PDF Version

Modelling of Electricity Markets Using Software Agents

A dissertation submitted to the University of Manchester Institute of Science and Technology

For the degree of

Master of Science

in Electrical Power Engineering

Georgios Papageorgiou

Department of Electrical Engineering and Electronics

UMIST

2002

Acknowledgments

I would like to thank my supervisor, Dr Daniel Kirschen, for his valuable guidance and constant support.

Abstract

Conventional Economic Modelling (CEM) can be successfully employed to describe different financial equilibria. Its ability, though, to provide insights into the strategic behaviour of generating companies competing in the new, restructured electricity markets is limited. This lead to the development of techniques based on computational modelling and simulation. In this project, a computational model has been developed in order to investigate the bidding behaviour of generating companies in a short-term bilateral market. The generating companies are represented by software agents. Each agent is modelled with specific strategic and operational objectives and bounded reasoning abilities based on principles of reinforcement learning.

A number of simulation studies have been carried out based on the software platform developed in this project. These studies have shown that market prices are likely to increase under Hourly Bidding with Pay Bid settlement. They have also shown that when the market share objective of the generating companies takes over their short-term profit maximisation objective, market prices are likely to decrease. Tacit collusion among generators to increase the market prices has also been observed under peak demand. In a market with a small number of generators, agents with large and flexible portfolios were able to exercise market power. This resulted in extremely high market prices. Conversely, in a market with a large number of symmetric generators, prices have experienced a significant reduction.

Chapter 1: Introduction

Introduction

This chapter serves as an introduction to the models used to study the trading of electricity in the UK. It also presents the objectives of this project and the methodology followed in order to achieve these objectives.

A brief summary of the chapters that follow is also provided.

Description of the Electricity Markets in the UK

In 1990, the Electricity Pool of England and Wales was put in operation in order to provide a competitive wholesale market. This included a mandatory spot market, where the system operator would pay all generators, whose bids were accepted, at the System Marginal Price (SMP). The SMP was defined as the last bid price accepted in order to satisfy the forecasted demand. The "Pool" model suffered from a number of shortcomings such as the limited participation of the demand-side, the low risk to which the generators were exposed, the high wholesale prices and the underdevelopment of secondary markets such as forwards and futures markets.

In November 2000, the New Electricity Trading Arrangements (NETA) replaced the mandatory, daily uniform price auction in the hope of increasing competition and initiating large-scale interaction between the supply and demand sides of the market.

The key element of NETA is summarised in a series of bilateral markets traded ahead of real time. These include the existence of a forwards and futures market, in which most participants

buy or sell to meet the bulk of their load requirement for each of the 48 half-hourly trading periods. Close to real time, a short-term bilateral market in the form of Power Exchanges operates in order to allow fine-tuning of the positions of the market participants. At gate closure i.e.1.0 hour ahead of the real trading period Final Physical Notifications (FPNs) are submitted to the System Operator (SO), together with bids/offers to the Balancing Mechanism (BM). The SO calls those bids and offers that are required to operate the system in terms of both energy and system balancing. None of the above markets is compulsory.

It is evident that NETA motivates the participants to manage their own risk, as they are completely free to engage in any type of contract prior to the actual trading. Furthermore, the supply companies residing on the demand side of the market are actively participating, rather than behaving as passive price takers. The scheduling has stopped being centrally administered and capacity payments are no longer applicable.

One of the most appealing features of NETA, is the philosophy governing the operation of its Spot Market, that is to discourage participants from trading in it. This is achieved by penalising "unbalanced" participants; a topping-up participant is obliged to buy from the Spot Market at a price (System Buying Price) much higher than the one normally achieved through bilateral trading, whereas a spilling participant is forced to sell at a price (System Selling Price) much lower than the one normally achieved through bilateral trading.

The volatility of the Spot Market prices is indicative of the additional risk carried by trading in that market. The following figure illustrates the variation of the System Selling and Buying Price during a typical trading day;

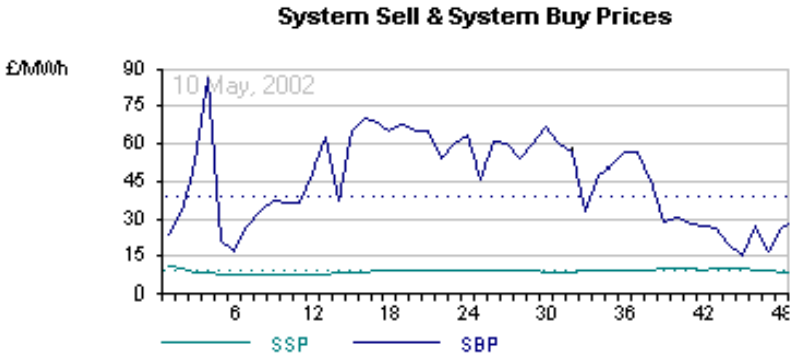


Figure 1- 1 System sell and system buy price variation on 10 May 2002

At present, NETA seems to benefit generators with portfolios that provide both flexibility and the ability to deliver exactly the amount of power contracted (e.g. Pumped Hydro and Nuclear power stations). It follows that under the current arrangements, there is little benefit for power schemes based on renewable energy resources due to the uncertainty associated with these. The existence, though, of forwards and futures markets has reduced wholesale prices that led to the creation of considerable profit margins in favour of generating companies. On the other hand, no significant reduction has been observed in retail prices due to the lack of competition in the retail market.

Scope of the Project

The task of developing adequate models that enable the prediction and understanding of the strategic behaviour of the participants in the restructured electricity markets described above, requires the ability to represent generating companies with a substantial degree of asymmetry and discontinuity. The term "asymmetry" refers to the differences among the portfolios of

generating companies, both in terms of size (capacity) and diversity (plant composition). Similarly, the term "discontinuity" refers to the shape of the supply function bid by each generator, as a result of the different characteristics of the plants that make up its portfolio.

Conventional economic modelling, based on equilibrium solutions, assumes the representation of generators by continuous supply functions and the supply side of the market by symmetrically equally sized firms. Furthermore, CEM "gives some comparative insights on market design, but provides an inadequate representation of the observed, dynamic evolution of these typically quite imperfect markets" [1][1].

Taking the above into consideration, new methods of predicting and analysing the strategic behaviour of market participants are currently being developed, providing an alternative to the limitations imposed by conventional economic modelling. In this project, a method to investigate the bidding behaviour of generating companies has been developed based on computational modelling and simulation. This method employs the concept of software agents to represent the various generating companies. Each agent is modelled with specific strategic and operational objectives and bounded reasoning abilities based on principles of reinforcement learning.

Objectives

The principle aim of this project has been the design and implementation of a software platform that would be used to investigate the bidding strategies developed by generating companies in a short-term bilateral market. In particular, this simulation platform required:

[1] See Chapter 2, Section 3.1 for a thorough analysis of the inadequacies of Conventional Economic Modelling (CEM).

- The design and implementation of a market module (trading environment), which would be capable of calling, accepting and processing bids (quantity and price) submitted by the generating companies for each of their plants. This module should be capable of operating based on two modes of bidding (Hourly and Daily). The same model should be capable of providing feedback to the agents with the accepted bids (offers), at the end of each trading day (iteration).

- The design and implementation of a set of agents representing the supply side of the market i.e. generating companies having plant portfolios with clearly defined characteristics (capacity, marginal costs, availability). A generating agent should be capable of receiving the accepted offers from the market mechanism and based on those, formulate its "next-day" bidding strategy. The bidding strategies should aim towards the fulfilment of the daily profit and market share maximisation objectives defined for each agent.

- The representation of the demand side of the market with a typical day load forecast. This would be kept constant during the whole iteration cycle in order to isolate characteristics linked with the bidding behaviour of agents. Such an action is perfectly justifiable, taking into consideration the low price elasticity exhibited by the demand side of the electricity market.

- To determine the impact of various bidding strategies on the following:

a. The System Price(s)

b. The profits of the participants

c. The opportunity to exercise market power or the development of tacit collusion among participants.

Methodology

The design of the simulation platform was based on work previously carried out by Bower and Bunn [11] and other pioneers in the field of Electricity Markets. For this reason, a broad literature survey has been conducted in order to gain understanding of the main concepts involved.

It should be emphasised, that this project required the collection of information from various backgrounds and disciplines. These included Artificial Intelligence (AI) and in particular the field of "machine learning", Computational Economics with emphasis on Electricity Market Modelling and concepts from the Auction Theory. The reader may refer to the Bibliography section where material related to these subjects is referenced.

The code developed for this software platform was entirely written in the "C" Programming Language. The reasons behind such a decision are explained in Chapter 3.

Four case studies were carried out using the software platform presented in this project. The results of these case studies are analysed and conclusions are drawn where appropriate.

Project Management

This project has been completed according to the time and plan schedule set during the planning phase. The reader will find this schedule in Appendices 14 (Gantt Chart) and 15 (Activity-On-Node Diagram).

Project Report Organisation

The main body of this report is comprised of four chapters. A brief outline of the individual objectives and contents of each of these chapters is provided below:

Chapter 2: Literature Survey on Agent Based Simulations

This chapter provides an introduction to the principles related to software agents and agent-based modelling. This is achieved through the classification of agents and the definition of agent software systems. An introduction to reinforcement learning-the preferred method of machine learning- is also provided.

Two important case studies are described; both studies employed agent-based simulations in order to investigate a range of issues related to electricity markets. Finally, a section is dedicated to the explanation of the inadequacies of conventional economic modelling to predict and analyse restructured electricity markets.

Chapter 3: Design, Methodology and Software Development

This chapter is used to outline the methodology followed in order to implement the proposed simulation platform. The initial design specifications are stated and a thorough analysis of each of the modules comprising the simulation platform is provided. Furthermore, a detailed analysis of the more specialised issues related to the design of the software is provided. The reader can also find a brief description of the main software functions. Finally, a reference to the testing procedures is made.

Chapter 4: Simulation Results

This chapter demonstrates the functionality of the simulation platform and at the same time

examines the bidding behaviour of agents in an electricity market. This is achieved through the analysis of the results produced by the simulation of four case studies.

The first case study examines the effect of Daily and Hourly Bidding on market prices and profits.

The second case study examines the impact of increasing the target utilisation rate of mid-merit generators on market prices and profits.

The third case study examines the issue of market power and "tacit collusion" exercised by the participating generators during trading periods of high demand. This is achieved by increasing the demand to 95% of the total available capacity during peak trading hours.

Finally, the fourth case study examines the impact of varying the number of the participating generators while maintaining the total available capacity constant on market prices.

Chapter 5: Conclusions

This chapter gives an account of what has been achieved in this project and provides a summary of the results obtained by simulation. The various limitations of the simulation platform are discussed and possible improvements suggested. One section is also dedicated to the future development of the simulation platform.

Summary

This chapter introduced the models used to study the trading of electricity in the UK. It also presented the objectives of this project and the methodology followed in order to achieve these objectives. A brief summary of the chapters that follow has also been provided.

Chapter 2: Literature Survey on Agent Based Simulations

Introduction

This chapter serves as an introduction to the concept of Agent-Based Modelling (ABM) as applied in financial market modelling. It also attempts to highlight the usefulness of this tool, as opposed to Conventional Economic Modelling (CEM), by establishing a basis for the understanding of Artificial Intelligence (AI) Multi-Agent Software Systems (MASS). This is required in order to realise how this new approach can perform a useful role in the face of modern challenges, which include the ability to understand, predict and evaluate the bidding behaviour of participants in new, restructured markets.

The second part of this chapter includes an epigrammatic description of the electricity market and trading conditions in the UK from 1990 up-to-date; this allows a review of previous studies that successfully employed the concepts of ABM and MASS to investigate various issues related to the nature of these markets.

Agent Based Simulations

Agent Terminology and Elements of Artificial Intelligence (AI)

"A MASS is a computer program with problem solvers situated in interactive environments, which are each capable of flexible, autonomous, yet socially organised actions that can, but need not be, directed towards predetermined objectives or goals.

Thus, the four criteria for an intelligent agent system include software problem solvers that are *situated*, *autonomous*, *flexible* and *social*." [2]

Situated intelligent agents are those agents, which are characterised by the capability of receiving stimuli (inputs) from their environment. They must also be in a position to affect changes in their environment. Examples of environments include the internet, a computer game or, in this instance, a financial market.

An *autonomous* agent is one that can interact with its environment without the direct intervention of other agents. This requires the agent to be in control of its own actions and internal state. Autonomous agents can also learn from past experience to improve their performance over time.

A *flexible* agent is characterised by a responsive and proactive nature. The responsiveness of the agent is due to the fact that it can receive stimuli from its external environment and respond to these stimuli in an appropriate and timely fashion. A proactive agent is in a position to exhibit opportunistic behaviour, be goal-directed and have a number of alternative actions to various situations.

Finally, an agent can be *social* i.e. able to interact with other software or human agents. The interactions of a social agent should aim towards the achievement of the goals set by the larger multi-agent system. A multi-agent system can be seen as a loosely coupled network of problem solvers that work together on problems that may be beyond the scope of any of the individual agents [3].

Machine Learning (Reinforcement Learning)

Machine learning, in the context of AI, is a multithreaded term. It can be viewed from symbol-based to connectionist and genetic or evolutionary perspectives, depending on the nature of the problem to be solved and the algorithm written to carry out such a task. The plethora of machine-

learning techniques has not prevented, though, the establishment of a general definition such as that given by Herbert Simon [4];

"…Learning is any change in a system that allows to perform better the second time on repetition of the same task drawn from the same population…".

In general, any agent who claims to be intelligent must possess some learning capabilities. This also implies that an agent evolves i.e. changes as the learning process takes place. Learning is important since it enables systems to initiate a process with a minimal amount of knowledge.

At this point the emphasis is shifted to reinforcement learning, which has been chosen as the primary method of learning for the agent system designed to serve the purpose of this project. Reinforcement Learning (RL) differs from any other learning process in the sense that the agent itself (i.e. in the absence of supervision) must learn an optimal policy through trial, error and feedback from the environment in which it is sited. The feedback from the environment is a direct consequence of the agent actions. On the other hand, trial and error requires the agent to continually explore and exploit its environment; this is in fact an essential ingredient of the RL process.

Sutton and Barto [5] define the agent to be the learner or the decision maker. Everything outside the agent is defined as the environment with which the agent is in continuous interaction. This interaction allows the agent to select actions and the environment to respond to those actions. The environment also gives rise to rewards i.e. special numerical values that the agent tries to maximise over time. A complete specification of an environment defines a task i.e. one instance of the reinforcement learning problem. At each time, the agent receives some representation of

the environment's state that forms the basis on which the agent selects an action from a set of possible actions. One step later – as a consequence of the action previously taken – the agent receives a reward and finds itself in a new state. The following figure illustrates the agent-environment interaction in reinforcement learning;

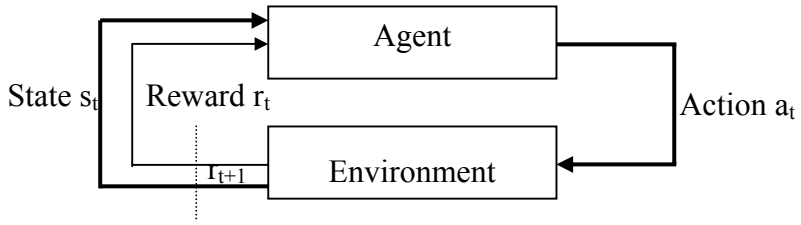


Figure 2- 1 The agent-environment interaction in reinforcement learning, [5] p. 52

George F Lugger in [2], proposes the following terminology for RL;

t is a discrete time step in the problem solving process

s_t is the problem state at t, dependent on s_{t-1} and a_{t-1} where

a_t is the action at t, dependent on s_t

r_t is the reward at t, dependent on s_{t-1} and a_{t-1}

π is a policy for taking an action in a state i.e. π is a mapping from states to actions.

π* is the optimal policy

V maps a state to its value, hence $V^{\pi}(s)$ is the value of state s under policy π.

The following figure summarises the key components of reinforcement learning;

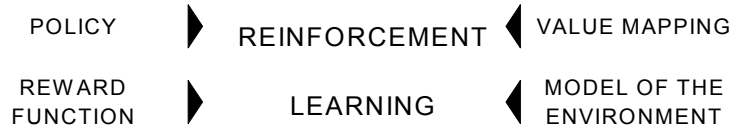


Figure 2- 2 Main components of reinforcement learning (RL)

Policy $\boldsymbol{\pi}$ is defined as the mapping the agent implements from states to probabilities of selecting each possible action at each time step. Usually, it can be represented by a set of production rules. This is the most important component of RL, since it is sufficient to generate behaviour even when the rest of the RL components are not present.

The Reward Function $\mathbf{r_t}$, defines the relationship between the state and the goal at time t. It maps each state-response pair into a reward measure that it is used to indicate how desirable is the action that generated the response with regards to achieving the goal.

The value function **V**, is a property of each state of the environment indicating the reward the system can expect for actions continuing on from that state i.e. it serves as an indication of the long-term desirability of a state.

There are three different families of RL inference algorithms according to the method employed; temporal difference learning, dynamic programming and Monte Carlo algorithms [2]. There is a number of advantages and disadvantages related to each of the aforementioned algorithms, such as the assumption made by classical dynamic programming that a perfect model is available; this is, though, offset by its ultra-fast policy search capability.

This dissertation is focused on the use of Monte Carlo[2] methods, since they do not require a complete model for the environment. On the other hand, they do require experience from on-line or simulated interactions with the environment. The model is required to be regenerative rather than analytical, in other words it should be capable of generating trajectories of states but it is not

[2]The term "Monte Carlo" is often used for any estimation method whose operation involves a significant random component.

required to calculate explicit probabilities for each state; a feature required by dynamic programming.

Reasons for using Agent Based Simulations – Case Studies

The Inadequacies of Conventional Economic Modelling (CEM)

Almost any financial market is characterised by a vast amount of information in the form of price signals. The market participants are constantly struggling to understand and interpret the complex relationship governing market prices and the signals emerging from these, hence new ways to simulate the market processes and trading transactions, in order to predict future behaviour, are under investigation.

As mentioned in the introduction of this chapter, "CEM can be successfully employed to describe different financial equilibria, but it fails to interpret the diverse nature of the dynamics occurring while learning is still active and equilibrium is never quite obtained" [6]. The governing dynamics of new, restructured financial markets are often much more complex than those involved in the simple supply-demand goods market.

The Electricity Market, although unique due to the nature of the commodity being traded, still possesses all those features that could not possibly justify its exception to the above observations. Hence, "CEM approaches have shown a limited ability to develop insights into the strategic behaviour of firms competing in new, restructured markets, such as electricity" [7]. CEM tends to oversimplify, over-aggregate and represent stylised versions of the Electricity Markets since these are characterised by an oligopoly of generators, little demand-side elasticity in the short term and complex, administered market mechanisms which are designed to facilitate both financial trading and physical, real-time system balancing.

Traditional economic modelling is deemed inadequate when it comes to investigating the behaviour of participants in electricity markets, since the analytical evaluation of the supply function equilibria involved, requires a number of assumptions to be made. Namely, that the generating units (supply side) be represented by a continuous supply function [8], or that the industry ownership be composed of symmetrically equally sized firms [9], or both [10]. Obviously, the above assumptions do not correspond to reality, owing to the fact that generating plants are made up of a number of individual units; hence their resultant supply function is an aggregate of the supply functions down to the unit level. This is shown in figure 2.2.

Furthermore, generating companies have plant portfolios, which vary in net power output and plant technologies.

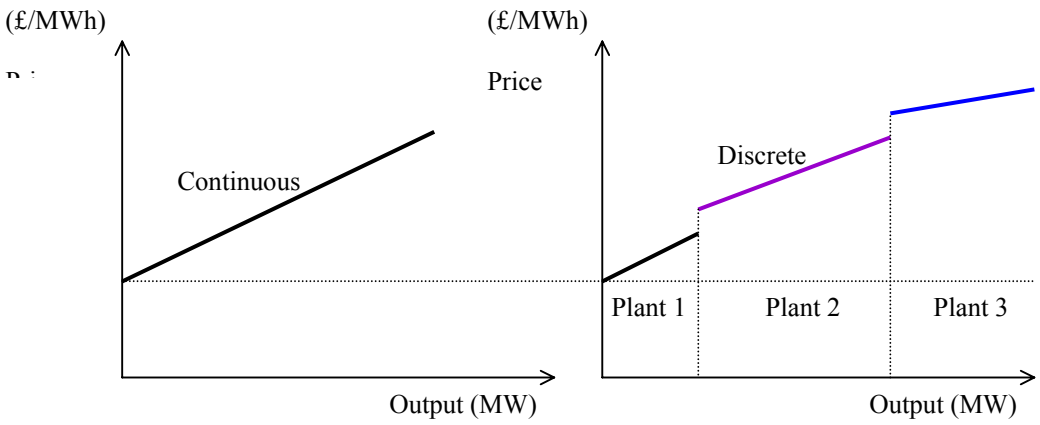


Figure 2- 3 Assumption regarding the supply function in CEM

The use of learning agents to model the electricity market trading environment, seems to surmount the aforementioned problems, however at the cost of building a fairly complex model to represent the agents and their environment. Nevertheless, financial markets are well suited to

agents in the sense that a certain level of simplicity can be assumed when agent objectives are defined.

Tesfatsion [12] refers to this relatively new branch of economics as agent-based computational economics (ACE). This is defined as the "computational study of economies modelled as evolving systems of autonomous interacting agents". There are two major drives for scientists in this field; one is descriptive i.e. it focuses on the explanation of the behaviour of emergent global economies and is concerned with answering questions related to the evolution and persistence of particular global regularities in real-world decentralised market economies, despite the top-down planning and control in the latter. The other drive is normative and focuses on mechanism design; it is concerned with answering questions related to the impact of particular market protocols or government regulations on economic efficiency. Finally, Tesfatsion [12] emphasises the common characteristics between ACE and Artificial Life researchers i.e. "to demonstrate constructively how global regularities might arise from the bottom up, through the repeated local interactions of autonomous agents. Both sets of researchers use computational models as descriptive tools for understanding existing phenomena and as normative tools for the design and testing of alternative possibilities; and both sets of researchers share a desire to develop coherent theories that are comprehensive in scope rather than fractured along outmoded disciplinary boundary lines".

Case Studies

The case study presented by Bower and Bunn in [11], acted as a guide for this project; a significant proportion of the concepts regarding agent based modelling presented in this study were directly adopted whilst others were modified to meet the specific requirements of the project.

This study was carried out at a time when the Review of Electricity Trading Arrangements (RETA), initiated by the Office of Electricity Regulation (now OFGEM) and the Department of Trade and Industry (DTI), sought confirmation in an attempt to replace the existing "Pool" model (see section 1.2). This confirmation emerged from computer simulation models, designed to test the potential impact of alternative trading arrangements on market prices.

Generating firms were represented as autonomous agents that used simple internal decision rules. These rules formed the basis that allowed them to discover and learn strategic solutions that satisfied their profit and market share objectives over time. In other words, these rules constituted a naïve reinforcement learning algorithm that sought to exploit successful bidding strategies and discard unsuccessful bidding strategies. In essence, each of the agents represented one of the generating firms that participated in the Pool during 1998 and was endowed with a portfolio of plants characterised by a number of parameters including capacity, fuel type, efficiency and availability.

The demand side of the market was represented by passive price takers with no ability to influence the market through strategic behaviour. Hence, a standardised daily load profile was created that corresponded to the demand patterns occurring during a typical winter day.

