
LRS	**U**	1.1	0.7	1.1	0.7	1.2	0.8	1.0	0.6	1.1	0.6
	N	0.4	0.0	0.8	0.6	1.0	0.6	1.0	0.6	1.0	0.6
	E	1.6	1.2	1.3	1.3	1.6	1.2	1.6	1.3	1.6	1.2
PS	**U**	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0
	N	0.0	0.0	0.0	0.0	0.1	0.0	0.1	0.0	0.1	0.0
	E	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0
EPS	**U**	1.6	0.6	1.6	0.6	1.6	0.6	1.5	0.5	1.6	0.6
	N	1.0	0.0	1.3	0.6	1.5	0.6	1.5	0.6	1.5	0.6
	E	1.7	0.7	1.5	0.7	1.7	0.7	1.7	0.7	1.7	0.7

Table B.21. Goodness of solutions (%): general-1

	n	50		100		150		200		250	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
LRE	**U**	100.5	0.9	100.1	0.3	100.3	0.6	100.2	0.3	100.2	0.3
	N	102.9	2.1	102.2	1.3	102.4	1.7	102.3	1.6	102.3	1.7
	E	101.8	2.3	102.2	2.5	103.2	5.2	101.8	2.0	101.6	2.3
LRS	**U**	99.3	1.2	99.0	1.8	98.6	2.5	98.6	1.5	98.4	2.7
	N	95.5	5.0	95.8	2.6	95.3	3.6	94.5	2.9	94.4	2.6
	E	98.1	3.5	94.9	4.3	94.4	7.2	96.1	3.6	96.2	3.3
PS	**U**	97.7	3.8	96.7	4.3	97.2	3.4	97.7	2.8	97.9	2.4
	N	96.0	5.4	96.9	2.2	97.3	2.4	96.8	3.1	96.4	3.2
	E	94.7	5.1	94.5	5.5	93.2	5.9	94.6	4.7	95.9	3.7
EPS	**U**	98.4	3.0	98.8	1.8	98.5	2.1	98.8	1.7	99.3	0.8
	N	98.2	2.1	98.0	1.8	98.5	1.3	98.3	1.6	98.5	1.4
	E	97.7	2.8	98.5	2.0	97.8	2.7	98.4	2.2	98.9	1.1

Table B.22. Running times (s): general-1

	n	50		100		150		200		250	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
OPT	**U**	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.2	0.1	0.1
	N	0.1	0.1	0.2	0.2	0.5	0.5	0.8	0.9	0.8	1.1
	E	0.0	0.0	0.1	0.0	0.2	0.1	0.2	0.1	0.2	0.2
LRE	**U**	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.1	0.1
	N	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.1	0.1
	E	0.0	0.0	0.1	0.0	0.1	0.1	0.1	0.1	0.2	0.2
LRS	**U**	0.0	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.2	0.1
	N	0.0	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.2	0.0
	E	0.0	0.0	0.1	0.0	0.2	0.1	0.2	0.1	0.3	0.2
PS	**U**	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.1	0.0
	N	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.1	0.0
	E	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.1	0.0
EPS	**U**	0.2	0.1	0.4	0.3	0.7	0.4	0.9	0.6	1.1	0.8
	N	0.2	0.1	0.7	0.0	1.1	0.1	1.4	0.1	1.8	0.1
	E	0.3	0.0	0.7	0.0	1.1	0.1	1.4	0.1	1.8	0.1