The Pool's day-ahead market and the bilateral short-term market were both modelled as daily repeated auction mechanisms; conclusions were drawn by comparing the market clearing prices obtained under four different combinations of trading and settlement. These are described in the following table;

Table 2- 1 Auction models tested in Case Study 1

Type of Market	Bidding Time Interval	Settlement Method
Pool day-ahead	Daily	Uniform Price (Pay SMP)
Pool day-ahead	Daily	Discriminatory (Pay Bid)
Bilateral short-term	Hourly	Uniform Price (Pay SMP)
Bilateral short-term	Hourly	Discriminatory (Pay Bid)

The clearing mechanism is fairly straightforward; an aggregate supply function is built by stacking plant bids, starting from the lowest to the highest, and by allocating demand to plants in strict merit order until the demand is exhausted for each of the 24 hourly periods. At the end of the day, the revenue for each plant is calculated on the basis of demand allocated multiplied by the reward price i.e. SMP or Price-Bid, depending on the market type. In the case of daily bidding, only one bid is allowed per plant per day.

Each agent had two strategic objectives, namely:

a. To increase its overall profitability and

b. To reach a target utilisation rate in every period

In order to achieve their strategic objectives, agents had to follow either a "price raising" or a "price lowering" strategy by adding/subtracting a random percentage to/from the prices bid in the previous day. Bid prices could take any value between zero and £1000.00 but plants with higher marginal cost always had to bid higher prices than plants with lower marginal cost in the same portfolio. If an agent failed to reach its target utilisation rate on the previous trading day, it lowered the bid prices on all of its plants for the next trading day. A successful bidding strategy could be transferred across the portfolio, by allowing the agent to raise the bid price of its plant

which bid the lowest price during the previous trading day, to the next highest bid submitted by a plant of the same category.

Agents could not communicate and ignored the state of the market. This is a fairly realistic assumption, since the information available to the participants in the Pool or the bilateral short-term market regarding prices bid by other participants, is quite limited. This assumption also permitted the impact of alternative bidding and settlement arrangements to be examined closely.

Analysis carried out on the simulation results revealed that the Pool day-ahead market with Pay SMP settlement and single daily bids produced the lowest prices while the bilateral model with Pay Bid settlement and hourly bids produced the highest prices. The simulations also verified the fact that base-load plants bid close to zero for Pay SMP; this was found to be the case for both bilateral and Pool market types. In contrast, agents learned quickly to bid a much flatter supply function, well above zero, for Pay Bid settlement.

The authors of this case study had also been able to verify that bid prices fell with increasing utilisation rate, in the case of both the Pool and bilateral markets. The agents followed a price-diminishing behaviour, as this was the only way to increase their market share.

The findings clearly suggested that OFFER's proposed bilateral model contradicted its expectations mainly for the following reasons;

i. The Pay Bid settlement was found to increase the risk for baseload generators, particularly IPPs with small plant portfolios. This was due to the fact that baseload generators had to raise the level of their bid prices closer to the market clearing price in order to

maximise their profits. This reduced competitive pressure on generators with mid-merit plant.

ii. Hourly bidding allowed generators to effectively segment demand into peak and off-peak hours, consequently extracting a greater proportion of the consumer surplus than under daily bidding.

Forcing baseload generators to participate in price setting (rather than adopting a Pool market, bid-close-to-zero, behaviour) was found to reduce competitive pressure on mid-merit generators, which allowed prices to rise. The results clearly showed that competition was reduced, rather than increased, through Pay Bid settlement. In addition, a rise in bidding prices was found to occur for hourly bidding, regardless of the market type. This was verified to be, partially, the result of agent "tacit collusion" and improved optimisation skills. Finally, bilateral market conditions were found to increase generator market power, especially during peak demand hours. This resulted in an increase in the risk associated with the occurrence of very large price variations, which is especially true during plant outages or under low plant margin conditions.

The case study by Oliveira and Bunn in [7], deals with the implementation of a large-scale application of multi-agent evolutionary modelling to the New Electricity Trading Arrangements (NETA). This agent-based simulation was implemented in order to gain pricing and strategic insights ahead of NETA's actual introduction.

In contrast to the model described in the previous case study, this model was designed to capture the interaction between bilateral trading and balancing market. The proposed simulation platform incorporated an active demand-side representation. It also succeeded to take into consideration daily dynamic constraints by adopting the concept of plant cycles.

A number of simplifications were made in order to model the balancing mechanism; the transmission system was neglected, therefore transmission constraints and regional imbalances were not modelled. In addition, only a typical day demand profile was simulated, since the model aimed at highlighting the process of finding the equilibrium solution for that specific demand profile. Furthermore, the forward and balancing markets were considered as two sequential markets despite the continuous nature of trading in NETA. Finally, the model assumed that no vertical integration existed in the industry.

The modelling of markets was achieved by adapting the Single Call Market (SCM) to the trading principles of NETA; the agents paid the price bid or received the price offered instead of paying (receiving) the clearing price.

Suppliers and generators were represented by agents with bounded rationality but with no communication capabilities. Bunn and Oliveira describe them as "conceptual identities that represent 'economic agents' in the market with capacity to receive information, learn from the interactions and act on the simulated environment".

These agents had a number of operational and strategic objectives. The operational objectives ensured that no crossing of information took place between the short-term bilateral and balancing markets. On the other hand, strategic objectives defined the purpose of the actions taken by each agent. These included maximisation of the daily profits and minimisation of the exposure in the balancing mechanism, both for suppliers and generators. Interrelation between operational and strategic objectives was preserved through the existence of operational rules.

The learning algorithm adopted is that of reinforcement learning (see section 2.2.2). Agents learned "on-line" meaning that "given the information set, an agent modified its actions in order to maximise its own profit".

The agents used a number of instruments, typically ratios between prices bid in the short-term power exchange (PX) and the balancing mechanism (BM). These included that ratio between the price bid (offered) in the PX during the current and previous days and the ratio between the price bid (offered) in the BM and the PX during the same day. According to the authors, a strong feature of this learning algorithm is its simplicity; agents only learned a different policy for each of the four mark ups derived from the ratios mentioned above. Another strong feature of this algorithm is that it assumes that the information set may contain the prices of only one trading day (the previous day). This assumption is justified by the Markov property [5]. This property characterises any state signal that succeeds in retaining any relevant information from the past. A state signal having this property is also known as the Markov signal. It follows that the agents tried to improve their position by assuming that the present contained all the relevant information acquired by past experience. In other words, the agents assumed that all "present" states where Markov states.

Agents did not learn how to choose prices but they learned how to choose mark ups based on the previous day's prices. At every state, the agents calculated their expected daily profit and expected acceptance rate for each one of the mark ups used at the specific iteration. The expected daily profit and utilisation rate were used in order to calculate the expected *reward* for each of the mark ups. The mark ups were ranked starting from that with the lowest reward to the one

with the highest reward. Then a utility value was assigned to each of the mark ups; the highest value was assigned to the mark up with the highest reward. The shape of this utility function was also determined by the value of the *search propensity* parameter. This parameter is, in essence, a heuristic control of the way agents transform past experience into future policies. Depending on the integer value assigned to this parameter, the agent is set to either implement stable or reactive policies. Finally, the agent transformed its utility function into a policy; each policy associated each mark up with its probability of being bid (offered) during the next day.

The algorithm employed by Bunn and Oliveira [7], has the following distinct characteristics;

1. The model is a pure strategy stochastic game
2. The agents define how much exploration and exploitation to undertake through both the utility function described above and a learning parameter, which is set at a suitable value in order to model a tracking problem.
3. The players have the same probabilities of exploitation or exploration, regardless the number of iteration of the game. This achieves to create a model that avoids converging at local equilibria, without prior search of the entire solution space.

The analysis of the results obtained showed that suppliers are the ones likely to carry most of the risk under NETA, due to the prediction errors associated firstly, to quantities traded in the Balancing Market and secondly, to imbalances. Prediction errors were found to be in the order of 5 to 10% with respect to the total trade in the power exchanges. The lower the prediction error, the higher the Profit-per-unit-sold of the industry as a whole, given an objective of 100% contract cover. For a contract cover equal to 115%, it was found that the Profit-per-unit-sold declined; the opposite effect took place when the contract cover was reduced to 85%. This was not result anticipated by the supply industry, and the explanation given was that in the case

that all suppliers under-contracted, the power exchange prices fell, given the reduction in demand. Following that, the generators had extra capacity available to sell thus the BM price decreased.

It had been proven that the average daily prices laid somewhere between the System Buy and System Sell prices. This is exactly what NETA was designed to achieve i.e. to create a Balancing Market characterised by an extremely volatile System Buying Price that would lie well above the average daily prices (see fig. 2.2). The above imply that trading in the Balancing Market is associated with excessive risk and, therefore, market participants are well motivated to develop more accurate load forecasting techniques that would enable most of their trading to take place in the forwards market.

As far as generators are concerned, the study has proven that technologically flexible plants tend to achieve above average profit levels. It was shown, though, that the value of a flexible plant is greatly dependent on the synthesis of the overall portfolio owned by the same generating company. Finally, generator collusive behaviour was found to be greatly related to the value of the Capacity Margin, that is the demand expressed as a percentage of the total capacity available.

Summary

This chapter has introduced the basic principles of agents and agent-based modelling. This was achieved through the classification of agents and the definition of agent software systems. Reinforcement learning was chosen as the preferred method of machine learning since its basic principles –exploration and exploitation- are compatible with the repetitive nature of the bidding

process taking place in Electricity Markets.

Two important case studies that modelled electricity markets using agents were presented. The first case study was carried out in an attempt to test the potential impact of alternative trading arrangements on market prices while the second was intended to provide an insight into the complex NETA mechanism prior to its actual introduction. Both of the studies used software agents to model electricity markets as an antidote to the inadequacies of Conventional Economic Modelling.

Chapter 3: Design, Methodology and Software Development

Introduction

This chapter presents with the model of the trading environment proposed, in order to investigate the bidding behaviour of generating companies in a simplified electricity market. The methodology followed is described and the reasons behind milestone decisions are justified. The analysis is two-fold; the overall design is firstly explained without particular reference made to the programming environment or methodology adopted. The last section of this chapter presents information related to the programming methodology followed in order to implement major functions related to the software platform that has been developed; it is emphasised that no particular references are made to the specific programming language used, thus achieving a generic framework.

Design of the Proposed Simulation Platform

The systems engineering approach has been adopted in the design of the simulation platform that has been developed in this project in order to investigate the bidding behaviour of generating companies in electricity markets. This is a structured, top-down method that is traditionally applied to the design of complex systems. Its foremost feature is that it permits details to be added to the initial design at successive stages. At each development stage, the full design cycle is repeated thus ensuring the eradication of design errors at the conceptual level and the abidance to initial design

specifications.

Initial Design Specifications

The initial design specifications –agreed between the client and the system designer- outline the desirable characteristics of the individual components comprising the simulation platform. These characteristics are chosen carefully in order to achieve the specific aims of the project. In this section, some of the design specifications concerned with the representation of the market environment and participants are described. It is emphasised that a number of additional specifications have also been agreed e.g. regarding user-feedback and data presentation.

One of the specifications requires the design of a module capable of simulating the market clearing process (trading mechanism). This should be capable of calling, accepting and processing the bids (quantity and price) submitted by the various generating companies as well as providing feedback to the market participants in the form of accepted offers. The user must be able to choose the method of payment (Uniform or Discriminatory) as well as the bidding time interval (Daily or Hourly).

In addition to the above, a software agent must be designed to represent the supply side of the market i.e. the generating companies. Each of the companies will be described in terms of their plant portfolio; each plant must have clearly defined characteristics such as marginal and operational costs, capacity and availability. A generating agent should be capable of receiving the accepted offers (quantity and price) from the market and form its "next-day" bidding strategy based on profit maximisation and target plant utilisation objectives.

Finally, a module should be designed to represent the economic environment i.e. the supply side of the market. It has been agreed that the demand be represented by a typical day forecast.

The following figure summarises the main modules of the proposed simulation platform;

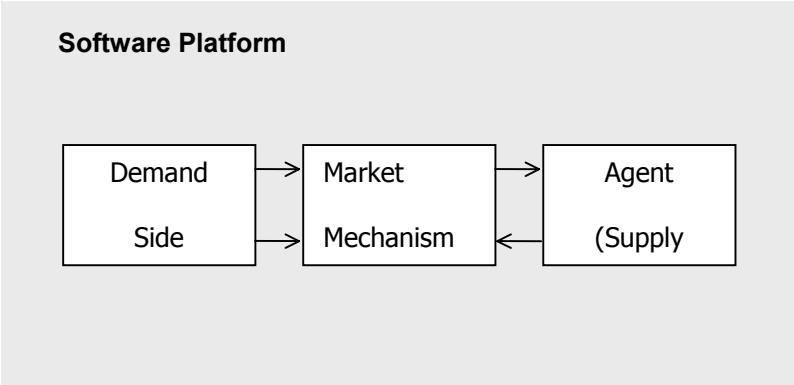


Figure 3- 1 Basic modules of the proposed software platform

Design Analysis

Market Mechanism

In Chapter 1, the three existing markets under NETA have been described. These include the long-term forwards market where generators and suppliers are free to engage in bilateral contracts for the bulk of their energy requirements. In addition, short-term power-exchange markets exist for the fine-tuning of the participants' position prior to gate closure and finally the balancing mechanism, which operates as an extremely price-volatile spot market. The sheer complexity of the trading environment under NETA required a number of assumptions to be made in order to build a simple model to represent the market mechanism;

1. The long-term forward and short– term power exchange markets were

considered to take place simultaneously. In fact, initial positions could be assumed for each of the generators thus eliminating the need to model the forwards market.

2. The continuous nature of trading in NETA was simplified in a one-shot transaction thus consequently simplifying the flow of information.
3. Since the transmission system was not modelled, no regional imbalances or transmission constraints were taken into consideration.

Bidding is allowed in either of two modes i.e. daily or hourly bidding. This implies that in the case of daily bidding, generators are required to submit only one bid per plant for the whole day. On the other hand, under hourly bidding, generators are required to submit 24 bids per plant per day. The hourly bids can be submitted just before the trading period is initiated whereas daily bids are submitted at the start of the day. It is emphasised that the market takes place in 24 hourly periods instead of 48 half-hourly periods (NETA).

Once the bids are submitted to the market mechanism, they are sorted starting with the bid with the lowest price and progressing to bids with higher prices. The balancing mechanism will accept bids until the forecasted demand for the particular trading period is fully satisfied. Successful bids will return the price bid in contrast to the obsolete England and Wales Pool arrangement where accepted bids returned the System Marginal Price. The clearing process is carried out for each one of the 24 hourly settlement periods in each trading day.

Following market clearance, the market mechanism is responsible for providing feedback to the generating agents with data such as the number of plants accepted and

the offer (price and quantity) to each plant. All agents receive simultaneously the results of their bids at the end of the trading day and even where separate hourly bids are submitted, there is no opportunity to observe the outcome of these until trading is completed.

The following diagram describes the various inputs\outputs of the market mechanism;

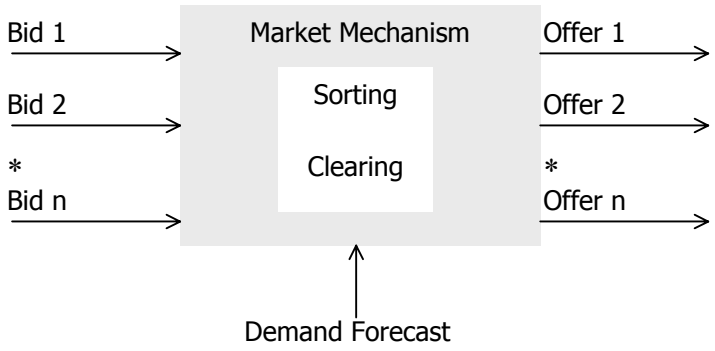


Figure 3- 2 Input data and feedback from the market mechanism

Agent Module

Agents are used to represent generating companies that participate in the Electricity Market. An analogy to a generating agent is a microprocessor. A microprocessor has Read-Only-Memory in which a set of instructions is stored in order to assist the microprocessor in performing its functions. Similarly, an agent has a Read-Only-Memory where all the characteristics of the plants in its portfolio are stored. Access to this memory is restricted to the user prior to run-time and to the external environment (i.e. the market mechanism) while the simulation is in progress.

A microprocessor has a Random-Access-Memory to store parameter values entered by the user via the keyboard whereas an agent has a temporary memory where it

stores data fed back from its external environment. The external environment can access this memory in order to store new data or amend existing data. The agent can access this memory to read, modify and store data.

Furthermore, a microprocessor has an arithmetic unit which enables it to perform mathematical calculations. Similarly, some of the agent functions are restricted to the evaluation of parameters such as daily profit and utilisation rate, based on information fed back by its external environment. Finally, a microprocessor has a logic unit which is used to take a number of decisions based on Boolean Algebra. Similarly, the agent makes a number of comparisons based on the values of the parameters that it has previously calculated in order to resolve a number of statements into true and false and thus form its next day bidding policy based on these outcomes.

The following diagram is used to illustrate the concept of a generating agent as well as the flow of information related to it;

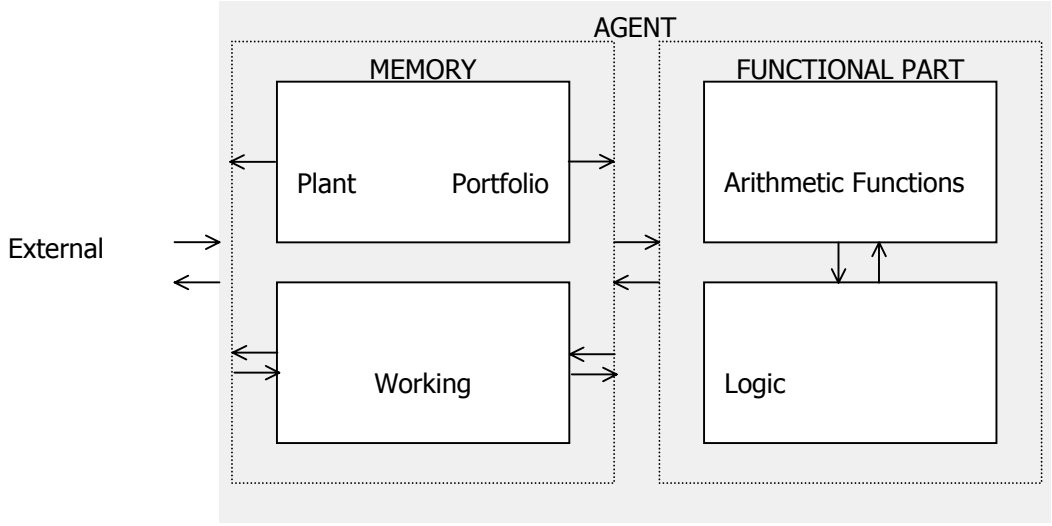


Figure 3- 3 The concept of a generating agent and related information flow

The plant portfolios of the four agents that participated in the simulations carried out in this project are described in Appendix A[3]. Each agent is distinguished by a unique number, which is known as the agent id; similarly, each plant is also distinguished by a unique number, which is known as the plant id. Hence, for example, the second plant of the first agent in the table is identified by the two-digit number "01" (0 stands for first agent and 1 stands for second plant; please note that counting starts from zero).

Each plant is described in terms of its mean available capacity C (MW) and its marginal cost MC (£/MWh) at maximum power output. In addition to that, the operating cost of a plant is represented by a quadratic function whose coefficients are given. Finally, the parameter "plant cycle" is defined in order to distinguish between various plant technologies, hence base-load plants have a plant cycle value equal to one (cycle 1), mid-merit plants have a plant cycle value equal to two (cycle 2) and finally, peak-load plants have a plant cycle value equal to three (cycle 3).

The following table lists the various plant technologies according to their plant cycle;

Table 3- 1 Plant technologies and corresponding plant cycle

Plant Cycle	Plant Type
1	Nuclear Combined Cycle Gas Turbine (CCGT)
2	Small Coal Medium Coal Large Coal
3	Open Cycle Gas Turbine (OCGT) Oil Pumped Storage

[3] Plant data taken from Appendix A in [10]; cost coefficients and plant cycles from our own estimates.

Each agent is assigned a target utilisation rate. The utilisation rate is defined as the percentage of total accepted capacity over total available capacity. This is shown by the following equation:

$$UR_j = \frac{\sum_{i=0}^{n_p} Q^i_{accepted}}{\sum_{i=0}^{n_p} C_i} \quad \ldots (1)$$

where UR_j is the utilisation rate for agent [j]

$Q^i_{accepted}$ is the accepted quantity for plant [j][i] at trading period t (MW)

C_i is the capacity of plant [j][i] (MW)

n_p+1 is the total number of plants in the portfolio of agent [j]

The utilisation rate, at each trading period t, is an indication of the agent's market share. The value of the target utilisation rate depends on the nature of the plant portfolio of each agent, hence an agent with only base-load plants, such as nuclear power plants, would always aim to achieve 100% or 1.0 utilisation rate whereas an agent with a mixture of plant technologies in its portfolio would typically aim to achieve an utilisation rate of less than 1.0. In this project, an initial utilisation rate may be assumed for each agent in order to take into consideration the effect of long-term contracts achieved in the forwards market.

Agents are governed by a number of operational rules, which help them achieve their strategic objectives. Their strategic objectives include the achievement of their target utilisation rate (market share objective) and the maximisation of their daily profits. These operational, internal-decision rules represent the naïve reinforcement

learning algorithm which is used by the agents in order to discover and learn strategic solutions that satisfy their objectives.

The agents receive feedback data from their external environment (market mechanism), which is related to the previous trading day. This includes the accepted bid price and quantity for each plant. On the other hand, agents ignore completely the past, current or future actions of other agents as well as the state of the market. This "eliminates the potential informational difference between generators as well as it ensures that the simulation results are purely due to the various bidding strategies followed by the agents" [11]. The accepted bid prices fed back to the agents, operate as signals (states) that, according to the Markov property (Chapter 2, Section 2.4.2), retain all relevant information from past transactions.

Agents submit the bid prices for each plant in their portfolio, at the beginning of the current trading day (Daily Bidding) or transaction period (Hourly Bidding) using a number of decision criteria. These decision criteria are followed in a certain order of precedence. If the utilisation rate achieved during the previous day is less than the agent target utilisation rate, then the agent subtracts a random percentage (0 to 10% in steps of 1%) from the previous day's bid price for each plant in its portfolio. This price-lowering policy aims at increasing the market share of the agent. Although the action of lowering all bid prices implies that previous successive bidding strategies are disregarded, the agent is still free to explore a number of bidding strategies since the utilisation rate is a global i.e. not plant-specific objective.

If the agent succeeds to meet its target utilisation rate, it raises the bidding price of the plant with the lowest achieved profit across its portfolio during trading period

t, to the bid price of the plant that has achieved the next highest profit during the same trading period. This applies for plants of the same plant cycle, as plants with higher marginal cost should always bid prices that are higher than those bid by plants with lower marginal cost.

The agent compares the profit made during the previous trading day (D-1) with that made the day before (D-2); if there is an increase in the daily profit then the agent will submit the same prices to the market mechanism, with those submitted during the previous trading day (D-1). If this is not the case i.e. the daily profit has either decreased or remained constant, then the agent will try one of two possible alternative routes. It will either increase or decrease the bid price submitted for each of its plants during the next trading day (D), by adding/subtracting a random percentage. This policy forces the agent to "explore" alternative routes of maximising its daily profit i.e. either by increasing its market share or by increasing its bid prices. The agent decides to either subtract or add a random percentage to its plant bid prices based on past experience.

It has been mentioned that agents will either follow a price-raising or a price-lowering policy as part of their bidding strategies. This implies that there must be an upper and a lower limit to the price bid by an agent (price restriction rule). An agent cannot bid a price below that yielding zero profit for the plant in question. In other words, the lowest price an agent can bid for its plant i at period t can be found by equating its revenue with its operating cost. This is shown below;

$$P_t^i = \frac{C(Q_t^i)}{Q_t^i} = aQ_t^i + b = \frac{dC(Q_t^i)}{dt}$$

…(2)

where,

P^i_t is the minimum price the agent is allowed to bid at time t for plant[i] (£/MWh)

Q^i_t is the bid quantity for plant [i] at period t (MW)

$C(Q^i_t)$ is the operating cost for producing quantity Q^i_t with plant [i] (£/MW)

$$[C(Q^i_t) = a(Q^i_t)^2+b(Q^i_t)+c \quad \ldots (3)\ ]$$

Equation (2), shows that the minimum price the agent is allowed to bid for its plant [i] at trading period t is equal to the marginal cost of the plant when generating a power output equal to Q^i_t.

On the other hand, the maximum price the agent is allowed to bid is limited to £1000; this value is considered to be sufficiently higher than the average bid price. The above restrictions ensure that agents behave rationally since they would avoid bidding a price that would generate negative profit (loss) or, on the other hand, result in the loss of market share. There is no obvious reason why the agents would prefer to bid unreasonably high prices under the current market conditions apart from their deliberate exclusion from the auction mechanism.

At this point, it should be emphasised that the learning process is greatly dependent upon the daily repetition of the auction mechanism. It is this feature of the electricity markets that makes possible the continuous update of the profit and market-share objectives of each agent which forces the generating companies into direct competition.

Demand Side Module

Under NETA, the demand side of the electricity market (Supply Companies) is not considered as a passive price taker since it is free to negotiate and engage in bilateral contracts with generators. Furthermore, it can sell or buy electricity in the short-term bilateral market as well as in the spot market. This is a direct consequence of the market decentralisation implemented by NETA. In this project, the demand side of the market is represented by a typical winter day load forecast. This is not seen to interfere with the overall process, since it permits the isolation of the generating agents' bidding behaviour. The aggregate demand curve is shown in Appendix 2.

Software Design

Programming Language

The "C" programming language was selected in order to implement the algorithm for the proposed simulation platform. This seemed to be a natural choice due to the structured nature of the program logic. This point can be further clarified by considering typical tasks performed by the program, for example the task of sorting the bids submitted by the agents as described in Section 3.2.2. It is apparent that such a task is performed by the market mechanism. On the other hand, the task of transferring successful bidding strategies to other plants naturally falls within the responsibilities of the agent.

Furthermore, the "C" programming language is ideal for implementing algorithms with sequential structures and repetitive flow patterns; a new task is initiated once the previous is completed and the whole cycle is repeated for a certain number of

iterations. This is in agreement with the fact that the events taking place in the electricity market model defined in Section 3.2 appear in a sequential manner. In addition, the repetitive nature of the auction mechanism is merely an iterative process.

Although the use of an object-oriented high-level programming language such as "C++" or "Java" offers the attractive feature of coupling data with the tasks that manipulate that data (through the use of objects), a similar feature was achieved in "C" through the extensive use of doubly linked lists and pointers.

Memory Organisation

The challenging task of organising the memory effectively in order to create conditions of easy access to the data residing in memory and at the same time managing that data consistently was achieved through the extensive use of the concept of doubly linked lists throughout the program. A doubly linked list is comprised of a collection of nodes. Each node contains crucial information related to a particular agent, such as the place in memory where the information associated to this agent is kept and the address of the next and previous nodes in the list. A doubly linked list is accessed by reference to its root, which contains information related to the total number of nodes and the location of the first and last nodes in memory.

Data structures are used in order to organise specific variables associated with agents whereas embedded data structures are employed in order to organise variables directly related to plants. The definitions of the agent, plant and cost coefficients data structures can be seen in Appendix 3 (Header File).

Variable Initialisation

Two functions were built in order to create doubly linked lists and initialise the variables pointed by these lists with data provided by the user. The user-defined data is entered in two input data files (plant.dat and demand.dat) in text format. The first file (plant.dat) contains data regarding the agents and their plants whereas the second file (demand.dat) contains a typical day load forecast. The user is responsible for creating these two data files according to the format described in Appendix 4 for the Agent Plant Data Input File (plant.dat) and in Appendix 5 for the Demand Profile Input File (demand.dat).

The initial bid (quantity and price) for each plant is automatically prepared and submitted to the market mechanism by the function that is used to transfer the user-defined data into the place in memory pointed by the nodes of the newly created linked lists. The initial bid price is set to be equal to the marginal cost of the plant at maximum power output and the initial bid quantity is set to be equal to the mean available capacity of the plant.

Function Description

This section is intended to provide a generic description of the main functions comprising the simulation platform. Special attention is paid to the functions directly related to the tasks performed by the market mechanism and the agents. The function prototypes and definitions can be found in Appendix 3.

Market Functions

make_plant_list

The function “make_plant_list” creates a doubly linked list whose nodes point

directly to the plants participating in the simulation. The order at which the plants are registered in the list is not important at this point. The function returns the root of the doubly linked list for general use by the main program.

sort_plant_list

The task of this function is to sort the plants contained in the doubly linked list created by the function "make_plant_list" described above, according to their bid price. The first node in the list is set to point to the plant with the lowest bid price whereas the last node is set to point to the plant with the highest bid price. The sorting algorithm used is a variant of the popular "bubble-sort" algorithm. This sorting algorithm was mainly chosen for its simplicity, although it may prove inefficient as the number of iterations or the number of plants participating in the simulation process is increased. This is due to the subsequent increase in the processing time since the sorting is achieved by comparing the prices bid by each plant on a one-by-one basis.

market_clearing (market_clearing_smp)

This function constitutes the "heart" of the market mechanism. There are two variations of this function; the pay bid price and the pay System Marginal Price (SMP) function. It should be mentioned that the latter is not used in this project but was included for the sake of completeness. The market clearing function receives the root of the sorted doubly linked list returned by the function described above as well as the load demand for the particular trading period. Firstly, it determines whether there is enough capacity available to satisfy the load demand for the particular trading period. If this is the case, it initiates the clearing process by accepting plant bids, starting from the plant pointed to by the first node i.e. the plant with the lowest bid

price and gradually progressing towards the plant pointed to by the last node i.e. the plant with the highest bid price. The clearing process is terminated once the demand is satisfied. The "bid flag" of an accepted plant is set to "1" and the accepted bid price is set to the price bid (pay bid) or the system marginal price (pay SMP). Furthermore, the accepted bid quantity of an accepted plant is set to the quantity bid by this plant unless this is the marginal (last accepted) plant. In that case, the function determines whether or not to accept the full quantity bid by the marginal plant.

The flow diagram shown in Appendix 6 describes the market mechanism functions together with the initialisation functions for hourly and daily bidding;

Agent Functions

agent (agent_daily)

The function "agent" or "agent_daily" is the function that defines the agent learning and bidding behaviour. This is not a stand-alone function in the sense that a number of auxiliary functions are called within the "agent" in order to help complete its tasks; these auxiliary functions are explained further below.

The function "agent" is common to all generating companies but the set of data this function acts upon changes with each different agent. Hence, the behaviour of a generating company is largely dependent on the specific values of the parameters kept in each agent's memory.

The agent function is given the root of the agent doubly linked list as its argument; this enables the function to access the data of each agent in order of appearance in the list, starting from the "head" node and gradually progressing towards the "tail" node. Hence, upon execution of the "agent" function, all generating companies form their

bidding policies in an asynchronous, sequential fashion. This is consistent with the simulation platform specifications (Section 3.2.2), since the agents ignore completely the actions taken by their competitors.

The first part of the "agent" function consists of the calculation of two crucial parameters, namely the profit and the utilisation rate achieved during the previous trading day (D-1). The method of calculating these two parameters varies slightly when considering hourly as opposed to daily bidding; this can be verified by examining the definition of the two functions given in Appendix 3.

The profit made during the previous day (D-1), is calculated by considering the total revenue and the total running cost that incurred during that day. The following equation can be used in order to calculate the profit made by agent [j] in a trading day;

$$\Pi_j = \sum_{i=0}^{n_p} \sum_{k=0}^{23} \left[Q_k^i . P_k^i - C_k^i (Q_k^i) \right] \quad \ldots (4)$$

where, Π_j is the Daily Profit of agent [j] (£).

$n_p + 1$ is the total number of plants owned by agent [j].

Q^i_k is the accepted bid quantity for plant [j][i] (MW),

P^i_k is the accepted bid price for plant [j][i] (£/MWh),

C^i_k is the running cost for plant i when generating power output Q^i_k (£/h),

during the hourly trading period k

The expression used for the calculation of the utilisation rate is given by equation (1) in Section 3.2.2.

Once the calculation of the above parameters is completed, the "agent" function proceeds with the application of the rules mentioned in Section 3.2.2. Firstly, the utilisation rate is compared against the target utilisation rate defined by the user in the "Agent Plant Data Input File" for all of the participating agents. If the achieved utilisation rate during the last trading day (D-1) is less than the target utilisation rate, then the agent subtracts a random percentage off the previous day's bid prices from all the plants across its portfolio. These new, reduced prices together with the plant capacities comprise the next day (D) bids. The random percentage subtracted is generated using a standard library function (random number generator). It should be emphasised that the random number generator is initialised with a new number (known as the "seed") every time the "agent" function is run. This ensures that a different sequence of random numbers is generated at each iteration. The shape of the distribution function from which the random numbers are drawn has little apparent effect on the outcome [11].

The transfer of successful bidding strategies is restricted to plants of the same cycle. For this reason, the function "make_plant_list_cycle" is used in order to create three plant linked lists – one for each cycle. A plant is eligible to enter the linked list corresponding to its cycle, only if it has been accepted by the market mechanism during the previous trading day. The function "sort_plant_list_cycle" sorts each of the three lists according to the daily profit generated by the plants; the "head" node is set to point to the least profitable plant, whereas the "tail" node to the most profitable plant. Furthermore, the function "check_list_cycle" checks whether the successful bidding policy of the plant that has achieved the next highest profit (compared to the

least profitable plant during the trading period t) has been successfully passed to all the plants that underperformed. If this is not the case, then the function "update_price_cycle" is called in order to change the bid price of the plants that have underperformed to that of the plant with the next most successful bidding strategy.

The "agent" function proceeds by comparing the daily profit of the previous trading day (D-1) with that made during the day before (D-2). If the profit either decreased or remained the same, then one out of two possible actions is followed for the reasons that have already been explained elsewhere. Again, the agent either increases or decreases the next day (D) bid prices by a random percentage. The generation of this random percentage follows the same guidelines as in the case explained above i.e. through the implementation of a standard library random number generator. The decision of either subtracting or adding a random percentage to the previous day's bid prices is taken based on past experience; for this reason, two "flags" have been created for each agent. These "flags" can only take one of two values: "0" or "1". The first flag is initialised to "0" and immediately changes to "1" the first time the agent generates a profit which is lower than that generated during the previous trading day. The second flag is randomly assigned a value since firstly the agent needs to explore before it decides whether a price raising or lowering policy will maximise its profits. Its value, though, changes the first time a decision based on that flag is made. Hence, if a certain policy fails to increase the profit-making of the agent, then a different policy will be followed during the next trading day.

On the other hand, if the profit increased then the agent bids the same prices for the

next day with those bid during the previous trading day. The flow diagram shown in Appendix 7 illustrates the sequence of the agent functions mentioned above.

Auxilliary Functions

A number of auxiliary functions were built in order to assist the main functions described above to carry out their tasks as well as to enable the analysis of the results obtained through simulation. These functions together with their location and a small description are shown in the following table;

Table 3- 2 Auxiliary Functions

Function Name	Description	Location (File)
sum_calculations	Calculates accumulated sums for peak and off-peak bid/accepted_bid prices	CPP
ave_calculations	Calculates average peak and off-peak bid/accepted_bid prices	CPP
make_root	Creates the root of a doubly linked list.	H
dbl_make_node	Creates a node in a doubly linked list	H
dbl_insert_data	Inserts data in a doubly linked list	H
reset_list	Initialises the data pointers of every node in the list to point to void (0x0000)	H
display_list display_list2	Traverses the nodes of the list whose contents are displayed	H
display_AGENT	Displays agent-related data residing in the node of the agent list indicated by the function “display_list”	H
display_PLANT	Displays plant-related data residing in the node of the plant list indicated by the function “display_list”	H
display_titles	Displays labels in the output files automatically generated with every simulation.	H
save_list	Saves the contents of a doubly linked list in an output binary file	H
recover_list	Recovers a doubly linked list from an output binary file	H
compare_BIDPRICE	Compares the accepted_bid prices of two plants	H
compare_DailyPlantProfit	Compares the daily profit of two plants	H

Testing the Simulation Platform

Testing was carried out in two phases;

Phase a (Individual Module Testing): Testing of individual modules was carried out in order to verify that each module produced results that corresponded to the design specifications and that identical or similar[4] results were generated in subsequent testing operations (Reproducibility).

Phase b (Overall Software Platform Testing): Testing of the software platform was carried out in order to examine module interoperability and overall system behaviour.

The two testing phases described above required the construction of a number of test input files, which contained carefully selected plant portfolios and demand patterns. The construction of different test input files facilitated the examination of particular aspects of the module operations. The basic "agent plant portfolio" input file comprised of three agents with a small number of plants. The portfolios of the three agents contained plants with similar characteristics; each generating company owned a base-load, a mid-merit and a flexible plant. Although not realistic but valid for testing purposes, the "cycle" parameter was defined to be the same for all the plants in order to monitor the effect of the successful bidding strategy transfer algorithm.

The results taken from tests carried out had shown that the modules fulfil all the design specifications described in this chapter and are fully compatible. The simulation system produces reasonable results; this may be verified by referring to Chapter 4.

[4]The phrase "similar results" is reserved for the agent module, which contains a random number generator. Hence, "identical numerical results" may not be obtained.

Summary

Chapter 3 dealt with various aspects of the design process related to the software platform presented in this project. The initial design specifications were outlined and an overview of the generic requirements of this platform was established. Following that, a detailed analysis of the various modules comprising the platform was presented. A section was dedicated to the explanation of the more practical aspects of the software architecture. The final section dealt with the various procedures followed in order to test the platform.

Chapter 4: Simulation Results

Introduction

This chapter presents the results obtained using the proposed simulation platform. A number of case studies have been chosen to investigate the bidding behaviour of the generating agents in the short-term electricity market as well as to demonstrate the strengths and weaknesses of the simulation platform.

Each case study is defined by a number of output files, which are included as Appendices in this report. Each of the output files contains detailed numerical results related to the generating agents and their corresponding plants. Appropriate reference to these Appendices is made in subsequent sections.

The standard input files "plant.dat" and "demand.dat" are included as Appendices 4 and 5 respectively.

Case Studies

Case Study 1 – Comparison of Daily and Hourly Bidding

The first simulation study is based on the standard input files mentioned in the introduction of this chapter. The agent population is comprised of three mid-merit and one base-load generator. The first agent (Agent [0]) is a comparatively small in capacity mid-merit generator (6734 MW or 17% of the total available capacity), with a relatively inflexible plant portfolio (three cycle "1", four cycle "2" and one cycle "3" plants). The second and third agents have portfolios, which are very similar in diversity and available capacity. The second agent has a mean available capacity of

16438 MW (40% of the total available capacity) and a plant portfolio which is made up of five cycle "1", eight cycle "2" and five cycle "3" plants. Similarly, the third agent has a mean available capacity of 14899 MW (36% of the total available capacity) and a plant portfolio which is made up of three cycle "1", five cycle "2" and three cycle "3" plants. Finally, the fourth agent (Agent [3]) has a mean available capacity of 2972 MW (7% of the total available capacity) and its portfolio is made up entirely of cycle "1" plants.

The first three mid-merit generators were assigned a target utilisation rate of 60% or 0.6. The target utilisation rate is used as a means to replicate the impact of forward contract cover. It is assumed that forward contracting reflects each generator's desire to guarantee itself a minimum target of utilisation for its plants. From this point of view, the figure of 0.6 chosen for mid-merit generators is considered to be representative. On the other hand, 100% or 1.0 target utilisation rate is assigned to the base-load generator.

According to the demand profile used for this case study, the lowest load demand is 24647 MW; this figure represents 60% of the total available capacity. On the other hand, the highest load demand is 32393 MW, which represents a percentage of 79% of the total available capacity.

A sampling period of 10 iterations was chosen in order to record data associated with the various bid prices, profits and target utilisation rates. Hence, the 100 samples recorded in Appendix 8 represent 1000 iterations. Each iteration represents a single trading day, although the demand pattern appearing at every iteration remains unchanged.

The following graph illustrates the formation of the average bidding price by the first agent i.e. Agent [0] under Daily Bidding. This graph also illustrates the average accepted bid price during peak and off-peak hours, based on the offers returned to the agent from the market mechanism at the end of each trading day.

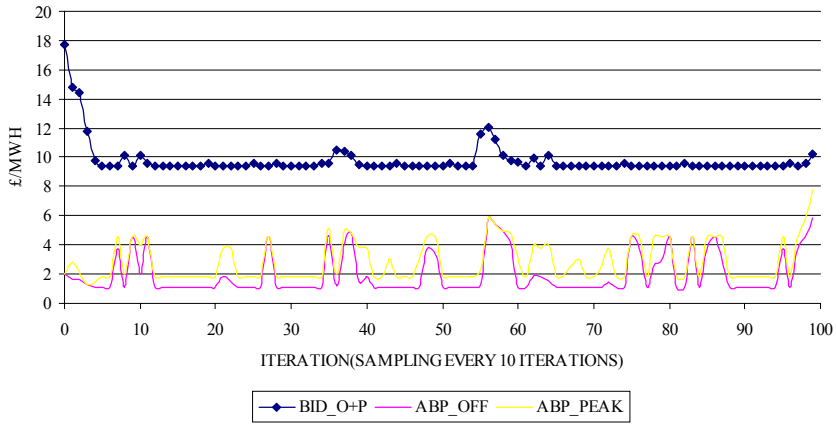


Figure 4- 1 Bid Prices for Agent [0] under Daily Bidding (Case Study 1)

BID_O+P: Bid Price during Peak and Off-Peak Hours; ABP_OFF: Accepted Bid Price during Off-Peak Hours; ABP_PEAK: Accepted Bid Price during Peak Hours

It is observed that the average bid price is the same for off-peak and peak trading hours, since only one bid per day is submitted by the agent to the market mechanism under Daily Bidding. On the other hand, the program calculates two values for the average accepted bid price; one for peak hours and another for off-peak hours. This is possible, since the market mechanism returns one price per hour to each agent at the end of each trading day. Hence, by classifying trading hours in peak and off-peak, corresponding average values can be calculated based on the returned or accepted prices.

The above graph shows that at the early stages of this simulation, the agent is forced to reduce its bid price from the initial[5] value of 17.7 £/MWh to a price which converges at an average of 9.8 £/MWh. The difference between the peak and off-peak accepted bid price is merely due to the increase in demand that occurs during peak trading hours and not due to the agent's ability to capitalise on this event. The increase in demand has simply brought an increase in the number of the agent's plants that have been accepted during the peak trading period and hence the average accepted bid price has experienced a small increase. It is important though, to observe that the average accepted bid price during peak hours is always higher or at least equal to the average accepted bid price during off-peak hours. The figure for this price during the off-peak period is around 2.0 £/MWh, whereas during the peak period around 2.9 £/MWh.

The difference between the bid price and the accepted bid price (peak and off-peak) shown in the above figure, is due to the fact that the achieved utilisation rate of this agent can never reach unity under the current demand profile. What is shown in the above graph is the average value of the prices bid by Agent [0] at each sampling instance for each of its plants. If all of the agents' plants managed to achieve an accepted bid price equal to their bid price for all of the 24 trading hours in each iteration, then the graph representing the accepted bid price would have been the same with the graph representing the bid price for this agent.

The following figure illustrates the bid and accepted bid prices for Plant [0][0];

[5] During the first iteration, the agents bid a different price for each of their plants; this price is equal to the maximum marginal cost of the plant Hence, the initial bid price shown in Figure 4-1 is found by

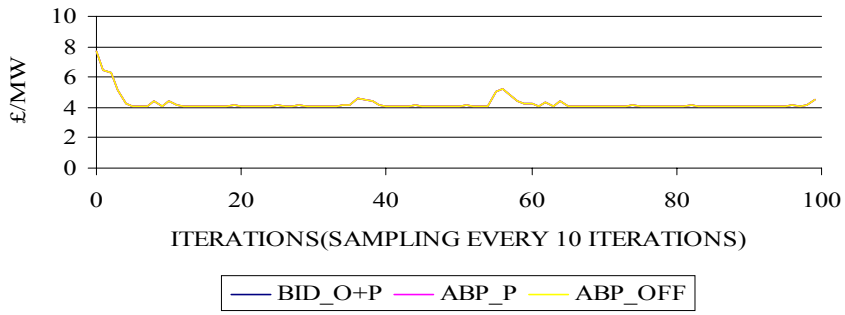


Figure 4- 2 Bid Prices for Plant [0][0] under Daily Bidding (Case Study 1)

BID_O+P: Bid Price during Peak and Off-Peak Hours; ABP_OFF: Accepted Bid Price during Off-Peak Hours; ABP_PEAK: Accepted Bid Price during Peak Hours

The above figure shows that the average accepted bid price during off-peak and peak trading hours for Plant [0][0], is equal to the average price bid for this plant. This is not the case, though, for Plant [0][2] as illustrated in figure 4-3;

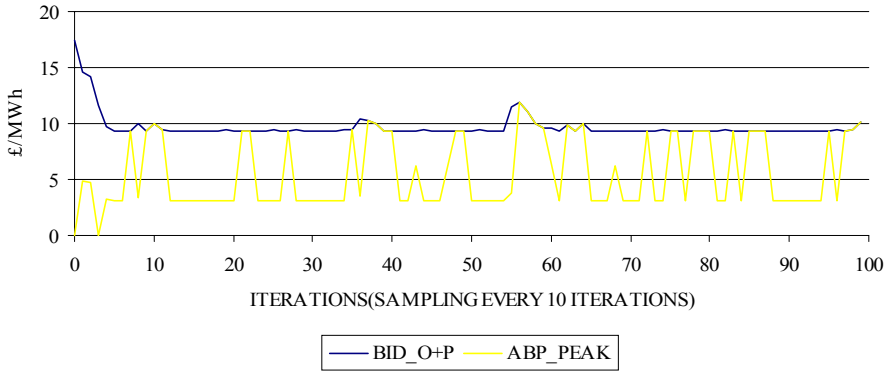


Figure 4- 3 Bid Prices for Plant [0][2] under Daily Bidding (Case Study 1)

BID_O+P: Bid Price during Peak and Off-Peak Hours; ABP_PEAK: Accepted Bid Price during Peak Hours

There is a limitation to the conclusions that can be drawn regarding the bidding behaviour of agents when analysing data taken at the plant level. This is because agents use their portfolio as a whole rather than their individual plants in order to achieve their strategic objectives.

averaging the maximum marginal costs of all the plants in the portfolio of Agent[0].

The formation of the average bidding price under Daily Bidding for the second agent i.e. Agent [1] is shown below;

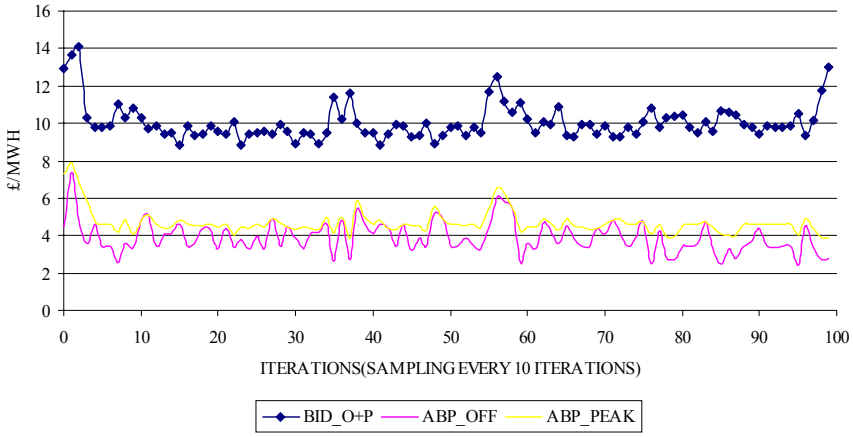


Figure 4- 4 Bid Prices for Agent [1] under Daily Bidding (Case Study 1)

BID_O+P: Bid Price during Peak and Off-Peak Hours; ABP_OFF: Accepted Bid Price during Off-Peak Hours; ABP_PEAK: Accepted Bid Price during Peak Hours

It is observed that, as far as the formation of the biding price is concerned, Agent [1] behaves in a similar way as Agent [0]. This is due to the fact that both agents belong to the same category i.e. that of mid-merit generators. The average bid price for this generator is around 10.0 £/MWh, whereas the average accepted bid price during off-peak trading hours is 3.9 £/MWh and during peak trading hours is 4.7 £/MWh.