Table B.23. Goodness of solutions (%): general-2

	n	50		100		150		200		250	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
LRE	**U**	102.2	1.7	101.8	1.3	101.9	1.7	101.2	1.3	101.3	1.0
	N	102.0	1.5	101.9	1.4	101.7	1.3	101.8	1.0	101.6	1.0
	E	102.4	2.6	106.7	7.5	102.9	3.0	103.0	2.5	108.1	7.9
LRS	**U**	93.8	5.4	95.7	3.3	94.4	4.7	93.6	6.4	95.5	3.3
	N	96.5	2.2	96.5	2.3	95.3	3.3	96.9	2.4	96.2	2.0
	E	94.9	5.6	86.6	16.4	89.1	11.7	94.6	5.8	89.8	13.3
PS	**U**	98.5	2.2	98.8	1.3	98.2	2.9	98.9	1.7	98.2	2.2
	N	97.6	1.9	96.6	1.9	97.1	2.4	97.4	2.0	97.4	1.7
	E	98.8	1.5	97.9	3.3	98.4	2.6	99.2	0.9	99.4	1.1
EPS	**U**	99.4	1.3	99.6	1.0	99.4	0.9	99.8	0.3	99.4	0.9
	N	98.9	1.2	99.0	1.1	98.8	1.5	99.2	0.9	99.0	0.8
	E	99.5	0.7	99.4	0.9	99.4	1.1	99.4	0.8	99.5	0.8

Table B.24. Running times (s): general-2

	n	50		100		150		200		250	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
OPT	**U**	1.7	2.1	5.5	3.9	21.8	48.1	16.2	21.4	38.3	33.3
	N	26.2	88.8	21.1	17.2	31.9	35.6	83.7	132.7	125.6	102.8
	E	0.9	0.7	4.2	5.4	10.2	11.2	10.4	10.9	23.3	24.6
LRE	**U**	0.1	0.0	0.3	0.1	0.5	0.2	0.8	0.3	1.2	0.4
	N	0.1	0.0	0.3	0.1	0.5	0.2	0.7	0.3	1.1	0.4
	E	0.2	0.1	0.6	0.1	0.9	0.1	1.4	0.1	2.2	0.3
LRS	**U**	0.1	0.0	0.3	0.1	0.6	0.1	0.9	0.3	1.3	0.4
	N	0.1	0.0	0.3	0.1	0.5	0.2	0.8	0.3	1.2	0.4
	E	0.2	0.1	0.7	0.1	1.0	0.1	1.5	0.1	2.3	0.3
PS	**U**	0.0	0.0	0.0	0.0	0.1	0.0	0.1	0.0	0.1	0.0
	N	0.0	0.0	0.0	0.0	0.1	0.0	0.1	0.0	0.1	0.0
	E	0.0	0.0	0.0	0.0	0.1	0.0	0.1	0.0	0.1	0.0
EPS	**U**	0.5	0.0	1.0	0.1	1.6	0.1	2.1	0.1	2.7	0.1
	N	0.5	0.0	1.1	0.1	1.6	0.1	2.1	0.1	2.6	0.2
	E	0.5	0.0	1.1	0.1	1.7	0.1	2.3	0.1	3.0	0.1

Table B.25. Goodness of solutions (%): general-3

	n	50		100		150		200		250	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
LRE	**U**	120.8	14.0	121.9	14.4	117.8	13.2	112.1	8.5	109.0	6.4
	N	119.6	8.9	117.8	7.4	115.7	10.5	112.4	5.5	102.0	0.0
	E	208.5	43.6	144.3	27.1	170.1	41.6	153.3	9.9	175.2	31.4
LRS	**U**	80.9	19.8	70.7	18.7	68.8	18.9	71.3	19.0	88.1	14.0
	N	89.2	9.9	81.0	21.8	82.4	21.3	68.6	28.5	100.0	0.0
	E	68.8	10.8	47.5	12.5	53.8	13.7	50.5	11.9	47.6	9.5
PS	**U**	97.2	3.6	97.3	4.3	95.8	5.4	97.7	3.2	98.2	3.8
	N	92.1	9.2	99.6	1.1	96.2	5.3	99.9	0.2	100.0	0.0
	E	94.7	4.3	97.6	4.2	97.6	4.6	96.6	4.4	99.3	1.4
EPS	**U**	99.3	1.6	99.1	2.1	99.8	0.4	99.5	1.1	99.9	0.1
	N	98.6	2.9	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0
	E	98.8	1.1	99.6	0.6	98.1	4.0	99.6	0.5	99.3	1.4