The above findings suggest that the third agent i.e. Agent [2] would, again, be expected to behave in a similar manner. An average of 10.7 £/MWh bid price, 4.2 £/MWh off-peak and 5.5 £/MWh peak average accepted bid price, makes this agent the most successful amongst the mid-merit generators in terms of achieved price levels. The off-peak and peak average accepted bid prices, though, are very close in value to those achieved by the second agent (Agent[1]), as it has been anticipated. This is due to the similarities between the portfolios of the two agents both in terms of

diversity –hence flexibility- and overall capacity.

One of the most interesting cases is, perhaps, that of the last agent (Agent[3]) which is the only base-load generator participating in the simulations. The following graphs illustrate the price formation for this agent;

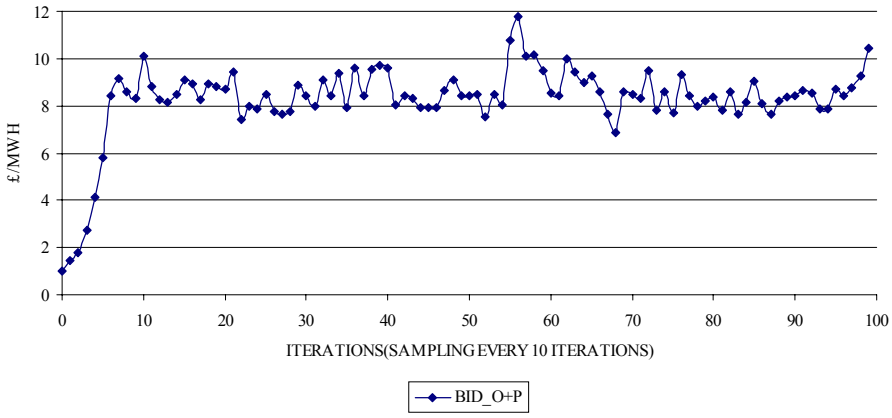


Figure 4- 5 Bid Prices for Agent [3] under Daily Bidding (Case Study 1)

BID_O+P: Bid Price during Peak and Off-Peak Hours

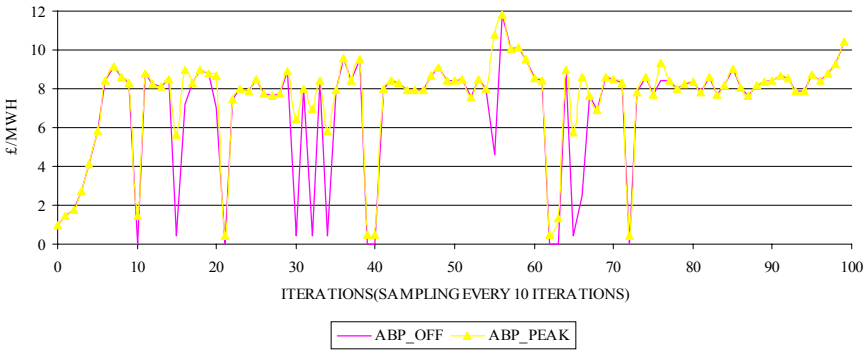


Figure 4- 6 Accepted Bid Prices for Agent [3] under Daily Bidding (Case Study 1)

ABP_OFF: Accepted Bid Price during Off-Peak Hours; ABP_PEAK: Accepted Bid Price during Peak Hours

The average bidding price submitted to the market mechanism by this agent is

8.3 £/MWh whereas the average accepted bid price during peak hours is 7.0 £/MWh and during off-peak hours is 7.5 £/MWh. It can be seen that this agent learns how to increase its bid price up to the point where competition from the other generators occurs. Then the generator follows a price bid policy, which ensures that its utilisation rate is unity most of the times and therefore its market share is maintained. The rapid price spikes shown in the above graph are due to the exploration (trial and error) initiated by the agent in an attempt to identify the best bidding policy.

The advantage of a base generator under these market conditions lies in the fact that there is plenty of opportunity at the initiating stages for its bid prices to increase whereas the mid-merit generators struggle from the initial stages to obtain and maintain their market share. This is due to the fact that a base-load generator will start by bidding the marginal price for each of its plants, which are in fact quite low. The consequence of this is that the market mechanism will always select all of the plants belonging to this agent before it considers any plants owned by other agents, purely due to the price difference, which works in favour of the base-load generator. This means that the target utilisation rate for this agent is fulfilled from the initial stages of the simulation process. On the other hand, as the base-load generator increases its prices, a point will be reached where competition will be injected by the other agents. At this point, the base-load generator follows a price formation policy that results in the stabilisation of its bid price.

Profit (Daily Bidding)

The graph representing the Profit per Available Capacity (PAC), has been plotted for each of the participating agents;

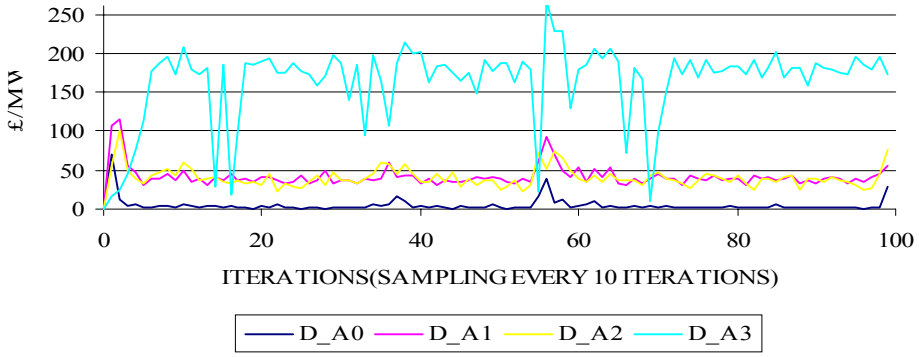


Figure 4- 7 Profit per available capacity under Daily Bidding (Case Study 1)

It is shown that by far the most successful agent in terms of profit-making, is the base-load generator. This achieves an average PAC in the order of 164.0 £/MW whereas the most successful mid-merit generator (Agent [1]) achieves an average PAC in the order of 41.0 £/MW.

Agent [2] achieves a PAC in the order of 40.0 £/MW which is very close to that achieved by Agent [1]. This agrees with the comments previously made regarding the similarity between these two agents. Agent [0] achieves the lowest PAC (4.7 £/MW); this is shown separately in the following figure since the scale difference in the graph above may lead to misinterpretation.

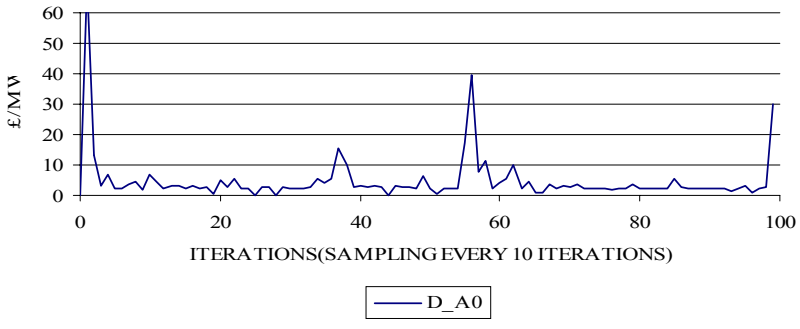


Figure 4- 8 Profit per available capacity for Agent [0] under Daily Bidding (Case Study 1)

Utilisation Rate (Daily Bidding)

The utilisation rate achieved by Agent [0] during the simulation is shown in the following figure;

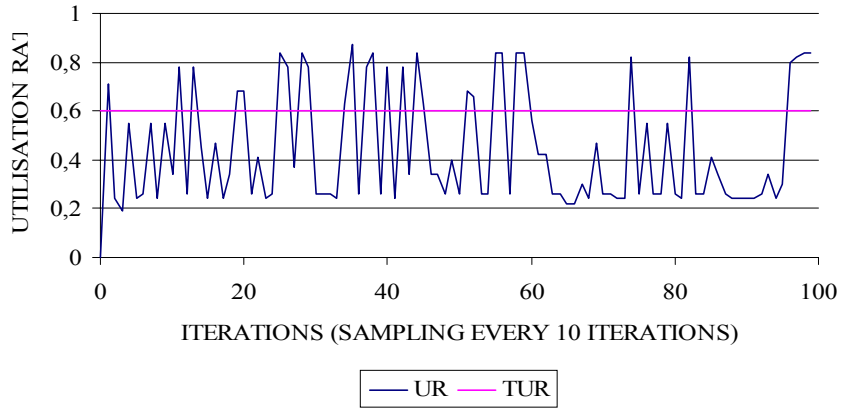


Figure 4- 9 Utilisation Rate for Agent [0] under Daily Bidding (Case Study 1)

UR: Utilisation Rate; TUR: Target Utilisation Rate

The above graph shows that Agent [0] does not maintain an utilisation rate above its target figure. It is evident that this agent is continually forced to return back to its market share objective due to the fierce competition injected by the bigger mid-merit generators.

The following figure illustrates the utilisation rate for Agent [1];

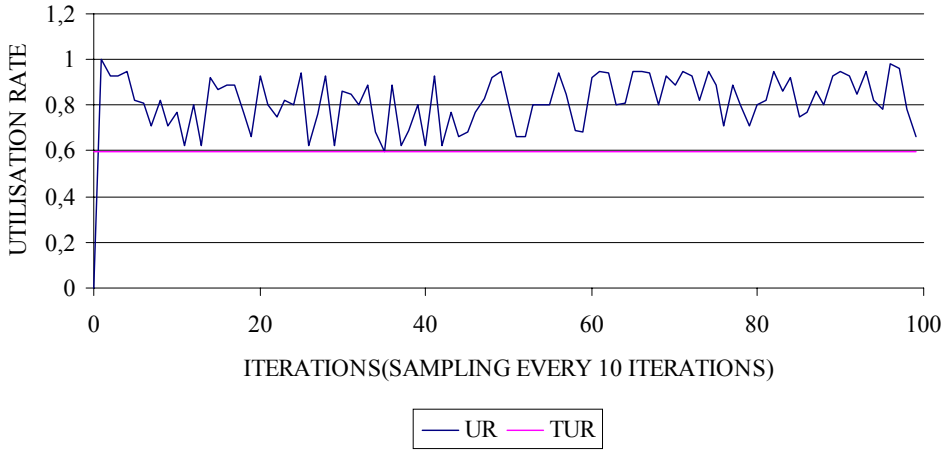


Figure 4- 10 Utilisation Rate for Agent [1] under Daily Bidding (Case Study 1) UR: Utiliation Rate; TUR: Target Utilisation Rate

It is observed that Agent [1] achieves an utilisation rate well above its target figure.

Once the target figure is achieved, this agent is never forced to revise its market-share objective. Similar results are obtained for Agent [2].

The following figure illustrates the utilisation rate for the base-load generator;

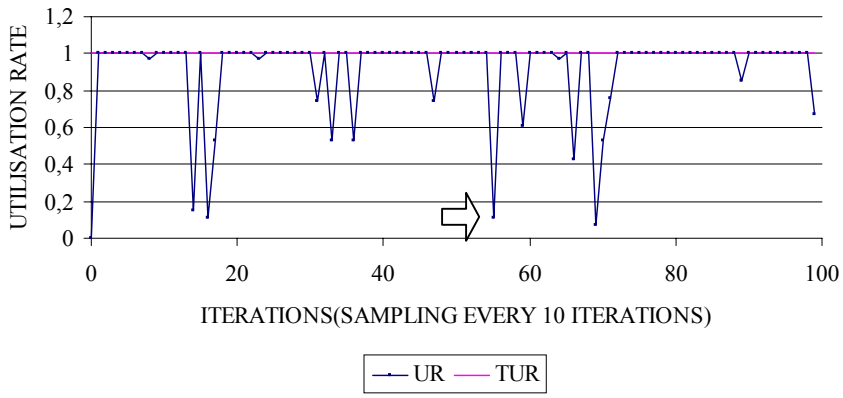


Figure 4- 11 Utilisation Rate for Agent [3] under Daily Bidding (Case Study 1)

UR: Utilisation Rate; TUR: Target Utilisation Rate

Agent [3] is shown to achieve its target utilisation rate for most of the time. The spikes shown in the figure above are due to the agent attempting to maximise its daily profit. Apparently, those attempts were unsuccessful because they resulted into the loss of the agent's market share. The agent regained its market share every time this was lost by following an appropriate corrective action. The following graph shows in detail the values of the bid price and utilisation rate before and after the latter falls to the value of 0.11. This low utilisation point is shown by the arrow in figure 4-11.

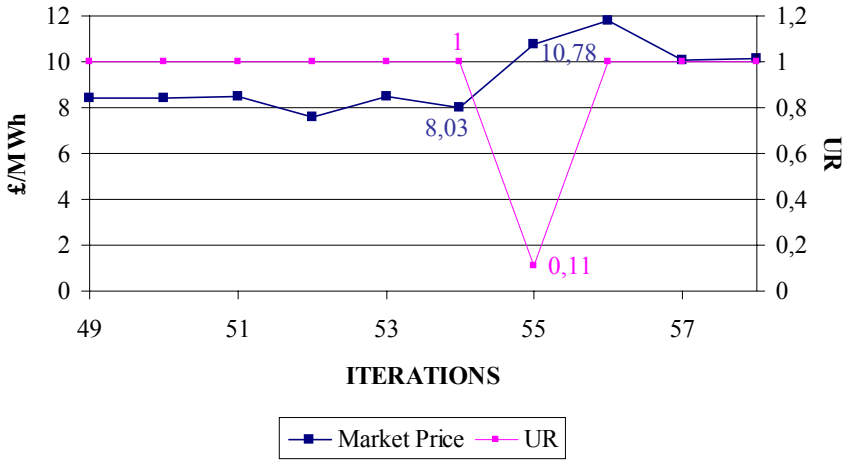


Figure 4- 12 Utilisation rate and bid price for Agent [3] around a low utilisation point.

The above graphs show clearly that the low utilisation rate of 0.11 is due to an increase in the price bid by this generator i.e. from 8.03 £/MWh to 10.78 £/MWh. It should be emphasised that the value of the price at which an agent looses its market share depends on the prices bid by the other agents. During the iteration indicated by the arrow in figure 4-11, the average value bid by the agents for all of their cycle "1" plants[6] is shown below;

Table 4- 1 Weighted Average of Prices Bid for Cycle "1" Plants

Agent	Bid Price for Cycle "1" Plants (Weighted Average) £/MWh
0	5.9
1	5.64
2	5.67
3	10.78

Clearly, Agent [3] lost its market share at that particular iteration due to the fact that it has bid the highest prices among the generators.

[6]Only cycle "1" plants are of interest at this instance since Agent [3] is made up only from this type of plants.

The following figure shows the average off-peak bid and accepted bid price for Agent [0] under Hourly Bidding;

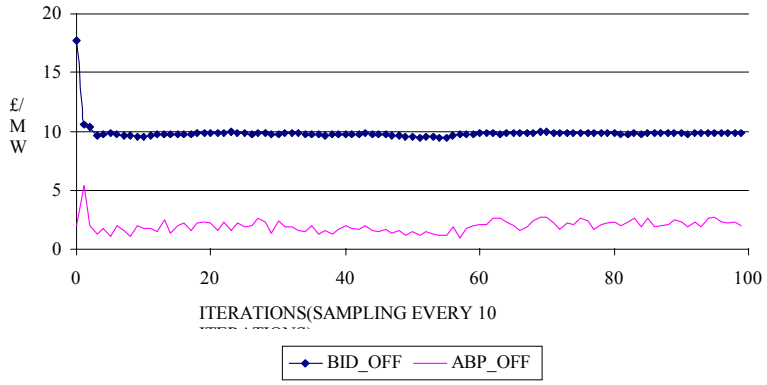


Figure 4- 13 Off-peak Bid Prices for Agent [0] under Hourly Bidding (Case Study 1)

BID_OFF: Bid Price during Off-Peak Hours; ABP_OFF: Accepted Bid Price during Off-Peak Hours

It is observed that this agent bids an average price of 9.9 £/MWh but its average accepted bid price is only 2.0 £/MWh. The formation of the bidding price during peak trading hours presents a whole different picture as shown by the following graph;

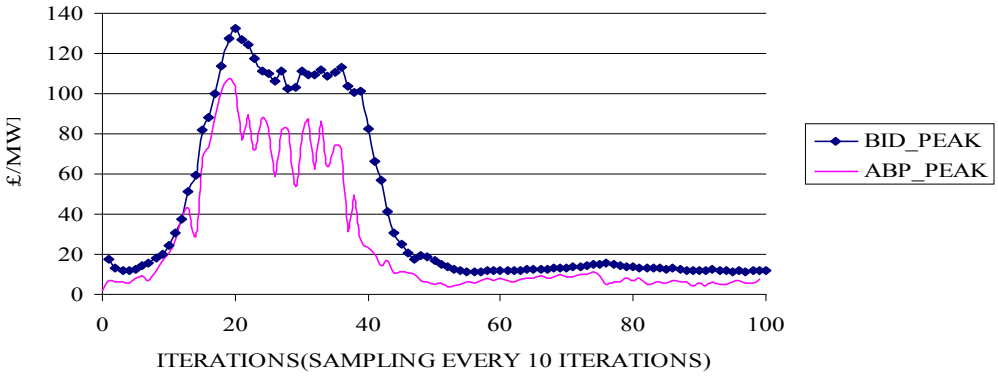


Figure 4- 14 Peak Bid Prices for Agent [0] under Hourly Bidding (Case Study 1)

BID_PEAK: Bid Price during Peak Hours; ABP_PEAK: Accepted Bid Price during Peak Hours

The agent is able to understand that because of the high demand during the peak hours, there is an opportunity for it to increase its bid price. This is pursued up to the point where competition is injected by the rest of the agents, although there is

some indication of temporary "tacit collusion" by all the agents to increase the market price. The competition forces the agent to start decreasing its bid price in an attempt to maintain its market share, hence the sharp negative slope in the above graph. The bid price finally stabilises to an average of 11.9 £/MWh whereas the accepted bid price averages to 10.2 £/MWh.

The second and third agents behave in a similar manner; the following table summarises the results obtained for these agents:

Table 4- 2 Bidding Prices for Agents [1] and [2] under Hourly Bidding

BP_OFF: Bid Price during Off-Peak Hours; ABP_OFF: Accepted Bid Price during Off-Peak Hours; BP_PEAK: Bid Price during Peak Hours; ABP_PEAK: Accepted Bid Price during Peak Hours

AGENT [1]				AGENT [2]			
BP_OFF £/MWh	ABP_OFF £/MWh	BP_PEAK £/MWh	ABP_PEAK £/MWh	BP_OFF £/MWh	ABP_OFF £/MWh	BP_PEAK £/MWh	ABP_PEAK £/MWh
8.0	3.7	12.8	6.8	8.5	3.7	13.8	7.0

The following illustrates the average bid and accepted bid price for Agent [3] during the off-peak trading period;

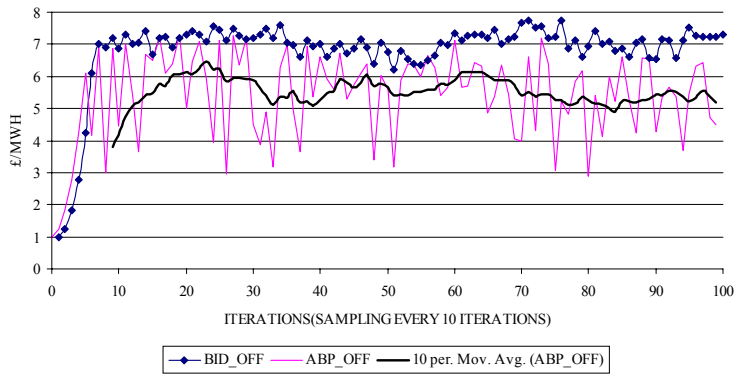


Figure 4- 15 Off-Peak Bid Prices for Agent [3] under Hourly Bidding (Case Study 1)

BID_OFF: Bid Price during Off-Peak Hours; ABP_OFF: Accepted Bid Price during Off-Peak Hours

It is shown that the bid price for this agent converges around 6.8 £/MWh, whereas the its accepted bid price is around 5.4 £/MWh. Once more, the price spikes appearing is

an indication of the agents' trial and error policy.

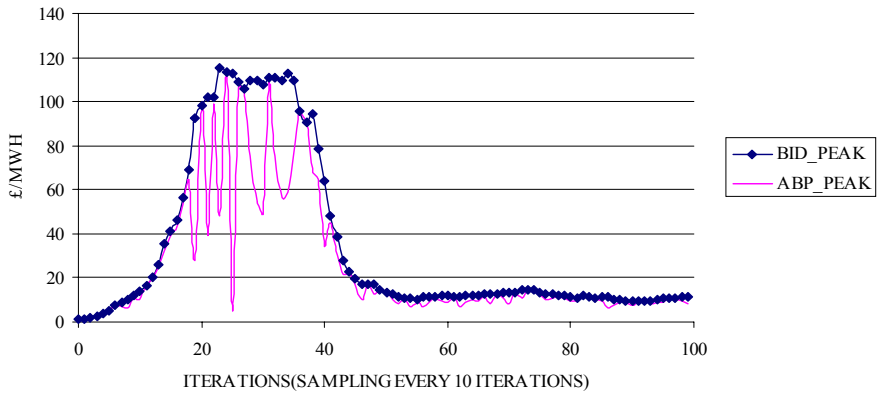


Figure 4- 16 Peak Bid Prices for Agent [3] under Hourly Bidding (Case Study 1)

BID_PEAK: Bid Price during Peak Hours; ABP_PEAK: Accepted Bid Price during Peak Hours

The graph above illustrates the average bid and accepted bid price during the peak trading hours for the base-load generator. It is seen that at the initial stages of the simulation, the agent increases its bid price; at some point, the competition injected by the other generators gains ground and the base-load generator starts to loose some of its market share. The agent decreases its bid price in order to regain its market share; finally convergence occurs for an average bid price of 11.7 £/MWh and an average accepted bid price equal to 9.9 £/MWh.

Profit (Hourly Bidding)

In terms of achieved profits per available capacity (PAC), the following results were obtained for hourly bidding;

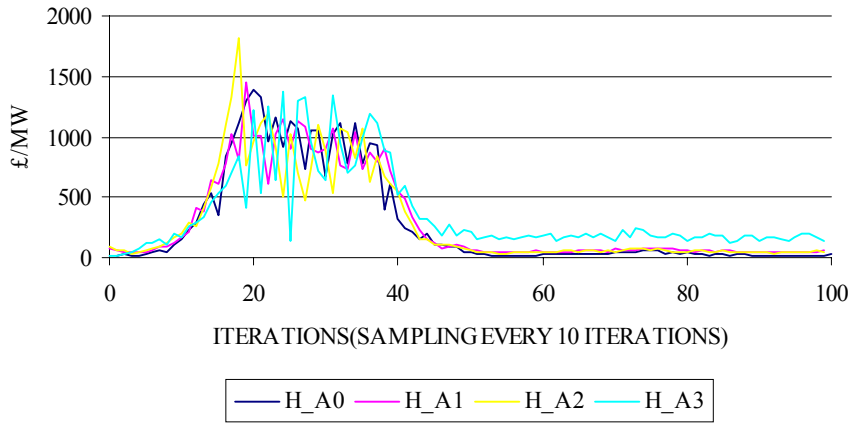


Figure 4- 17 Profit per available capacity under Hourly Bidding (Case Study 1)

The first 460 iterations were considered to constitute an intermediate transient state (learning process) and therefore the focus has been entirely shifted to the part of the PAC graphs where convergence seems to occur;

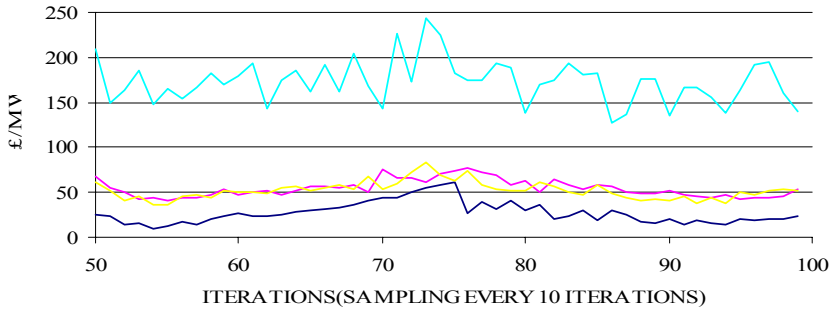


Figure 4- 18 Profit per available capacity under Hourly Bidding – Last 500 iterations

(Case Study 1)

One of the most significant pieces of information conveyed by the group of graphs shown above, is the fact that the base-load agent has, once more, succeeded to obtain and maintain dominance over profit making. This agent managed to achieve a PAC figure of 173.0 £/MW, which is by 9.0 £/MW greater when compared to that achieved during Daily Bidding.

From the mid-merit generators, Agent [1] managed to achieve a PAC of 54.0 £/MW

which represents an increase of 13.0 £/MW when compared to that achieved during Daily Bidding. Agent [2] achieved a PAC equal to 52.0 £/MW which represents an increase of 12.0 £/MW when compared to that achieved during Daily Bidding. Finally, Agent [0] achieved a PAC of 27.0 £/MW, which is significantly higher than the figure of 4.7 £/MW that has been achieved during Daily Bidding.

Hourly versus Daily Bidding

The following table summarises the prices bid by the agents under daily and hourly bidding;

Table 4- 3 Comparison of Daily and Hourly Bid Prices

Agent	Daily Bid Price (£/MWh)	Hourly Bid Price (£/MWh)
0	9.6	11.9
1	10.0	12.8
2	10.7	13.8
3	8.3	11.7

It is noticed that there is an increase in the prices bid during hourly bidding when compared to those during daily bidding. This is in agreement with Bower and Bunn in [11]. The PAC for each agent under the two bidding schemes is summarised by the following table;

Table 4- 4 Comparison of Daily and Hourly PAC

Agent	Daily PAC (£/MW)	Hourly PAC (£/MW)
0	4.7	27.2
1	41.0	54.4
2	40.0	52.2
3	164.0	173.0

The base-load generator seems to have experienced the lowest percentage rise in its profits (5.5%). The two mid-merit generators with similar portfolios, namely Agent

[1] and Agent [2], have experienced a percentage rise in the order of 32.7% and 30.5% respectively. The first agent i.e. Agent [0] has increased its profits by almost 6 times, thus proving that small generators are much likely to survive under Hourly Bidding rather than Daily Bidding.

The above findings suggest that Hourly Bidding as opposed to Daily Bidding offers the agents the necessary flexibility to vary their bid prices according to the trading conditions prevailing during the hourly period under consideration. Hence, the decisions that the agents make have a lesser impact on the overall profitability in the short term. The effects of the corrective actions taken by the agents transpire before steep profit losses are experienced. As a result, agents increase their profits regardless of the diversity or the size of their portfolios.

Case Study 2 – Increased Target Utilisation Rate for Mid-Merit Generators

In this case study, the standard agent plant and demand input files have been used, but the target utilisation rates of the three mid-merit generators have been increased to 0.8 from 0.6. This action is expected to enhance the competition between the participating generators in the simulations; it is the purpose of this study to examine the effect of increased competition on the price levels and the bidding behaviour of the agents.

The results obtained verified that increased competition –in the sense that mid-merit generators look at the market with higher aspiration- results in a decrease in the prices bid by the agents, especially under Hourly Bidding. This is due to the direct competition between the mid-merit generators that are aiming to achieve a

fairly high utilisation rate and therefore market share. Hence, the market share objective takes over the profit maximisation objective. A full listing of the simulation results as well as plots of the main parameters can be found in Appendix 9.

Daily Bidding

The following table summarised the results obtained under Daily Bidding;

Table 4- 5 Bid Prices under Daily Bidding (Case Study 2)

BP: Bid Price during Off-Peak and Peak Hours; ABP_OFF: Accepted Bid Price during Off-Peak Hours; ABP_PEAK: Accepted Bid Price during Peak Hours

AGENT	BP £/MWh	ABP_OFF £/MWh	ABP_PEAK £/MWh
0	9.8 (9.6)	1.3 (2.0)	2.0 (2.9)
1	9.3 (10.0)	3.8 (3.9)	4.5 (4.7)
2	9.9 (10.7)	4.6 (4.2)	5.3 (5.5)
3	7.3 (8.3)	6.1 (7.0)	7.0 (7.5)

The above results suggest that no significant reductions in the bid prices have been recorded under Daily Bidding, when compared to the ones obtained in the first case study[7].

Profit (Daily Bidding)

The following graph illustrates the profit per available capacity (PAC) achieved by the agents during Daily Bidding;

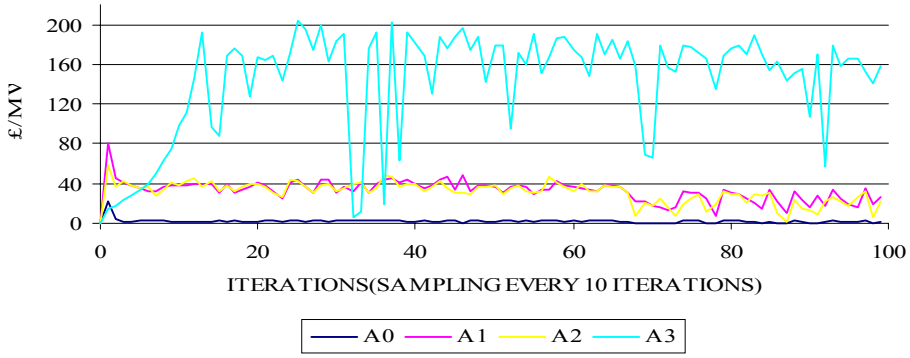


Figure 4- 19 Profit per available capacity under Daily Bidding (Case Study 2)

The base-load generator has again achieved the highest average PAC which is equal to 144.0 £/MW. On the other hand, Agent [0] has achieved the lowest average PAC which is equal to 2.0 £/MW. Agent [1] and Agent [2] have achieved an average PAC equal to 33.0 and 31.0 £/MW respectively.

Hourly Bidding

The results obtained under Hourly Bidding are summarised in the following table;

Table 4- 6 Bid Prices under Hourly Bidding (Case Study 2)

BP_OFF: Bid Price during Off-Peak Hours; ABP_OFF: Accepted Bid Price during Off-Peak Hours; BP_PEAK: Bid Price during Peak Hours; ABP_PEAK: Accepted Bid Price during Peak Hours

AGENT	BP_OFF £/MWh	ABP_OFF £/MWh	BP_PEAK £/MWh	ABP_PEAK £/MWh
0	9.4 (9.9)[8]	1.0 (2.0)	9.4 (11.9)	1.9 (10.2)
1	7.2 (8.0)	3.3 (3.7)	8.2 (12.8)	4.2 (6.8)
2	7.3 (8.5)	3.4 (3.7)	8.5 (13.8)	4.8 (7.0)
3	6.2 (6.8)	5.2 (5.4)	7.2 (11.7)	6.3 (9.9)

[7] The Daily Bid Prices obtained in the first case study are included in parentheses for comparison purposes.
[8] The Hourly Bid Prices obtained in the first case study are included in parentheses for comparison purposes.

The results shown above reveal that all agent's bid prices have been decreased; this is especially true during peak-hours. The explanation given to this is that since the agents have to fulfil their market-share objective before considering their profit maximisation objective, they start by reducing their prices until the former objective is met. In the case of hourly bidding as opposed to daily bidding, this price reduction policy occurs on an hourly basis and therefore a noticeable reduction in the bid prices is produced.

Profit (Hourly Bidding)

The figure below illustrates the PAC achieved by the agents under Hourly Bidding;

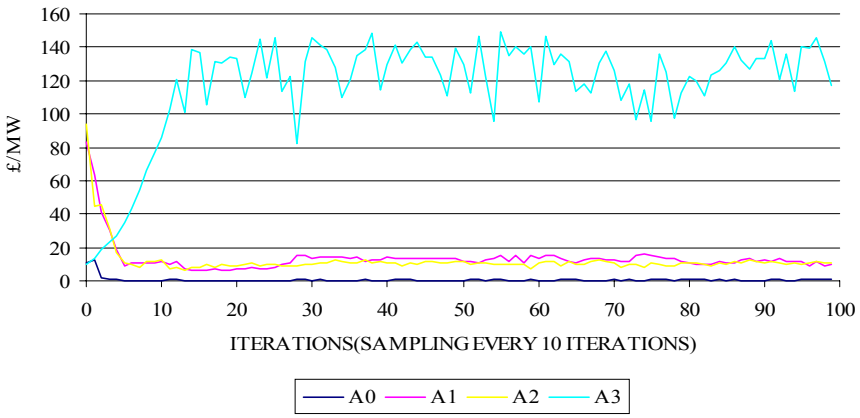


Figure 4- 20 Profit per available capacity under Hourly Bidding (Case Study 2)

The base-load generator is shown to have achieved an average PAC of 117.0 £/MW. This represents a percentage decrease in the order of 32.2% when compared to the previous case study. This average hourly PAC is even lower than that achieved during Daily Bidding, since the mid-merit generators have more chances to refine their bidding policy during Hourly Bidding, thus achieving to bring the market prices even lower. There is a dramatic reduction in the average PAC achieved by Agent [0], which is only 0.6 £/MW (87.2% decrease compared to the previous case study). Agent [1] and Agent [2] have achieved an average PAC in the order of 13.5

and 12 £/MW respectively, which represent an equally dramatic reduction in the order of 75.2% and 74.1%.

Case Study 3 – Increased Demand during Peak Trading Hours

This case study examines the effect of increased demand during peak hours on the prices and the overall behaviour of the generators under hourly bidding. The demand during peak trading hours has been increased from 70 to 90% and from 79 to 95%. The standard input file "plant.dat" (Appendix 4) has been used for this case study. Appendix 10 contains a complete listing of the results generated by this simulation as well as plots of the main parameters.

Hourly Bidding (Off –Peak Hours)

The results obtained during off-peak trading hours, are similar to those presented in the first case study since no changes have been made to the off-peak load profile. These are summarised in the following table;

Table 4- 7 Off-Peak Bid Prices under Hourly Bidding (Case Study 3)

Agent	Bid Price (Weighted Average) £/MWh	Accepted Bid Price (Weighted Average) £/MWh
0	9.8	1.5
1	7.9	3.8
2	8.4	3.8
3	6.4	5.0

Hourly Bidding (Peak Hours)

The results obtained during peak hours differ significantly from those presented in the first case study. The following graph describes the bid price formation for the first agent during peak trading hours;

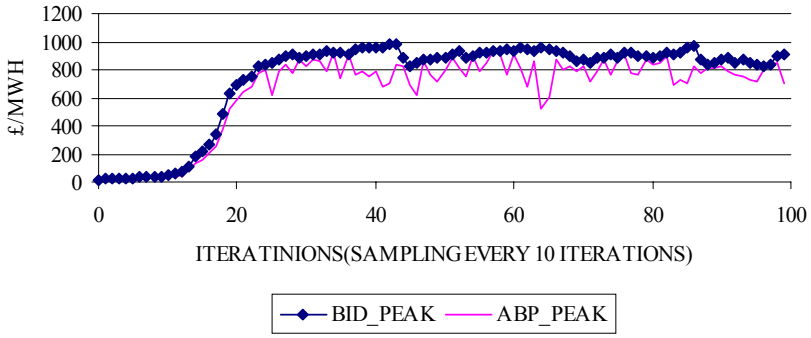


Figure 4- 21 Peak Bid Prices for Agent [0] under Hourly Bidding (Case Study 3)

BID_PEAK: Bid Price during Peak Hours; ABP_PEAK: Accepted Bid Price during Peak Hours

The bid-price for Agent [0] converges at an average of 902.5 £/MWh, whereas the corresponding average accepted bid price is 793.3 £/MWh. The utilisation rate for this agent is close to its target utilisation rate; this was an anticipated result due to the high level of demand.

The following graph illustrates the results obtained for Agent [1];

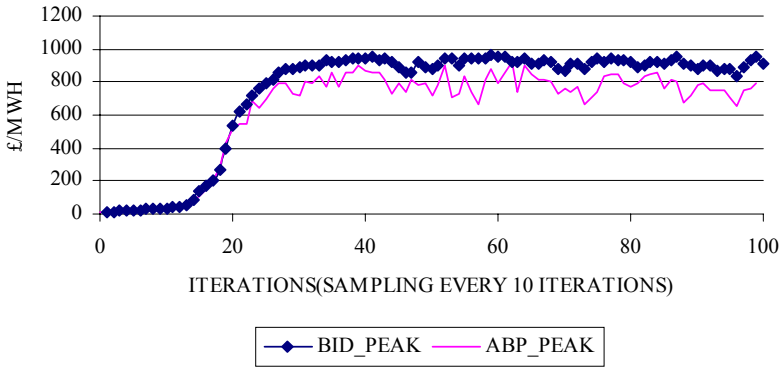


Figure 4- 22 Peak Bid Prices for Agent [1] under Hourly Bidding (Case Study 3)

BID_PEAK: Bid Price during Peak Hours; ABP_PEAK: Accepted Bid Price during Peak Hours

The bid price for this agent converges at an average of 895.4 £/MWh, whereas the average accepted bid price is 770.7 £/MWh. The utilisation rate of this agent is well above its target figure.

Similar results are obtained for Agent [2], as shown in the following figure;

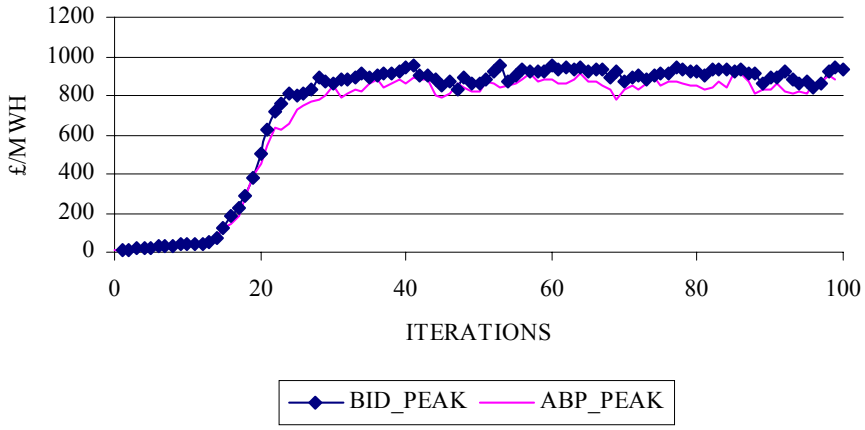


Figure 4- 23 Peak Bid Prices for Agent [2] under Hourly Bidding (Case Study 3)

BID_PEAK: Bid Price during Peak Hours; ABP_PEAK: Accepted Bid Price during Peak Hours

This agent achieved a bid price, which converges at an average of 888.8 £/MWh, whereas its accepted bid price converges at an average value of 825.2 £/MWh. Again, the utilisation rate of this agent is well above its target figure.

The results obtained for the base-load generator are shown below;

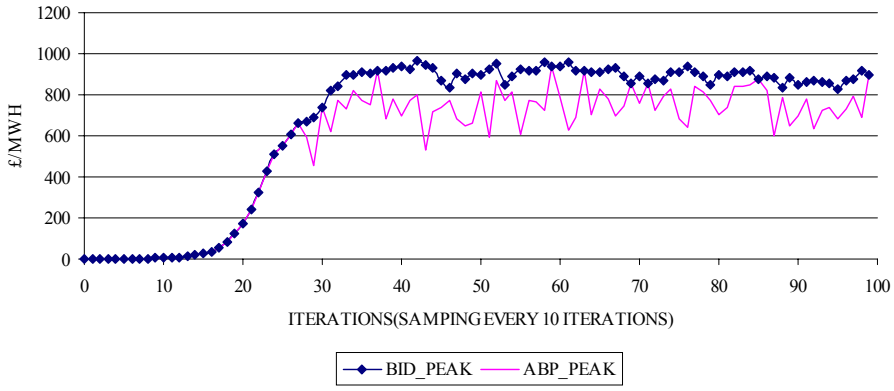


Figure 4- 24 Peak Bid Prices for Agent [3] under Hourly Bidding (Case Study 3)

BID_PEAK: Bid Price during Peak Hours; ABP_PEAK: Accepted Bid Price during Peak Hours

The price bid by this agent converges at the average value of 877.5 £/MWh and the accepted bid price at 740.4 £/MWh.

The following table summarises the results presented above. It also displays in parentheses the corresponding prices obtained in the first case study for the purpose of comparison;

Table 4- 8 Peak Bid Prices under Hourly Bidding (Case Study 3)

Agent	Bid Price (Average) £/MWh	Accepted Bid Price (Average) £/MWh
0	902.5 (11.9)	793.3 (10.2)
1	895.4 (12.8)	770.7 (6.8)
2	888.8 (13.8)	825.2 (7.0)
3	877.5 (11.7)	740.4 (9.9)

The results shown above suggest that at the initial stages of the simulation, "tacit collusion" on behalf of the generators is strongly exercised. This behaviour was anticipated owing to the high load demand during peak trading hours. These extreme levels of demand, provided a fertile ground for the daily maximisation objective of each generator to prevail since their market-share objective has long been satisfied. As a result, the market prices have by far exceeded those obtained in the first simulation study, confirming this way the basic relationship between demand and supply.

Profit (Hourly Bidding)

The profit per available capacity (PAC) made by each of the agents under Hourly Bidding, is illustrated by the following figure;

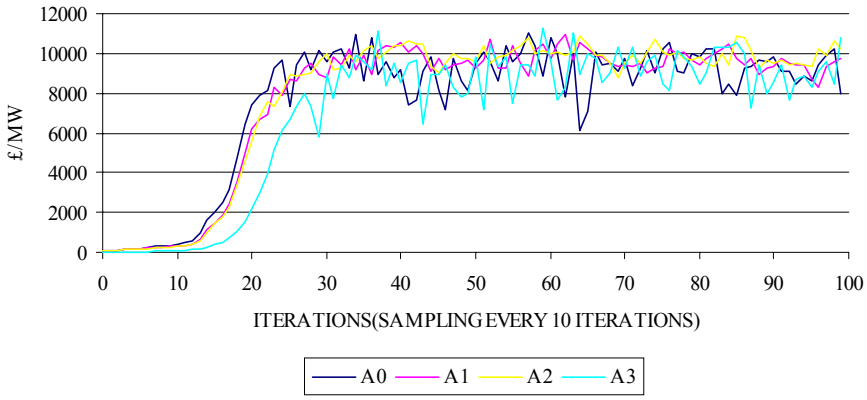


Figure 4- 25 Profit per available capacity under Hourly Bidding (Case Study 3)

In terms of average PAC, the most successful generator seems to be Agent [2] with 9935.3 £/MW. Agent [1] follows with 9706.7 £/MW and Agent [0] with 9214.1 £/MW. The base-load generator achieved a PAC of 9173.6 £/MW.

The above results suggest that all of the agents have taken the opportunities given to them by the sufficiently high load demand during the peak trading period. As a result, they have managed to drive the market prices at a high level and thus achieved to generate extremely high profits.

Case Study 4 – Impact of the Number of Generators on Market Prices

This case study examines the impact of varying the number of generating companies while maintaining the total available capacity constant. Appendix 12 shows the input file that was used in order to simulate a market with six generators. The first five agents represent mid-merit generators, whereas the sixth agent is a base-load generator. The composition and capacity of each of these generators is shown in the following table;

Table 4- 9 Agent portfolio composition and capacity (Appendix 12)

Agent	Cycle 1 (units)	Cycle 2 (units)	Cycle 3 (units)	Agent Capacity (MW)	Percentage of Total Capacity (%)
0	3	3	1	5789.0	14.2
1	2	4	2	10159.0	24.8
2	2	4	2	5360.0	13.0
3	2	3	2	8530.0	20.8
4	2	3	2	8233.0	20.0
5	7	-	-	2972.0	7.2

Appendix 13 shows the input file that was used in order to simulate a market with only two generators. The first agent represents a mid-merit generator, whereas the second agent is a base-load generator. The composition and capacity of each of these generators is shown in the following table;

Table 4- 10 Agent portfolio composition and capacity (Appendix 13)

Agent	Cycle 1 (units)	Cycle 2 (units)	Cycle 3 (units)	Agent Capacity (MW)	Percentage of Total Capacity (%)
0	11	17	9	38071.0	87.0
1	7	-	-	2972.0	13.0

A full listing of the results obtained is included in Appendix 11.

The following figure compares the bid prices achieved under Hourly Bidding with four and six generators;

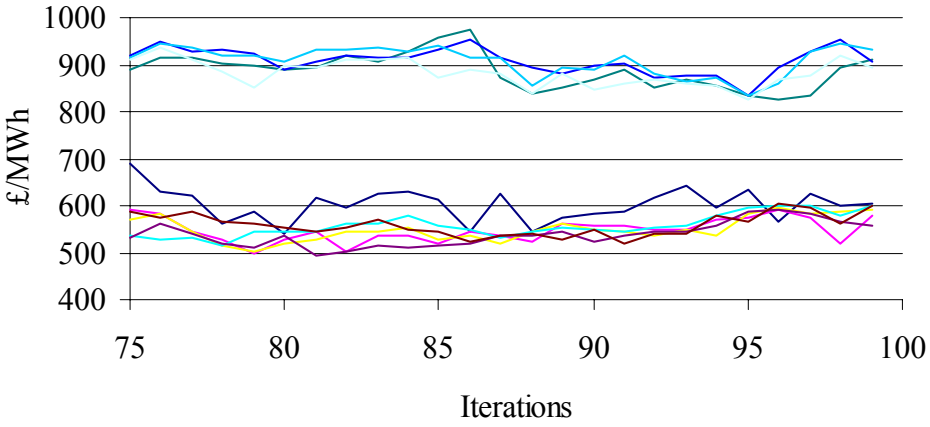


Figure 4- 26 Bid prices under Hourly Bidding (Four generators versus six generators)

There are two groups of graphs in the above figure; the group on the upper part represents the prices bid by four generators. Conversely, the group on the lower part of the figure represents the prices bid by six generators. Clearly, the market prices with six smaller generators are considerably lower when compared to those obtained with four bigger generators. This result was anticipated, since a larger number of generators with smaller portfolios reduce the effect of market power.