Table B.26. Running times (s): general-3

	n	50		100		150		200		250	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
OPT	**U**	77.2	174.5	244.5	235.4	164.5	147.4	406.6	315.2	552.8	311.2
	N	145.0	188.0	184.3	128.8	384.5	258.7	381.3	303.5	405.4	0.0
	E	67.0	84.2	109.4	67.1	312.4	321.4	584.1	313.0	513.4	263.5
LRE	**U**	0.5	0.1	1.7	0.5	3.0	0.8	5.6	1.4	7.1	1.3
	N	0.4	0.1	1.3	0.4	3.2	0.9	5.5	1.6	9.9	0.0
	E	0.8	0.2	1.7	0.1	3.7	0.7	6.0	0.4	8.5	0.4
LRS	**U**	0.5	0.1	1.7	0.5	3.1	0.8	5.7	1.4	7.2	1.3
	N	0.4	0.1	1.3	0.4	3.3	0.9	5.6	1.6	10.0	0.0
	E	0.8	0.2	1.8	0.1	3.8	0.7	6.1	0.4	8.6	0.4
PS	**U**	0.0	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0
	N	0.0	0.0	0.0	0.0	0.1	0.0	0.1	0.0	0.1	0.0
	E	0.0	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0
EPS	**U**	0.6	0.1	1.2	0.1	1.8	0.1	2.6	0.2	3.2	0.2
	N	0.5	0.0	1.1	0.1	1.6	0.1	2.3	0.1	2.7	0.0
	E	0.6	0.0	1.3	0.1	2.0	0.1	2.6	0.0	3.5	0.2

APPENDIX C: CONFIGURATIONS OF TESTS

Table C.1. Base configuration for uniformly distributed test files

	Dist. Type	Prm. 1	Prm. 2
Number of resource types	fixed	40	
Set of units of resources	uniform	(1,15)	
Number of bids	fixed	100	200
Number of subbids for bids	uniform	(1,15)	
Size of requested subset of resources	uniform	(1,40%)	
Method for determining elements of s_jk	fixed	0	1
Requested amount of resources	uniform	(1,40%)	
OR factor for ORed requests	fixed	0.98	
AND factor for ANDed requests	fixed	1.03	
Variance for price	fixed	20%	

Table C.2. Base configuration for normally distributed test files

	Dist. Type	Prm. 1	Prm. 2
Number of resource types	fixed	40	
Set of units of resources	normal	(8,2)	
Number of bids	fixed	100	200
Number of subbids for bids	normal	(8,2)	
Size of requested subset of resources	normal	(20%,5%)	
Method for determining elements of s_jk	fixed	0	1
Requested amount of resources	normal	(20%,5%)	
OR factor for ORed requests	fixed	0.98	
AND factor for ANDed requests	fixed	1.03	
Variance for price	fixed	20%	

Table C.3. Base configuration for exponentially distributed test files

	Dist. Type	Prm. 1	Prm. 2
Number of resource types	fixed	40	
Set of units of resources	exponential	8	
Number of bids	fixed	100	200
Number of subbids for bids	exponential	8	
Size of requested subset of resources	exponential	20%	
Method for determining elements of s_jk	fixed	0	1
Requested amount of resources	exponential	20%	
OR factor for ORed requests	fixed	0.98	
AND factor for ANDed requests	fixed	1.03	
Variance for price	fixed	20%	

REFERENCES

1. Foster, I. and C. Kesselman (editors), *The Grid: Blueprint for a Future Computing Infrastructure*, Morgan-Kaufmann, 1999.

2. Foster, I., C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organization", *The International Journal of High Performance Computing Applications*, Vol. 15, No. 3, pp. 200–222, Fall 2001.

3. Krauter, K., R. Buyya, and M. Maheswaran, "A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing", *International Journal of Software: Practice and Experience*, Vol. 32, No. 2, pp. 135–164, February 2002.

4. "The Globus Alliance", `http://www.globus.org`, 2004.

5. Foster, I., C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration", `http://citeseer.ist.psu.edu/foster02physiology.html`, 2002.

6. Wolski, R., J. S. Plank, J. Brevik, and T. Bryan, "Analyzing Market-Based Resource Allocation Strategies for the Computational Grid", *International Journal of High Performance Computing Applications*, Vol. 15, No. 3, pp. 258–281, Fall 2001.