The following figure compares the bid prices achieved under Hourly Bidding with four and two generators;

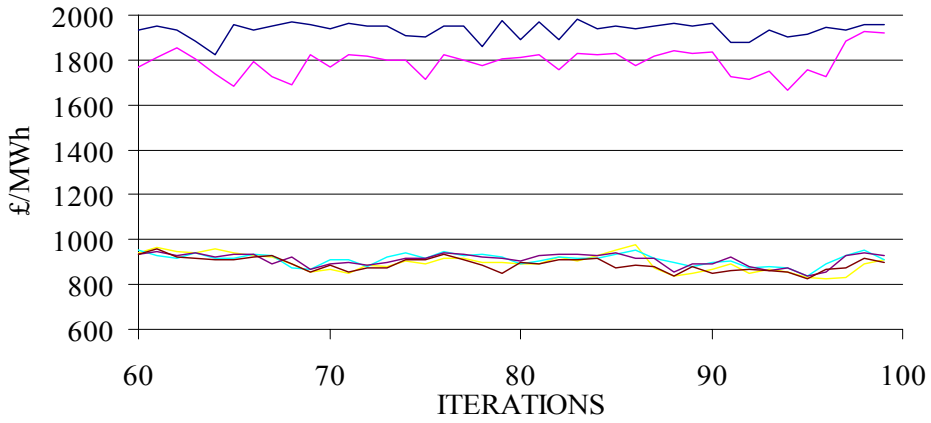


Figure 4- 27 Bid prices under Hourly Bidding (Two generators versus four generators)

There are two groups of graphs in the above figure; the group on the upper part represents the prices bid by two generators, whereas the group on the lower part of the figure represents the prices bid by four generators. It is evident that the market prices with two big generators in the market are substantially higher to those taken with four generators in the market. This result proves that generators with large portfolios tend to exercise market power.

The following table summarises some of the most important statistical parameters related to the three groups of graphs shown in figures 4-26 and 4-27;

Table 4- 11 Statistical Parameters related to the Graphs shown in Figures 4-26 and 4-27

Number of Agents	Agent	Statistical Parameters					
		Mean £/MWh	Median £/MWh	St. Dev. £/MWh	Coefficient of Variation	Min £/MWh	Max £/MWh
2	0	1933.3	1946.7	35.3	54.8	1821.7	1979.9
	1	1792.3	1801.5	58.5	30.6	1667.6	1926.6
4	0	896.5	893.9	39.1	22.9	825.5	974.5
	1	911.2	914.8	26.4	34.5	834.7	954.5
	2	910.7	919.4	28.2	32.3	835.9	947.2
	3	890.2	890.3	30.3	29.4	826.9	959.0
6	0	608.1	617.4	34.2	17.8	687.3	538.9
	1	564.8	552.7	48.4	11.7	697.2	496.9
	2	564.5	548.9	43.5	13.0	683.6	503.3
	3	569.7	558.5	39.0	14.6	675.2	516.1
	4	554.2	542.1	46.8	11.8	690.9	493.6
	5	576.6	563.6	48.2	12.0	708.6	520.9

The standard deviation (STDEV) is a measure of how widely prices are dispersed from the mean value. The coefficient of variation (CV), which is given by the ratio of the mean value and the standard deviation, is a measure of the price variability. Within the same group of agents, generators with high CVs are more stable than generators with lower CVs.

The above statement can be further clarified by considering the group of two generators. The CV of the mid-merit generator is almost twice as that corresponding to the base-load generator. This suggests that although the base-load generator converges around a mean value of 1792.3 £/MWh, its prices vary twice as much around this mean value when compared to the prices that correspond to the mid-merit generator. This result can be justified on the basis that the mid-merit generator has a very large and diverse portfolio. Hence it is expected to be more stable than the comparatively smaller and less versatile base-load generator.

As the number of agents participating in the simulations increases, the portfolios of

the generators become more symmetrical due to the total capacity being maintained constant. As a result, the differences between the CVs of generators that belong to a group of four or six generators seem to decrease. Hence, increasing the number of participating generators in the market results in lower and more stable market prices.

Discussion

This section compares the results shown above with those obtained by Bower and Bunn in [11].

Bower and Bunn have shown that Pay Bid settlement under Hourly Bidding produces the highest prices. Pay Bid settlement under Daily Bidding produces prices, which are greater than those taken under Off-Peak Hourly Bidding but less than those taken under Peak Hourly Bidding. These finding are verified by the results obtained above, although prices and price differences seem to be on a smaller scale. This is due to the different load profile and cost data used in our simulation platform. It has been proven though, that as the demand increases the prices rise sharply and therefore the price differences become more evident.

Bower and Bunn verified that under Pay Bid settlement, base-load generators are forced to participate in price setting thus increasing the risk of being taken out of competition by mid-merit generators. The above results verify this outcome, since the base-load generator is always seen to actively participate in the bidding process. This reduces the competitive pressure on mid-merit generators, which allows prices to rise.

Bower and Bunn also verified that "daily bidding forces generators to try and optimise their bids horizontally across 24 hours, but with hourly bidding it seems that they can more easily optimise, and perhaps event tacitly collude in any given hour". This was exactly what the above results have verified i.e. that hourly bidding allows generators to effectively segment demand into peak and off-peak hours, hence extracting a greater proportion of the consumer surplus (see Table 4-3).

Summary

This chapter has been used in order to demonstrate the functionality of the simulation platform and at the same time examine the bidding behaviour of the agents in the electricity market. This has been achieved through the analysis of the results produced by the simulation of four typical case studies.

The first case study was based on the standard input files and has verified that Hourly Bidding –as opposed to Daily Bidding- yields in higher market prices and PAC. This is due to the fact that the agents are able to refine their policies on an hourly basis by bidding 24 different prices per plant instead of a single price per plant for the whole day.

The second case study examined the impact of increased competition on the market prices. This was achieved by increasing the mid-merit agents' target utilisation rate. It has been verified that market prices under Hourly Bidding decrease, as a result of the agents' efforts to gain their target market share. The PAC for each agent also decreased dramatically due to the market share objective taking over the profit

maximisation objective. There was no significant reduction in prices under Daily Bidding for the two cases.

The third case study dealt with the issue of "tacit collusion" by the participating generators. It has been proven that "tacit collusion" is likely to appear when the competition levels are low, typically when the demand level is extremely high. It has also been proven that market prices rise sharply as the demand increases.

The fourth case study examines the impact of varying the number of generating companies while maintaining the total available capacity constant. It has been proven that a large number of companies with small portfolios result in reduced market prices since the effect of market power is also reduced. Conversely, a small number of companies with large portfolios exercise market power that results in increased market prices.

Chapter 5: Conclusions

Introduction

This chapter gives an account of what has been achieved in this project and provides a summary of the results obtained. The various limitations of the simulation platform are discussed and possible improvements suggested. One section is also dedicated to the future development of the simulation platform.

Achievements

This project has verified that alternative[9] ways of modelling can provide a useful insight into restructured electricity markets. The simulation platform developed in this project may be used to represent any number of generating companies under two bidding schemes (Hourly and Daily). Each generator (agent) may be realistically represented by a discontinuous supply function. Generators may not be symmetrical, thus achieving to examine the behaviour of companies with different types and sizes of portfolios in the market place.

It has been verified that Pay Bid settlement under Hourly Bidding results in higher market prices when compared to Pay Bid settlement under Daily Bidding. The profits generated under Hourly Bidding are also substantially higher. This is due to the ability of generators to effectively segment demand into peak and off-peak hours, hence extract a greater proportion of the consumer surplus. In addition, the decisions that the agents make under Hourly Bidding, have a lesser impact on their short-term

profitability. The effects of the corrective actions taken by the agents take place before substantial profit losses are experienced. As a result, agents increase their profits regardless of the diversity or the size of their portfolios.

Base-load generators under Pay Bid settlement exercise similar bidding policies with those practiced by mid-merit generators. This proves that in the short-term bilateral market, base-load plants face the risk of being driven out of competition by mid-merit generators. As a result, mid-merit generators face less competitive pressure and the market prices start to rise.

The simulation results have also proven that if the market share objective of the generating companies takes over their short-term profit maximisation objective, the market prices decrease. As a result, the profits generated experience a sharp fall. This also suggests that if the generating companies fail to establish their desired contracting position in the long term bilateral market, then the increased competition in the short-term bilateral market will result in reduced market prices.

Market prices under Hourly Bidding vary dramatically with demand. It has been proven that an increase in the demand in the order of 25% yields in market prices that are 60 times greater. This verifies the fact that lack of competition leads to tacit collusion, which results in an increase in the market prices.

The impact of varying the number of generating companies while maintaining the total available capacity constant has been investigated. It has been verified that a

[9] I.e. other than Conventional Economic Modelling

small number of large companies will almost certainly exercise market power. This results in excessively high market prices. On the other hand, a great number of smaller companies may lead to tacit collusion but will also keep the level of competition high enough so that market prices remain at comparatively low levels.

Model Limitations

There are some limitations related to this simulation platform. These are discussed below together with possible suggestions for improvement.

Demand Representation

It has been assumed that due to the low price elasticity, the demand-side of the market may be represented by a 24-hourly load profile. This load profile remains fixed at each iteration. Although this is acceptable for the specific objectives of this project, it may constitute an important limitation when investigating changes in the bidding behaviour as a result of seasonal load variations. This problem may be overcome with the creation of a separate demand profile for each day of the year. The code will have to be slightly modified in order to accommodate this change.

Cost Data

Generators are generally reluctant in providing cost data related to their units. Hence, the cost-coefficients shown in Appendix 4 had to be calculated from the maximum marginal costs found in Appendix A in [11]. This could only be possible by setting the third coefficient of the quadrature cost function to zero. The implication of this is that no-load costs were not taken into consideration in the cost calculations. This does not constitute a problem for this project, since unit constraints were neglected.

Unit Constraints

As mentioned above, unit constraints were neglected in this model. This has the following implications:

1. Over-estimated Profits; This is due to starting and no-load costs not being taken into consideration. On the other hand, since minimum power output constraints are also neglected, additional costs incur when units generate small amounts of power.

2. Violation of mechanical and thermal constraints; Generators are able to schedule any of their units at any given trading period in order to achieve their strategic objectives. This ignores the minimum up time and minimum down time constraints.

Limitations Related to the Software Algorithm

There are some limitations related to the software algorithm. These include the following;

1. Sorting Algorithm; The task of sorting plant bid prices was required in the market functions and some auxiliary agent functions. The sorting algorithm employed is a variant of the "Bubble Sort" algorithm that was mainly chosen for its simplicity. As the number of agents and iterations increases, this algorithm may be proven inefficient as it compares the prices bid by plants on a one-by-one basis. This sorting technique may be replaced with more efficient algorithms such as the "Quicksort" algorithm.

2. Random number generator; It has been shown that the degree of price

variability depends on a number of factors that include the type and size of the agent population. Variability in prices should also be attributed to the random number generator. It is therefore proposed that the percentage added or subtracted to the bid prices be reduced in order to decrease the price variability.

Future Development of the Simulation Platform

The simulation platform could be further developed in order to examine a range of issues related to electricity markets. These are discussed below;

1. Active representation of the demand side of the market; The bidding behaviour of suppliers in the short-term bilateral market could be investigated. This would necessitate the representation of the demand side of the market using supply agents. This change would require the redesign of the market mechanism to incorporate features of the NETA markets, namely the short-term Bilateral Market (-24 hrs PX) and the Balancing Mechanism (-1.0 hrs BM). The strategic objectives of these agents should be directed towards their daily profit maximisation and the minimisation of their exposure in the Spot Market.

2. Effect of plant constraints on bidding policies; This can be achieved by including parameters such as the minimum up/down time and minimum power output in the agent model. The market clearing mechanism should also be revised in order to take into consideration plant constraints.
3. Forecasting capabilities; In this simulation platform, agents have no

forecasting capabilities. They follow a certain policy according to their profit and utilisation rate achieved during the previous day. As an alternative, agents could use this data in order to forecast their next day expected profits and utilisation rate. These parameters can be calculated using exponential smoothing over their past values. Based upon the values of the expected profit and utilisation rate, agents could choose their next day bidding policy.

Summary

This chapter outlined what has been achieved in this project and provided a summary of the results obtained by simulation. The various limitations of the simulation platform were discussed and possible improvements were suggested. One section has also been dedicated to the future development of the simulation platform.

Appendix 1

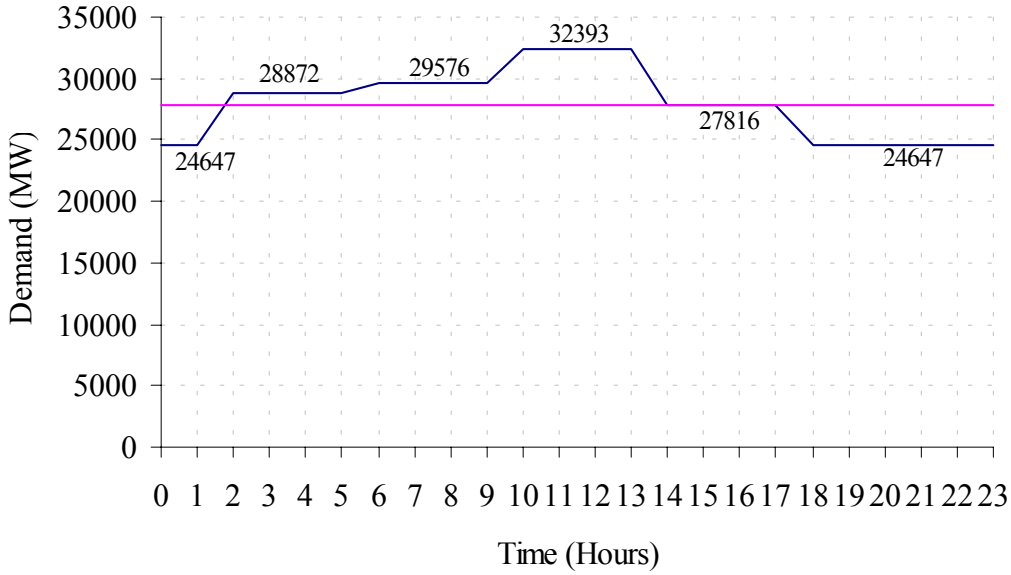
Agent Plant Portfolio

Agent ID	Plant ID	Capacity (MW)	Marginal Cost (£/MWh)	Cost Coefficients C(Q) = aQ² + bQ + c			Cycle
				a	b	c	
0	0	380	7.73	0.0095	0.5	0.0	1
	1	405	8.19	0.0095	0.5	0.0	1
	2	1972	17.49	0.0042	1.0	0.0	1
	3	996	17.49	0.0083	1.0	0.0	2
	4	970	17.64	0.0086	1.0	0.0	2
	5	976	19374	0.0097	1.0	0.0	2
	6	945	20.32	0.01	1.0	0.0	2
	7	90	33.3	0.168	3.0	0.0	3
1	0	690	7.44	0.005	0.5	0.0	1
	1	970	7.58	0.0036	0.5	0.0	1
	2	680	7.87	0.0054	0.5	0.0	1
	3	500	7.87	0.0073	0.5	0.0	1
	4	650	8.19	0.0059	0.5	0.0	1
	5	1935	10.8	0.0025	1.0	0.0	2
	6	1935	11.43	0.0027	1.0	0.0	2
	7	1455	11.66	0.0037	1.0	0.0	2
	8	2005	11.83	0.0027	1.0	0.0	2
	9	680	11.91	0.0082	0.8	0.0	2
	10	1960	12.07	0.0028	1.0	0.0	2
	11	626	12.72	0.0096	0.8	0.0	2
	12	456	13.52	0.014	0.8	0.0	2
	13	685	14.13	0.0087	2.5	0.0	3
	14	484	14.29	0.012	2.5	0.0	3
	15	188	14.28	0.032	2.5	0.0	3
	16	270	27.3	0.046	2.5	0.0	3
	17	269	27.3	0.046	2.5	0.0	3
2	0	1500	7.44	0.0021	1.0	0.0	
	1	740	7.58	0.0048	0.5	0.0	
	2	940	8.03	0.0038	0.8	0.0	
	3	2008	11.55	0.0025	1.5	0.0	
	4	1940	11.57	0.0026	1.5	0.0	
	5	2000	11.62	0.0025	1.5	0.0	
	6	1470	11.77	0.0037	1.0	0.0	
	7	1950	11.78	0.0026	1.5	0.0	
	8	2025	15.46	0.0034	1.5	0.0	

	9	163	27.3	0.08	0.5	0.0	
	10	163	27.3	0.08	0.5	0.0	
3	0	1050	1.00	0.00038	0.2	0.0	1
	1	475	1.00	0.00084	0.2	0.0	1
	2	445	1.00	0.0009	0.2	0.0	1
	3	440	1.00	0.0009	0.2	0.0	1
	4	430	1.00	0.00093	0.2	0.0	1
	5	240	1.00	0.0017	0.2	0.0	1
	6	192	1.00	0.002	0.2	0.0	1

Appendix 2

Aggregate Demand Curve



Appendix 3

Program Code in the "C" Programming Language

(daily.cpp)

```
/*MSc in Electrical Power Engineering
  Modelling of Electricity Markets using Software Agents
  Daily Bidding - 24 hours*/

//Standard Library Header Files
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>

//Global Variables
int k,iters;

//User-Defined Header Files
#include "daily_pri2.h"

//Macros
#define s1(x)  (((AGENT*)(node->data))->x)
#define s2(x)  ((((AGENT*)(node->data))->plant[i])->x)
#define s3(x)  (((PLANT*)(node->data))->x)
#define MAIN int agvc, char *agv[]


//Function Prototypes

/*1  This function is used to read data from a user input file (plant.dat)
     and initialise AGENT/PLANT variables*/
     ROOT *read_agent_data (ROOT **, char *);

/*2  This function is used to read a 24 hour demand profile
     from a user input file (demand.dat)*/
     int read_demand_profile(float *demand,char *filename);

/*3  This function is used to construct a doubly linked list
     whose nodes point at PLANT type structures.*/
     ROOT *make_plant_list(ROOT **root1,ROOT **root2);

/*4  This function is used to sort the plants according to
     their bid price and, therefore, construct the aggregate
     supply function*/
     int sort_plant_list (ROOT **root2, float (*cmp)(void*, void*));

/*5  This function is used to clear the market--The bid_flag
     is set and the price/quantity accepted are determined for each
     plant*/
     int market_clearing_daily (ROOT **root2, float demand);

/*6  System Marginal Price (SMP) version of function 5*/
     int market_clearing_daily_smp (ROOT **root2, float demand);
```

```
/*7*/int plant_profit_calculation_hourly (ROOT **root2);

/*8*/int plant_profit_calculation_daily (ROOT **root2);

/*9*/int make_plant_list_cycle_daily(NODE *node,ROOT **c1,ROOT **c2,ROOT **c3);

/*10*/int  sort_plant_list_cycle_daily  (ROOT  **c1,  ROOT  **c2,ROOT  **c3,float(*cmp)(void*,
void*));

/*11*/int update_price_daily(ROOT **c1, ROOT **c2, ROOT **c3);

/*12*/int check_cycle_daily (ROOT **c1, ROOT **c2, ROOT **c3);

/*13 This function is used to aid calculations carried out by function 12*/
    int sum_calculations_daily(ROOT **root2);

/*14 Function used to calculate the average bid and accepted bid price for
    each plant*/
    int ave_calculations_daily (ROOT **root1, ROOT **root2);

/*15  Agent Function*/
    int agent_daily(ROOT **root);


//MAIN PROGRAM

int main (MAIN){


 ROOT *root1=NULL, *root2=NULL;
 FILE *f[7];
 int i;
 float demand[24];                  //Array for demand values temporary storage

 if(agvc!=3){
   printf("Valid Calling Format-> program_name <plant_input_file>"
                                    "<demand_input_file>\n");
   exit(0);
 }

 //Open primary output streams

 if((f[0]=fopen("out.dat", "a"))==NULL){
  perror("Cannot open file out.dat");
  exit(0);
 }

 if((f[1]=fopen("out1.dat", "a"))==NULL){
  perror("Cannot open file out1.dat");
  exit(0);
 }

 if((f[2]=fopen("out2.dat", "a"))==NULL){
  perror("Cannot open file out2.dat");
  exit(0);
 }
```

```
if((f[3]=fopen("out3.dat", "a"))==NULL){
 perror("Cannot open file out3.dat");
 exit(0);
}

if((f[4]=fopen("out4.dat", "a"))==NULL){
 perror("Cannot open file out4.dat");
 exit(0);
}

if((f[5]=fopen("out5.dat", "a"))==NULL){
 perror("Cannot open file out5.dat");
 exit(0);
}

if((f[6]=fopen("out6.dat", "a"))==NULL){
 perror("Cannot open file out6.dat");
 exit(0);
}

/*Initialise the data segment of the AGENT doubly linked list with values from
  the input file plant.dat*/

if((root1=read_agent_data(&root1,agv[1]))==NULL){
  printf("Could not create agent list...\n");
  exit(EXIT_FAILURE);
}

display_list2(&root1,display_titles,*f);

//Read the demand profile for all 24 hours from the input file demand.dat

if(read_demand_profile(demand,agv[2])!=0){
 printf("Could not create demand profile list...\n");
 exit(EXIT_FAILURE);
}

//The user is promped to enter the number of iterations

printf("Please enter number of iterations:");
scanf("%d",&iters);

for(i=0;i<iters;i++){

 /*Create a doubly linked list and initialise the node data segment to
 point directly to the PLANT structures created by read_agent_data*/

 if((root2= make_plant_list(&root1,&root2))==NULL){
  printf("Could not create plant list...\n");
  exit(EXIT_FAILURE);
 }

 /*Function to adjust the node data pointers to point to PLANT structures
   according to their bid price*/
 if(sort_plant_list(&root2,compare_BIDPRICE)!=0){
  printf("Could not sort plant list...");
```

```
   exit(EXIT_FAILURE);
  }

  //The market is cleared for each of the 24 trading periods

  for(k=0;k<24;k++){

   //This function clears the market

   if(market_clearing_daily(&root2,demand[k])!=0){
    printf("Could not clear the market...");
    exit(EXIT_FAILURE);
   }

   //Calculation of Profits in each of the 24 trading periods

   plant_profit_calculation_hourly(&root2);

   if(sum_calculations_daily(&root2)!=0){
    printf("Could not calculate average quantities...");
    exit(EXIT_FAILURE);
   }

  }

  //Calculation of the Daily Profits
  plant_profit_calculation_daily(&root2);

  //Calculation of Average Quantities
  ave_calculations_daily (&root1, &root2);

  //Results inserted in output files

 if((i%10)==0){
  printf("%d\n", i);
   display_list(&root2, display_PLANTb, f[0]);
   display_list2(&root1,display_AGENTb,*f);
 }

  //Control is passed on to the agents
  if(agent_daily(&root1)!=0)
   printf("Error with the agent...");

 }

 for (i=0;i<7;i++){
  fclose(f[i]);
  if(ferror(f[i]))
   perror("Error with file out.dat...");
 }

return 0;
}

//---------------------------------------------------------------------------
//Function 1
```

```
ROOT *read_agent_data (ROOT **root, char *filename) {

 NODE *node;
 AGENT *data;
 int plant_flag=1,agent_flag=1, i=0, j=0,m;

 FILE *f1;

 randomize();

 if((f1=fopen(filename, "r"))==NULL){
  perror("Cannot Open File...");
  exit(EXIT_FAILURE);
 }

 while(agent_flag==1){

  data=NEW(AGENT);

  while(plant_flag ==1){

   (data->plant[i])=NEW(PLANT);

   fscanf(f1,"%f %f",&((data->plant[i])->cap),&((data->plant[i])->mc));
   fscanf(f1,"%f %f %f %d %d\n",&(data->plant[i]->cost_coef.cca)
          ,&(data->plant[i]->cost_coef.ccb),&(data->plant[i]->cost_coef.ccc)
          ,&(data->plant[i]->cycle),&plant_flag);

   data->agent_id = (data->plant[i])->agent_id=j;
   (data->plant[i])->id =i;
   (data->plant[i])->bid_quantity=(data->plant[i])->cap;
   (data->plant[i])->bid_price = (data->plant[i])->mc;
   (data->plant[i])->prev_bid_q=0.0;
   (data->plant[i])->prev_bid_p=0.0;

   for(m=0;m<24;m++){
    (data->plant[i])->accept_bid_q[m]=0.0;
    (data->plant[i])->accept_bid_p[m]=0.0;
    (data->plant[i])->bid_flag[m] =0;
   }

   data->plant_num = ++i;
  }

  fscanf(f1,"%f %d\n",&data->tur, &agent_flag);
  data->profit =data->prev_profit=0.0;
  data->ur=data->prev_ur=0.0;
  data->gene[0]=0;
  data->gene[1] = random(2);

  if(dbl_insert_data(root,data,1)!=0){
   printf("Unable to add agent in the list");
   exit(EXIT_FAILURE);
  }

  plant_flag=1;
  ++j;
```

```
 i=0;
}

 node = (*root)->head;

 for(j=0;j<(*root)->num;j++){
  for(i=0;i<((AGENT*)(node->data))->plant_num;i++){
   ((((AGENT*)(node->data))->plant[i])->pnt_agent) = node;
  }
  node= node->next;
 }

 if(ferror(f1))
 perror("Error Reading Source File...");

 fclose(f1);

return(*root);
}

//-------------------------------------------------------------------------
//Function 2

int read_demand_profile (float *demand, char *filename) {

 int i;
 FILE *f2;

 if((f2=fopen(filename, "r"))==NULL){
  perror("Cannot Open File...");
  exit(EXIT_FAILURE);
 }

 for(i=0;i<24;i++){
  fscanf(f2,"%f\n",(demand+i));
 }

 if(i<23)
  printf("\nThere are no 24 demand values in the demand file...\n"
         "Please check and re-run\n");

 if(ferror(f2))
  perror("Error Reading Source File...");

 fclose(f2);

return 0;
}

//-------------------------------------------------------------------------
//Function 3

ROOT *make_plant_list(ROOT **root1,ROOT **root2){

 NODE *node = (*root1)->head;
 int i,j;
```

```
 reset_list(root2);

 if((*root1)!=NULL){
  for(j=0;j<(*root1)->num;j++){
   for(i=0;i<sc2(plant_num);i++){
    dbl_insert_data(root2, (((AGENT*)(node->data))->plant[i]), 1);
   }
   node=node->next;
  }
 }

return (*root2);
}

//---------------------------------------------------------------------------
//Function 4

int sort_plant_list (ROOT **root2, float (*cmp)(void*, void*)){

 NODE *node= (*root2)->head;
 NODE *node_next = node->next;
 void *temp;
 int i,j;

 for(i=0; i<((*root2)->num); i++){

  for(j=1;j<((*root2)->num);j++){

   if(cmp((node_next->data),(node->data))<0.0){

    temp = node->data;
    (node->data) = (node_next->data);
    (node_next->data) = temp;
   }
   node_next = node_next->next;
  }
  node = node->next;
  node_next=(*root2)->head;
 }

return 0;
}

//---------------------------------------------------------------------------
//Function 5

int market_clearing_daily (ROOT **root2, float demand){

 NODE *node = (*root2)->head;
 int i;
 float total_cap=0.0;

 if(!VALID(root2))
  return (-1);

 if(demand<=0.0){
  printf("Demand should be positive integer...\n");
```

```
 return(-1);
}

for(i=0;i<(*root2)->num;i++){
 ((PLANT*)(node->data))->accept_bid_q[k]=0.0;
 ((PLANT*)node->data)->accept_bid_p[k]=0.0;
 ((PLANT*)node->data)->bid_flag[k]=0;
 total_cap = total_cap + ((PLANT*)node->data)->bid_quantity;
 node = node->next;
}

if(total_cap<demand){
 printf("There is not enough capacity to satisfy the load requirements\n");
 return(-1);
}

node = (*root2)->head;

for(i=0;i<(*root2)->num;i++){

 ((PLANT*)node->data)->accept_bid_q[k]=((PLANT*)node->data)->bid_quantity;
 ((PLANT*)node->data)->accept_bid_p[k]=((PLANT*)node->data)->bid_price;
 demand = demand -((PLANT*)node->data)->accept_bid_q[k];

 if(demand>0.0){

  ((PLANT*)node->data)->bid_flag[k] = 1;
  node=node->next;
 }

 else if(demand<0.0) {

  ((PLANT*)node->data)->accept_bid_q[k]                =                ((PLANT*)node->data)-
>accept_bid_q[k]+demand;
  ((PLANT*)node->data)->bid_flag[k] = 1;
  break;
 }

 else {
  ((PLANT*)node->data)->bid_flag[k] = 1;
  break;
 }
}

return 0;
}

//---------------------------------------------------------------------------
//Function 6

int market_clearing_daily_smp (ROOT **root2, float demand){

 NODE *node = (*root2)->head;
 int i;
 float total_cap=0.0;
 float temp_price;
```

```
if(!VALID(root2))
 return (-1);

if(demand<=0.0){
 printf("Demand should be positive integer...\n");
 return(-1);
}

for(i=0;i<(*root2)->num;i++){
 ((PLANT*)node->data)->accept_bid_q[k]=0.0;
 ((PLANT*)node->data)->accept_bid_p[k]=0.0;
 ((PLANT*)node->data)->bid_flag[k]=0;
 total_cap = total_cap + ((PLANT*)node->data)->bid_quantity;
 node = node->next;
}


if(total_cap<demand){
 printf("There is not enough capacity to satisfy the load requirements\n");
 return(-1);
}

node = (*root2)->head;

for(i=0;i<(*root2)->num;i++){

 ((PLANT*)node->data)->accept_bid_q[k]=((PLANT*)node->data)->bid_quantity;
 temp_price = ((PLANT*)node->data)->bid_price;
 //((PLANT*)node->data)->accept_bid_p=((PLANT*)node->data)->bid_price;
 demand = demand -((PLANT*)node->data)->accept_bid_q[k];

 if(demand>0.0){

  ((PLANT*)node->data)->bid_flag[k] = 1;
  node=node->next;
 }

 else if(demand<0.0) {

  ((PLANT*)node->data)->accept_bid_q[k]                =                ((PLANT*)node->data)-
>accept_bid_q[k]+demand;
  temp_price = ((PLANT*)node->data)->bid_price;
  ((PLANT*)node->data)->bid_flag[k] = 1;
  break;
 }

 else {
  ((PLANT*)node->data)->bid_flag[k] = 1;
  temp_price = ((PLANT*)node->data)->bid_price;
  break;
 }
}

node = (*root2)->head;

for(i=0;i<(*root2)->num;i++) {
```

```
  if(((PLANT*)node->data)->bid_flag[k] == 1)
   ((PLANT*)node->data)->accept_bid_p[k] = temp_price;

  node = node->next;
 }

 return 0;
}

//-------------------------------------------------------------------------
//Function 7

int plant_profit_calculation_hourly (ROOT **root2) {

  NODE *node = (*root2)->head;
  int i;

  for(i=0;i<(*root2)->num;i++){
   s3(plant_profit[k])= (s3(accept_bid_p[k])*s3(accept_bid_q[k]))-
                    ((s3(cost_coef.cca))*(s3(accept_bid_q[k]))*(s3(accept_bid_q[k])) +
                     (s3(cost_coef.ccb))*(s3(accept_bid_q[k]))+ (s3(cost_coef.ccc)));
   node = node->next;
  }

return 0;
}


//-------------------------------------------------------------------------
//Function 8

int plant_profit_calculation_daily (ROOT **root2) {

  NODE *node = (*root2)->head;
  int i;

  for(i=0;i<(*root2)->num;i++){
    s3(daily_plant_profit) = 0.0;
    node = node->next;
  }

  node = (*root2)->head;

  for(i=0; i<(*root2)->num;i++){
   s3(daily_plant_profit)= s3(daily_plant_profit) + s3(plant_profit[k]);
   node = node->next;
  }

return 0;
}

//-------------------------------------------------------------------------
//Function 9

int make_plant_list_cycle_daily(NODE *node,ROOT **c1,ROOT **c2,ROOT **c3){

 int i;
```

```
 reset_list(c1);
 reset_list(c2);
 reset_list(c3);

 if((node->data)!=NULL ){

   for(i=0;i<sc2(plant_num);i++){

     if(sc3(daily_plant_profit)!=0.0) {

      switch(sc3(cycle)) {

       case 1: dbl_insert_data(c1, (((AGENT*)(node->data))->plant[i]), 1);
               break;

       case 2: dbl_insert_data(c2, (((AGENT*)(node->data))->plant[i]), 1);
               break;

       case 3: dbl_insert_data(c3, (((AGENT*)(node->data))->plant[i]), 1);
               break;
     }
    }
   }
 }



 return 0;
}

//-------------------------------------------------------------------------
//Function 10

int sort_plant_list_cycle_daily (ROOT **c1, ROOT **c2,ROOT **c3,float(*cmp)(void*, void*)){

 NODE *node1;
 NODE *node2;
 NODE *node3;
 NODE *node_next1;
 NODE *node_next2;
 NODE *node_next3;
 void *temp1, *temp2,*temp3;

 int i,j;

 if(*c1!=NULL){

  node1= (*c1)->head;
  node_next1 = node1->next;

  for(i=0; i<((*c1)->num); i++){
    for(j=1;j<((*c1)->num);j++){
      if(cmp((node_next1->data),(node1->data))<0.0){
       temp1 = node1->data;
       (node1->data) = (node_next1->data);
       (node_next1->data) = temp1;
```

```
     }
    node_next1 = node_next1->next;
   }
  node1 = node1->next;
  node_next1=(*c1)->head;
 }

 node1 = (*c1)->head;

 for(i=0;i<(*c1)->num;i++) {
  printf("%f\n", ((PLANT*)(node1->data))->bid_price);
  node1 = node1->next;
 }
}

//---

if(*c2!=NULL){

 node2= (*c2)->head;
 node_next2 = node2->next;

 for(i=0; i<((*c2)->num); i++){

  for(j=1;j<((*c2)->num);j++){

   if(cmp((node_next2->data),(node2->data))<0.0){

    temp2 = node2->data;
    (node2->data) = (node_next2->data);
    (node_next2->data) = temp2;
   }
  node_next2 = node_next2->next;
  }
 node2 = node2->next;
 node_next2=(*c2)->head;
 }
 node2 = (*c2)->head;

 for(i=0;i<(*c2)->num;i++) {
  printf("%f\n", ((PLANT*)(node2->data))->bid_price);
  node2 = node2->next;
 }


}
//--

if(*c3!=NULL){

 node3= (*c3)->head;
 node_next3 = node3->next;

 for(i=0; i<((*c3)->num); i++){

  for(j=1;j<((*c3)->num);j++){
```

```
   if(cmp((node_next3->data),(node3->data))<0.0){

     temp3 = node3->data;
     (node3->data) = (node_next3->data);
     (node_next3->data) = temp3;
   }
  node_next3 = node_next3->next;
  }
 node3 = node3->next;
 node_next3=(*c3)->head;
 }

 node3 = (*c3)->head;

  for(i=0;i<(*c3)->num;i++) {
   printf("%f\n", ((PLANT*)(node3->data))->bid_price);
   node3 = node3->next;
  }
 }


 return 0;
}

//-------------------------------------------------------------------------
//Function 11

int check_cycle_daily(ROOT **c1, ROOT **c2,ROOT **c3) {

 NODE *node1, *node2, *node3;
 NODE *node_next1, *node_next2, *node_next3;
 int a=1,b=1,c=1, flag;


 int i;


 if(VALID(c1)){

  if(NUM(c1)>=2){

   node1= (*c1)->head;
   node_next1 = node1->next;

  for (i=1;i<NUM(c1);i++){

    if( ((PLANT*)(node_next1->data))->bid_price == ((PLANT*)(node1->data))->bid_price){
      node_next1 = node_next1->next;
      a = 1;
    }

    else{
     a = 0;
     break;
    }
  }
 }
```

```
}

 //---


 if(VALID(c2)){

  if(NUM(c2)>=2){

   node2= (*c2)->head;
   node_next2 = node2->next;

   for (i=1;i<NUM(c2);i++){

    if( ((PLANT*)(node_next2->data))->bid_price == ((PLANT*)(node2->data))->bid_price){
      node_next2 = node_next2->next;
      b = 1;
    }

    else{
     b = 0;
     break;
    }
  }
 }
}
 //--

if(VALID(c3)){

  if(NUM(c3)>=2){

   node3= (*c3)->head;
   node_next3 = node3->next;

   for (i=1;i<NUM(c3);i++){

    if( ((PLANT*)(node_next3->data))->bid_price == ((PLANT*)(node3->data))->bid_price){
      node_next3 = node_next3->next;
      c = 1;
    }

    else{
     c = 0;
     break;
    }
   }
  }
}


 if((a==1) && (b==1) && (c==1))
  flag = 1;
 else
  flag =0;

return flag;
```

```
}

//---------------------------------------------------------------------------
//Function 12

int update_price_daily(ROOT **c1, ROOT **c2,ROOT **c3) {

 NODE *node1, *node2, *node3;
 NODE *node_next1, *node_next2, *node_next3;


 int i;


 if(VALID(c1)){

  if(NUM(c1)>=2){

   node1= (*c1)->head;
   node_next1 = node1->next;

   for (i=2;i<NUM(c1);i++){

    if( ((PLANT*)(node_next1->data))->bid_price == ((PLANT*)(node1->data))->bid_price)
      node_next1 = node_next1->next;
    else

    break;
   }

   while(((PLANT*)(node1->data))->bid_price != ((PLANT*)(node_next1->data))->bid_price){
    ((PLANT*)(node1->data))->bid_price = ((PLANT*)(node_next1->data))->bid_price;
    node1 = node1->next;
   }
  }
 }


 //---

 if(VALID(c2)){

  if(NUM(c2)>=2){

   node2= (*c2)->head;
   node_next2 = node2->next;

   for (i=2;i<NUM(c2);i++){

    if( ((PLANT*)(node_next2->data))->bid_price == ((PLANT*)(node2->data))->bid_price)
      node_next2 = node_next2->next;
    else

    break;
   }

   while(((PLANT*)(node2->data))->bid_price != ((PLANT*)(node_next2->data))->bid_price){
```

```
    ((PLANT*)(node2->data))->bid_price = ((PLANT*)(node_next2->data))->bid_price;
    node2 = node2->next;
   }
  }
 }

//--

 if(VALID(c3)){

  if(NUM(c3)>=2){

   node3= (*c3)->head;
   node_next3 = node3->next;

   for (i=2;i<NUM(c3);i++){

    if( ((PLANT*)(node_next3->data))->bid_price == ((PLANT*)(node3->data))->bid_price)
      node_next3 = node_next3->next;
    else

    break;
   }

   while(((PLANT*)(node3->data))->bid_price != ((PLANT*)(node_next3->data))->bid_price){
    ((PLANT*)(node3->data))->bid_price = ((PLANT*)(node_next3->data))->bid_price;
    node3 = node3->next;
   }
  }
 }


 return 0;
}

//-------------------------------------------------------------------------
//Function 13

int sum_calculations_daily (ROOT **root2) {

 NODE *node = (*root2)->head;

 int i;

 if(k<=11){

  for(i=0;i<((*root2)->num);i++) {
   ((PLANT*)node->data)->p_sum_accept_bid_p =  ((PLANT*)node->data)-
>p_sum_accept_bid_p + ((PLANT*)node->data)->accept_bid_p[k];
   node = node->next;
  }
 }
 else if(k>11) {
  for(i=0;i<((*root2)->num);i++) {
   ((PLANT*)node->data)->o_sum_accept_bid_p =  ((PLANT*)node->data)-
>o_sum_accept_bid_p + ((PLANT*)node->data)->accept_bid_p[k];
   node = node->next;
```

```
 }
 }

return 0;
}


//------------------------------------------------------------------------
//Function 14

int ave_calculations_daily (ROOT **root1, ROOT **root2) {

 NODE *node1 = (*root1)->head;
 NODE *node2 = (*root2)->head;

 int i,j;


  for(i=0;i<((*root2)->num);i++) {
   ((PLANT*)(node2->data))->p_ave_accept_bid_p          =          ((PLANT*)(node2->data))-
>p_sum_accept_bid_p/12.0;
   ((PLANT*)(node2->data))->o_ave_accept_bid_p          =          ((PLANT*)(node2->data))-
>o_sum_accept_bid_p/12.0;
   node2 = node2->next;
  }


  for(i=0;i<((*root1)->num);i++) {
   for(j=0;j<((AGENT*)(node1->data))->plant_num;j++){
    ((AGENT*)(node1->data))->sum_bid_p       =       ((AGENT*)(node1->data))->sum_bid_p
+((AGENT*)(node1->data))->plant[j]->bid_price;
    ((AGENT*)(node1->data))->p_sum_accept_bid_p       =       ((AGENT*)(node1->data))-
>p_sum_accept_bid_p +((AGENT*)(node1->data))->plant[j]->p_ave_accept_bid_p;
    ((AGENT*)(node1->data))->o_sum_accept_bid_p       =       ((AGENT*)(node1->data))-
>o_sum_accept_bid_p+((AGENT*)(node1->data))->plant[j]->o_ave_accept_bid_p;
   }
   node1 = node1->next;
  }


  node1 = (*root1)->head;

  for(i=0;i<((*root1)->num);i++) {
   ((AGENT*)(node1->data))->ave_bid_p       =       ((AGENT*)(node1->data))->sum_bid_p
/(((AGENT*)(node1->data))->plant_num);
   ((AGENT*)(node1->data))->p_ave_accept_bid_p          =          ((AGENT*)(node1->data))-
>p_sum_accept_bid_p /(((AGENT*)(node1->data))->plant_num);
   ((AGENT*)(node1->data))->o_ave_accept_bid_p          =          ((AGENT*)(node1->data))-
>o_sum_accept_bid_p /(((AGENT*)(node1->data))->plant_num);
   node1 = node1->next;
  }


  node1 = (*root1)->head;

  for(i=0;i<((*root1)->num);i++) {
   ((AGENT*)(node1->data))->sum_bid_p = 0.0;
```

```
   ((AGENT*)(node1->data))->p_sum_accept_bid_p = 0.0;
   ((AGENT*)(node1->data))->o_sum_accept_bid_p =0.0;
  node1 = node1->next;
  }


  node2 = (*root2)->head;

  for(i=0;i<((*root2)->num);i++) {
   ((PLANT*)node2->data)->p_sum_accept_bid_p = 0.0;
   ((PLANT*)node2->data)->o_sum_accept_bid_p = 0.0;
    node2 = node2->next;
  }


return 0;
}

//---------------------------------------------------------------------------
//Function 15

int agent_daily(ROOT **root){

 NODE *node = (*root)->head;
 ROOT *c1 = NULL ,* c2 = NULL ,* c3 = NULL;

 float m, w, cost;
 float total_cap_a, total_cap_b, accept_cap, accept_cap_total;
 float temp_price;
 int s,i,j;

 randomize();

 for(j=0;j<((*root)->num);j++){

  s1(prev_profit) = s1(profit);
  s1(profit)=0.0;
  total_cap_a=0.0;
  total_cap_b = 0.0;
  accept_cap=0.0;
  accept_cap_total = 0.0;

  for(i=0;i<(s1(plant_num));i++) {    //the calculation of the profit & UR

   for(k=0;k<24;k++){

    if(s2(bid_flag[k])==1) {

      cost=(s2(cost_coef.cca))*(s2(accept_bid_q[k]))*(s2(accept_bid_q[k])) +
          (s2(cost_coef.ccb))*(s2(accept_bid_q[k]))+ (s2(cost_coef.ccc));

      s1(profit)= (s1(profit))+ (s2(accept_bid_p[k]))*(s2(accept_bid_q[k]))- cost;
    }

    accept_cap= accept_cap + (s2(accept_bid_q[k]));
    total_cap_a = total_cap_a + s2(cap);
  }
```

```
 accept_cap_total = accept_cap_total + accept_cap;
 total_cap_b = total_cap_b + total_cap_a;
 s2(prev_bid_p)= s2(bid_price);
}


s1(ur)=(accept_cap_total)/(total_cap_b);

if((s1(ur))>=(s1(tur))){

 make_plant_list_cycle_daily(node,&c1,&c2,&c3);

 sort_plant_list_cycle_daily(&c1,&c2,&c3,compare_DailyPlantProfit);

 if(check_cycle_daily(&c1, &c2, &c3)==0)

  update_price_daily(&c1, &c2, &c3);

 else if(check_cycle_daily(&c1, &c2, &c3)==1){

  if(s1(profit) <= s1(prev_profit)){

    if(s1(gene[0])==0){

     s1(gene[0]) = 1;

          s=random(100);

          if(s>=49)
                        m =1.0+ (1.0*random(11)/100.0);
                  else if (s<49)
                m =1.0- (1.0*random(11)/100.0);


          for(i=0;i<(s1(plant_num));i++) {
                s2(bid_price) = (s2(prev_bid_p))*m;
     }
    }

    else if(s1(gene[0])==1) {

     if(s1(gene[1])==0){

      s1(gene[1])=1;
          m =1.0+ (1.0*random(11)/100.0);
     for(i=0;i<(s1(plant_num));i++) {
                s2(bid_price) = (s2(prev_bid_p))*m;
     }
    }


     else if(s1(gene[1])==1){
      s1(gene[1])=0;
          m =1.0- (1.0*random(11)/100.0);
     for(i=0;i<(s1(plant_num));i++) {
                s2(bid_price) = (s2(prev_bid_p))*m;
```

```
       }
      }

    }
   }
   else if(s1(profit)>s1(prev_profit)){

    for(i=0;i<(s1(plant_num));i++) {
                    s2(bid_price) = s2(prev_bid_p);
    }
    if(s1(gene[1])==0)
     s1(gene[1])=1;
    else
    if(s1(gene[1])==1)
     s1(gene[1])=0;
   }
  }
 }
 else if((s1(ur))<(s1(tur))){

  s1(gene[0])=0;
  s1(gene[1])=random(2);
  w=1.0-(1.0*random(11)/100.0);

  for(i=0;i< (s1(plant_num));i++){
   s2(bid_price) = (s2(prev_bid_p))*(w);
  }
 }

 //Required in order to eliminate negative profit
 for(i=0;i<(s1(plant_num));i++) {

  cost=(s2(cost_coef.cca))*(s2(bid_quantity))*(s2(bid_quantity)) +
       (s2(cost_coef.ccb))*(s2(bid_quantity))+ (s2(cost_coef.ccc));

  temp_price = cost/s2(bid_quantity);

   if((s2(bid_price))<temp_price || (s2(bid_price)) >1000.00){

    if((s2(bid_price))<temp_price)
     s2(bid_price) = temp_price;
    else
    s2(bid_price) = 1000.0;
   }
 }



 node = node->next;
}

return 0;
}
```