7. Czajkowski, K., I. T. Foster, and C. Kesselman, "Resource Co-Allocation in Computational Grids", *Proceedings of the The Eighth IEEE International Symposium on High Performance Distributed Computing*, 1999.

8. Czajkowski, K., I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A Resource Management Architecture for Metacomputing Systems", *Lecture Notes in Computer Science*, Vol. 1459, pp. 62–82, 1998.

9. Raman, R., *Matchmaking Frameworks For Distributed Resource Management*, Ph.D. thesis, University Of Wisconsin Madison, 2001.

10. Raman, R., M. Livny, and M. Solomon, "Policy Driven Heterogeneous Resource Co-allocation with Gangmatching", *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, IEEE Computer Society, 2003.

11. "ILOG CPLEX", `http://www.ilog.com/products/cplex`, 2004.

12. Krishna, V., *Auction Theory*, Academic Press, 2002.

13. Vickrey, W., "Counterspeculation, Auctions and Competitive Sealed Tenders", *Journal of Finance*, pp. 8–37, 1961.

14. Fujishima, Y., K. Leyton-Brown, and Y. Shoham, "Taming the Computational Complexity of Combinatorial Auctions: Optimal and Approximate Approaches", Thomas, D. (editor), *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99-Vol1)*, pp. 548–553, Morgan Kaufmann Publishers, 1999.

15. Rothkopf, M. H., A. Pekec, and R. M. Harstad, "Computationally Manageable Combinatorial Auctions", Technical Report 95-09, *Center for Discrete Mathematics and Theoretical Computer Science*, 1995.

16. Sandholm, T., "An Algorithm for Optimal Winner Determination in Combinatorial Auctions", Thomas, D. (editor), *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99-Vol1)*, pp. 542–547, Morgan Kaufmann Publishers, 1999.

17. Sandholm, T., S. Suri, A. Gilpin, and D. Levine, "CABOB: A Fast Optimal Algorithm for Combinatorial Auctions", Nebel, B. (editor), *Proceedings of the Seventeenth International Conference on Artificial Intelligence (IJCAI-01)*, pp. 1102–

1108, Morgan Kaufmann Publishers, 2001.

18. de Vries, S. and R. Vohra, "Combinatorial Auctions: A Survey", *INFORMS Journal of Computing*, Vol. 15, No. 3, pp. 200–222, 2003.

19. "Federal Communications Commision", `http://www.fcc.gov`, 2004.

20. Lawrence, A., M. Peter, C. McAfee, and J. McMillan, "Synergies in Wireless Telephony: Evidence from the Broadband PCS Auctions", `http://citeseer.ist.psu.edu/ausubel97synergies.html`, 1997.

21. Özer, A. H., *Multi Unit Combinatorial Auction Problem*, Senior Project, 2002.

22. Sandholm, T. and S. Suri, "Improved Algorithms for Optimal Winner Determination in Combinatorial Auctions and Generalizations", *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pp. 90–97, 2000.

23. Gonen, R. and D. J. Lehmann, "Optimal Solutions for Multi-Unit Combinatorial Auctions: Branch and Bound Heuristics", *ACM Conference on Electronic Commerce*, pp. 13–20, 2000.

24. Gonen, R. and D. Lehmann, "Linear Programming Helps Solving Large Multi-Unit Combinatorial Auctions", `http://citeseer.ist.psu.edu/gonen01linear.html`, 2001.

25. Sandholm, T., S. Suri, A. Gilpin, and D. Levine, "Winner Determination in Combinatorial Auction Generalizations", Gini, M., T. Ishida, C. Castelfranchi, and W. L. Johnson (editors), *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)*, pp. 69–76, ACM Press, 2002.

26. Garey, M. R. and D. S. Johnson, *Computers and Intractability – A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

27. Silberschatz, A. and P. B. Galvin, *Operating Systems Concepts*, Addison-Wesley, 1994.

28. Foster, I., C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy, "A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation", *Proceedings of the International Workshop on Quality of Service*, 1999.

29. Ahuja, R., T. Magnanti, and J. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice-Hall, Englewood Cliffs, 1993.

30. "GSL - GNU Scientific Library", `http://www.gnu.org/software/gsl`, 2004.