(daily_pr2.h)

```
/*MSc in Electrical Power Engineering
  Modelling of Electricity Markets using Software Agents

  Daily Bidding - 24 hours

  ---------------------------------------------
  DBL_LIST_PRI2.H HEADER FILE
  ---------------------------------------------

  This header file contains fundamental functions:
  A. For the manipulation of doubly linked lists
  B. To aid result manipulation and presentation
  C. For auxilliary tasks*/

//Macros
#define VALID(x) ((*x)!=NULL)
#define NUM(x) ((*x)->num)
#define NEW(x) (x*)malloc(sizeof(x))
#define sc1(x) ((((AGENT*)data)->plant[i])->x)
#define sc2(x) (((AGENT*)(node->data))->x)
#define sc3(x) ((((AGENT*)(node->data))->plant[i])->x)


//User Defined Type Definitions

//PLANT COST COEFFICIENTS
typedef struct cost_coefficients {
 float cca,ccb,ccc;
}CC;

//PLANT TYPE
typedef struct plant{
 void *pnt_agent;      //This pointer is set to point to the parent plant agent.
 int agent_id;         //Actual assignment can be found in agent_read_data
 int id;
 int cycle;
 float cap, mc;
 float bid_quantity, bid_price;
 float accept_bid_q[24], accept_bid_p[24];
 float prev_bid_q, prev_bid_p;
 int bid_flag[24];
 float p_sum_accept_bid_p;
 float p_ave_accept_bid_p;
 float o_sum_accept_bid_p;
 float o_ave_accept_bid_p;
 float plant_profit[24];
 float daily_plant_profit;
 CC cost_coef;
}PLANT;

//AGENT DATA TYPE
typedef struct agent{
 PLANT *plant[BUFSIZ];
 int agent_id;
 int plant_num;
```

```
 int gene[2];
 float tur;
 float ur;
 float prev_ur;
 float profit;
 float prev_profit;
 float sum_bid_p, p_sum_accept_bid_p;
 float ave_bid_p, p_ave_accept_bid_p;
 float o_sum_accept_bid_p;
 float o_ave_accept_bid_p;
}AGENT;

//TYPE DEFINTITIONS FOR DBL LINKED LISTS

/*NODE OF A DBL LINKED LIST*/
typedef struct node {
  void  *data;
  struct node *next;
  struct node *prev;
}NODE;

//ROOT OF A DBL LINKED LIST
typedef struct {
   long num;
   NODE *head;
   NODE *tail;
}ROOT;


/*-------------------------------------------------------------
  FUNCTIONS RELATED TO THE MANIPULATION OF DOUBLY LINKED LISTS
  -------------------------------------------------------------*/

//1.Function to make root of a doubly linked list - Required by F3

ROOT *make_root (void) {

 ROOT *root;

  if((root=NEW(ROOT))!=NULL) {
   root->head = root->tail=NULL;
   root->num = 0;
  }

return root;
}

//2.Function to make node of a doubly linked list - Required by F3

NODE *dbl_make_node (void *data){

 NODE *node;

  if((node = NEW(NODE))!=NULL) {
   node->data = data;
   node->next = NULL;
   node->prev = NULL;
```

```
  }

return node;
}

//3.Function to insert data in a doubly linked list

int dbl_insert_data (ROOT **root, void *data, int position) {

 NODE *neu;

 if(!VALID(root))
  if((*root = make_root())==NULL)
   return(-1);

 if((neu = dbl_make_node (data)) == NULL){
  if(NUM(root)==0){
   free(*root);
   *root=NULL;
  }
  return (-1);
 }

 if(NUM(root)==0)
  (*root)->tail=(*root)->head= neu;

 else {

  if(position == 0) {
   neu->next = (*root)->head;
   (*root)->head->prev = neu;
   (*root)->head = neu;
  }

  else{
   (*root)->tail->next = neu;
   neu->prev = (*root)->tail;
   (*root)->tail = neu;
  }

 }

 (NUM(root))++;

return 0;
}


//4.Function to reset a doubly linked list

int reset_list (ROOT **root) {

 NODE *thes, *next;

  if(!VALID(root))
   return (-1);
```

```
  thes = (*root)->head;

  do {
   next = thes->next;
   free(thes);
   thes = next;
  }while (thes!=NULL);


  free(*root);
  *root = NULL;

return 0;
}


/*-------------------------------------------------------------
  FUNCTIONS RELATED TO OUTPUT FILE GENERATION
  -------------------------------------------------------------*/

//5. Function to display data in a doubly linked list

void display_list (ROOT **root, void (*disp) (void *data, FILE *), FILE *f1) {

 NODE *node;

 if(VALID(root)) {
  node = (*root)->head;

  do {
   disp(node->data, f1);
   node = node->next;
  } while (node!=NULL);
 }

 fputc('\n',f1);
}

//5b. Function to display data in a doubly linked list

void display_list2(ROOT **root, void (*disp) (void *data, FILE *),FILE *f) {

NODE *node;
int i;

 if(VALID(root)) {
  node = (*root)->head;

  for(i=0;i<(*root)->num;i++){
   disp(node->data, f+(i+1));
   node = node->next;
  }

  putchar('\n');
 }
}
```

```
//6.

void display_AGENTb (void *data, FILE *filename){

int i;
float capacity=0.0;

 for(i=0;i<((AGENT*)data)->plant_num;i++){
  fprintf(filename,"%8.2f\t",((AGENT*)data)->plant[i]->bid_price);
  fprintf(filename,"%8.2f\t",((AGENT*)data)->plant[i]->o_ave_accept_bid_p);
  fprintf(filename,"%8.2f\t",((AGENT*)data)->plant[i]->bid_price);
  fprintf(filename,"%8.2f\t",((AGENT*)data)->plant[i]->p_ave_accept_bid_p);
 }

 fprintf(filename,"%8.2f\t",((AGENT*)data)->ave_bid_p);
 fprintf(filename,"%8.2f\t",((AGENT*)data)->o_ave_accept_bid_p);
 fprintf(filename,"%8.2f\t",((AGENT*)data)->ave_bid_p);
 fprintf(filename,"%8.2f\t",((AGENT*)data)->p_ave_accept_bid_p);
 fprintf(filename,"%8.2f\t",((AGENT*)data)->profit);

 for(i=0;i<((AGENT*)(data))->plant_num;i++)
  capacity = capacity+((AGENT*)(data))->plant[i]->cap;

 fprintf(filename,"%8.2f\t",(((AGENT*)data)->profit)/capacity);

 fprintf(filename,"%8.2f\t",((AGENT*)data)->ur);

 fprintf(filename,"\n");

}

//7.

void display_titles (void *data, FILE *filename){

 int i;

 fprintf(filename,"Agent[%d]\n",((AGENT*)data)->agent_id);

 fprintf(filename,"\n");

 for(i=0;i<((AGENT*)data)->plant_num;i++){
  fprintf(filename,"Plant[%d]\tCycle:%d                \t\t\t",i,((AGENT*)data)->plant[i]->cycle);
 }

 fprintf(filename,"AGENT SUMMARY                    \t\t\t\t");

 fprintf(filename,"\n");

 for(i=0;i<((AGENT*)data)->plant_num+1;i++){
  fprintf(filename,"OFF-PEAK        \t\tPEAK            \t\t",i);
 }

 fprintf(filename,"\n");

 for(i=0;i<2*((((AGENT*)data)->plant_num)+1);i++){
  fprintf(filename,"BID     \tABP     \t");
```

```
}

 fprintf(filename,"Profit  \tProf_PIC\tUR      \t");

 fprintf(filename,"\n");

}

/*
//8. Function to display titles
void display_titles (void *data, FILE *filename){

 int i;

 fprintf(filename,"Agent[%d]\n",((AGENT*)data)->agent_id);

 for(i=0;i<((AGENT*)data)->plant_num;i++){
  fprintf(filename,"Plant[%d]            \t\t",i);
 }

 fprintf(filename,"\n");

 for(i=0;i<((AGENT*)data)->plant_num;i++){
 // fprintf(filename,"Bid Price \tAcc B Pric\t");
  fprintf(filename,"Bid Price \t");
 }

 fprintf(filename,"\n");

}
*/

/*
//9. Function to display bid price and accepted bid price in column format

void display_AGENTb (void *data, FILE *filename){

 int i;

 for(i=0;i<((AGENT*)data)->plant_num;i++){
  fprintf(filename,"%10.2f\t",((AGENT*)data)->plant[i]->bid_price);
  //fprintf(filename,"%10.2f\t",((AGENT*)data)->plant[i]->accept_bid_p);
 }

 fprintf(filename,"\n");
}

//10.

void display_AGENTc (void *data, FILE *filename){

 int i,j;


 for(i=0;i<((AGENT*)data)->plant_num;i++){
  fprintf(filename,"%10.2f\t",((AGENT*)data)->plant[i]->bid_price);
  fprintf(filename,"%10.2f\t",((AGENT*)data)->plant[i]->bid_quantity);
```

```
 }

 fprintf(filename,"\t");


 for(i=0;i<((AGENT*)data)->plant_num;i++){

  for(j=0;j<24;j++)
   fprintf(filename,"%10.2f\t",((AGENT*)data)->plant[i]->accept_bid_p[j]);

  fprintf(filename,"\t");
 }


 fprintf(filename,"\n");

//    for(i=0;i<((AGENT*)data)->plant_num;i++)
//   for(j=0;j<24;j++){
//    fprintf(filename,"%7.2f\t",((AGENT*)data)->plant[i]->accept_bid_p[j]);
//    fprintf(filename,"\n");
//   }

 //fprintf(filename,"%f\n",((AGENT*)data)->profit);
 //fprintf(filename,"%f\n",((AGENT*)data)->ur);
 //fprintf(filename,"%f\n",((AGENT*)data)->plant[0]->accept_bid_p);

}
*/

//11.Function to display plant structure contents

void display_PLANT(void *data, FILE *f1){


  fprintf(f1,"Agent[%d],Plant[%d]->Bid Price=%f,Bid Quantity=%f\n"
             "Accepted Bid Price=%f,Accepted Bid Quantity=%f,Bid_Flag=%d\n"
         ,((PLANT*)data)->agent_id,((PLANT*)data)->id
         ,((PLANT*)data)->bid_price,((PLANT*)data)->bid_quantity
         ,((PLANT*)data)->accept_bid_p,((PLANT*)data)->accept_bid_q
         ,((PLANT*)data)->bid_flag);
  fprintf(f1,"----------------------------------------\n");

}

//12.
void display_PLANTb(void *data, FILE *f1){


  fprintf(f1,"Agent[%d],Plant[%d]->BQ: %f, BP: %f, ABQ: %u, ABP: %u\n"
         ,((PLANT*)data)->agent_id,((PLANT*)data)->id
         ,((PLANT*)data)->bid_quantity,((PLANT*)data)->bid_price
         ,((PLANT*)data)->accept_bid_q
         ,((PLANT*)data)->accept_bid_p
         );
}

/*-------------------------------------------------------
```

```
  COMPARE FUNCTIONS
  ------------------------------------------------------------*/

//13. Function to compare two prices - tree

float compare_BIDPRICE (void *d1, void *d2) {
 return ((((PLANT*)d2)->bid_price)-(((PLANT*)d1)->bid_price));
}

//14.
float compare_DailyPlantProfit (void *d1, void *d2) {
 return ((((PLANT*)d2)->daily_plant_profit)-(((PLANT*)d1)->daily_plant_profit));
}


/*------------------------------------------------------------
  SAVING AND RECOVERING DOUBLY LINKED LISTS FROM A BINARY FILE
  ------------------------------------------------------------*/

//15. Function to save a doubly linked list to a file

int save_list(char *filename, ROOT **root, size_t datasize) {

 FILE *f1;
 NODE *node;

 if(VALID(root)) {
  if((f1=fopen(filename,"wb"))==NULL)  /*Open and write to the file in binary mode*/
   return (-1);

  node = (*root)->head;

  do{
   if(fwrite(node->data,datasize,1,f1)!=1)
   return(-1);
   node=node->next;
  } while (node!=NULL);

  return(fclose(f1));

 }

return 0;
}

//16. Function to recover a double linked list from a file
int recover_list (char *filename, ROOT **root, size_t datasize) {

 FILE *f1;
 void *dptr;

 if((f1=fopen(filename, "rb"))==NULL)
  return(-4);

 do{
  if((dptr = malloc(datasize))==NULL)
  return (-1);
```

```
  if(fread(dptr,datasize,1,f1)!=1) {
  if(feof(f1))
   return 0;
   else {
    free(dptr);
    return (-2);
   }
  }

  if(dbl_insert_data(root,dptr,1)!=0)
   return (-3);

 } while (1);

}
```

(hourly.cpp)

```
/*MSc in Electrical Power Engineering
  Modelling of Electricity Markets using Software Agents
  Hourly Bidding - 24 hours*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <time.h>

//Global Variables
int k, iters;

#include "hourly_prg2.h"

//Macros
#define MAIN int agvc, char *agv[]
#define s1(x)  (((AGENT*)(node->data))->x)
#define s2(x)  ((((AGENT*)(node->data))->plant[i])->x)


//Function Prototypes

/*1. Function used to read initialising data for the
     AGENT/PLANT variables from an input stream e.g. plant.dat*/
     ROOT *read_agent_data (ROOT **, char *);

/*2  Function used to read a 24-hourly demand profile from an input stream*/
     int read_demand_profile (float *demand, char *filename);
```

```
/*3. Function used to construct a doubly linked list with nodes that point
     at PLANT type structures.*/
     ROOT *make_plant_list(ROOT **root1,ROOT **root2);

/*4. Function used to constuct a doubly linked list whose nodes point at
     plants of the same cycle*/
     int make_plant_list_cycle(NODE *node,ROOT **c1,ROOT **c2,ROOT **c3);

/*5. Function used to sort the plants according to their bid price
            and, therefore, construct the aggregate supply function*/
     int sort_plant_list (ROOT **root2, float (*cmp)(void*, void*));

/*6. Function used to sort the plant cycle linked-list*/
     int sort_plant_list_cycle (ROOT **c1, ROOT **c2,ROOT **c3,float(*cmp)(void*, void*));

/*7. Function used to clear the market according to Price Bid.
     Bid_flag set and accepted price/quantity determined for each plant*/
     int market_clearing (ROOT **root2, float demand);

/*8. Function used to clear the market according to SMP
     Bid_flag set and accepted price/quantity determined for each plant*/

     int market_clearing_smp (ROOT **root2, float demand);

/*9. This function is used to update the price of plants that have been
     previously sorted by function 6 */
     int update_price(ROOT **c1, ROOT **c2, ROOT **c3);

/*10.This function is used to check if all the plants in the plant cycle
     linked-list have the same price*/
     int check_list_cycle (ROOT **c1, ROOT **c2, ROOT **c3);

/*11.This function is used to aid calculations carried out by function 12*/
     int sum_calculations(ROOT **root2);

/*12.Function used to calculate the average bid and accepted bid price for
     each plant*/
     int ave_calculations (ROOT **root1, ROOT **root2);

/*13.Agent Function*/
     int agent(ROOT **root);

//---------------------------------------------------------------------------
//MAIN PROGRAM

int main (MAIN){


 ROOT *root1=NULL, *root2=NULL;
 FILE *f[7];
 int i;
 float demand[24];                  //Array for demand values temporary storage

 if(agvc!=3){
   printf("Valid Calling Format-> program_name <agent_data_file>"
                                  "<demand_data_file>\n");
   exit(0);
```

```
}

//Open primary output streams

if((f[0]=fopen("out.dat", "a"))==NULL){
 perror("Cannot open file out.dat");
 exit(0);
}

if((f[1]=fopen("out1.dat", "a"))==NULL){
 perror("Cannot open file out1.dat");
 exit(0);
}

if((f[2]=fopen("out2.dat", "a"))==NULL){
 perror("Cannot open file out2.dat");
 exit(0);
}

if((f[3]=fopen("out3.dat", "a"))==NULL){
 perror("Cannot open file out3.dat");
 exit(0);
}

if((f[4]=fopen("out4.dat", "a"))==NULL){
 perror("Cannot open file out4.dat");
 exit(0);
}

if((f[5]=fopen("out5.dat", "a"))==NULL){
 perror("Cannot open file out5.dat");
 exit(0);
}

if((f[6]=fopen("out6.dat", "a"))==NULL){
 perror("Cannot open file out6.dat");
 exit(0);
}

/*Initialise the data segment of the AGENT doubly linked list with values from
  the input file plant.dat*/

if((root1=read_agent_data(&root1,agv[1]))==NULL){
  printf("Could not create agent list...\n");
  exit(EXIT_FAILURE);
}

display_list2(&root1,display_titles,*f);

//Read the demand profile for all 24 hours from the input file demand.dat

if(read_demand_profile(demand,agv[2])!=0){
 printf("Could not create demand profile list...\n");
 exit(EXIT_FAILURE);
}

//The user is promped to enter the number of iterations
```

```
 printf("Please enter number of iterations:");
 scanf("%d",&iters);


 for(i=0;i<iters;i++){

  for (k=0;k<24;k++){

   if((root2= make_plant_list(&root1,&root2))==NULL){
    printf("Could not create plant list...\n");
    exit(EXIT_FAILURE);
   }

   if(sort_plant_list(&root2,compare_BIDPRICE)!=0){
    printf("Could not sort plant list...");
    exit(EXIT_FAILURE);
   }

   if(market_clearing(&root2,demand[k])!=0){
   //if(market_clearing_smp(&root2,demand[k])!=0){
    printf("Could not clear the market...");
    exit(EXIT_FAILURE);
   }

   if(sum_calculations(&root2)!=0){
    printf("Could not calculate average quantities...");
    exit(EXIT_FAILURE);
   }

//  display_list(&root2, display_PLANTb, f[0]);
//  display_list2(&root1,display_AGENTb,*f);

   if(agent(&root1)!=0)
    printf("Error with the agent...");
  }
  ave_calculations(&root1, &root2);

  if((i%10)==0){
   printf("%d\n", i);
   display_list2(&root1,display_AGENTb,*f);
  }

 }

 for(i=0;i<7;i++){
  fclose(f[i]);
  if(ferror(f[i]))
   perror("Error with file out.dat...");
 }

return 0;
}

//---------------------------------------------------------------------------
//Function 1
```

```
ROOT *read_agent_data (ROOT **root, char *filename) {

 AGENT *data;
 NODE *node;
 int plant_flag=1,agent_flag=1, i=0, j=0;

 FILE *f1;

 randomize();

 if((f1=fopen(filename, "r"))==NULL){
  perror("Cannot Open File...");
  exit(EXIT_FAILURE);
 }

 while(agent_flag==1){

  data=NEW(AGENT);

   while(plant_flag ==1){

    (data->plant[i])=NEW(PLANT);

    fscanf(f1,"%f %f",&((data->plant[i])->cap),&((data->plant[i])->mc));
    fscanf(f1,"%f %f %f %d %d\n",&(data->plant[i]->cost_coef.cca)
            ,&(data->plant[i]->cost_coef.ccb),&(data->plant[i]->cost_coef.ccc)
            ,&(data->plant[i]->cycle),&plant_flag);

    data->agent_id = (data->plant[i])->agent_id=j;
    (data->plant[i])->id =i;

    for(k=0;k<24;k++){
     (data->plant[i])->bid_quantity[k]=(data->plant[i])->cap;
     (data->plant[i])->bid_price[k] = (data->plant[i])->mc;
     (data->plant[i])->accept_bid_q[k]=0.0;
     (data->plant[i])->accept_bid_p[k]=0.0;
     (data->plant[i])->prev_bid_q[k]=0.0;
     (data->plant[i])->prev_bid_p[k]=0.0;
     (data->plant[i])->bid_flag[k]=0;
     data->profit[k] =data->prev_profit[k]=0.0;
     data->ur[k]=data->prev_ur[k]=0.0;
     data->gene[k].g[0] = 0;
     data->gene[k].g[1] = random(2);
    }
    data->plant_num = ++i;
   }

   fscanf(f1,"%f %d\n",&data->tur, &agent_flag);

   if(dbl_insert_data(root,data,1)!=0){
    printf("Unable to add agent in the list");
    exit(EXIT_FAILURE);
   }

   plant_flag=1;
   ++j;
   i=0;
```

```
 }

 node = (*root)->head;

 for(j=0;j<(*root)->num;j++){
  for(i=0;i<((AGENT*)(node->data))->plant_num;i++){
   ((((AGENT*)(node->data))->plant[i])->pnt_agent) = node;
  }
  node= node->next;
 }

 if(ferror(f1))
  perror("Error Reading Source File...");

 fclose(f1);

return(*root);
}
//---------------------------------------------------------------------------
//Function 2

int read_demand_profile (float *demand, char *filename) {

 int i;
 FILE *f2;

 if((f2=fopen(filename, "r"))==NULL){
  perror("Cannot Open File demand.dat ...");
  exit(EXIT_FAILURE);
 }

 for(i=0;i<24;i++){
  fscanf(f2,"%f\n",(demand+i));
 }

 if(i<23)
  printf("\nThere are no 24 demand values in the demand file...\n"
         "Please check and re-run\n");

 if(ferror(f2))
  perror("Error Reading Source File...");

 fclose(f2);

return 0;
}

//---------------------------------------------------------------------------
//Function 3

ROOT *make_plant_list(ROOT **root1,ROOT **root2){

 NODE *node = (*root1)->head;
 int i,j;

 reset_list(root2);
```

```
 if((*root1)!=NULL){
  for(j=0;j<(*root1)->num;j++){
   for(i=0;i<sc2(plant_num);i++){
    dbl_insert_data(root2, (((AGENT*)(node->data))->plant[i]), 1);
   }
   node=node->next;
  }
 }

return (*root2);
}

//---------------------------------------------------------------------------
//Function 4

int sort_plant_list (ROOT **root2, float (*cmp)(void*, void*)){

 NODE *node= (*root2)->head;
 NODE *node_next = node->next;
 void *temp;
 int i,j;

 for(i=0; i<((*root2)->num); i++){

  for(j=1;j<((*root2)->num);j++){

   if(cmp((node_next->data),(node->data))<0.0){

    temp = node->data;
    (node->data) = (node_next->data);
    (node_next->data) = temp;
   }
   node_next = node_next->next;
  }
 node = node->next;
 node_next=(*root2)->head;
 }

return 0;
}

//---------------------------------------------------------------------------
//Function 5

int market_clearing (ROOT **root2, float demand){

 NODE *node = (*root2)->head;
 int i;
 float total_cap=0.0;

 if(!VALID(root2))
  return (-1);

 if(demand<=0.0){
  printf("Demand should be positive integer...\n");
  return(-1);
 }
```

```
 for(i=0;i<(*root2)->num;i++){
  ((PLANT*)node->data)->accept_bid_q[k]=0.0;
  ((PLANT*)node->data)->accept_bid_p[k]=0.0;
  ((PLANT*)node->data)->bid_flag[k]=0;
  total_cap = total_cap + ((PLANT*)node->data)->bid_quantity[k];
  node = node->next;
 }

 if(total_cap<demand){
  printf("There is not enough capacity to satisfy the load requirements\n");
  return(-1);
 }

 node = (*root2)->head;

 for(i=0;i<(*root2)->num;i++){

  ((PLANT*)node->data)->accept_bid_q[k]=((PLANT*)node->data)->bid_quantity[k];
  ((PLANT*)node->data)->accept_bid_p[k]=((PLANT*)node->data)->bid_price[k];
  demand = demand -((PLANT*)node->data)->accept_bid_q[k];

  if(demand>0.0){

   ((PLANT*)node->data)->bid_flag[k] = 1;
   node=node->next;
  }

  else if(demand<0.0) {

   ((PLANT*)node->data)->accept_bid_q[k]          =          ((PLANT*)node->data)-
>accept_bid_q[k]+demand;
   ((PLANT*)node->data)->bid_flag[k] = 1;
   break;
  }

  else {
   ((PLANT*)node->data)->bid_flag[k] = 1;
   break;
  }
 }

return 0;
}

//---------------------------------------------------------------------------
//Function 6

int market_clearing_smp (ROOT **root2, float demand){

 NODE *node = (*root2)->head;
 int i;
 float total_cap=0.0;
 float temp_price;

 if(!VALID(root2))
  return (-1);
```

```
if(demand<=0.0){
 printf("Demand should be positive integer...\n");
 return(-1);
}

for(i=0;i<(*root2)->num;i++){
 ((PLANT*)node->data)->accept_bid_q[k]=0.0;
 ((PLANT*)node->data)->accept_bid_p[k]=0.0;
 ((PLANT*)node->data)->bid_flag[k]=0;
 total_cap = total_cap + ((PLANT*)node->data)->bid_quantity[k];
 node = node->next;
}


if(total_cap<demand){
 printf("There is not enough capacity to satisfy the load requirements\n");
 return(-1);
}

node = (*root2)->head;

for(i=0;i<(*root2)->num;i++){

 ((PLANT*)node->data)->accept_bid_q[k]=((PLANT*)node->data)->bid_quantity[k];
 temp_price = ((PLANT*)node->data)->bid_price[k];
 //((PLANT*)node->data)->accept_bid_p=((PLANT*)node->data)->bid_price;
 demand = demand -((PLANT*)node->data)->accept_bid_q[k];

 if(demand>0.0){

  ((PLANT*)node->data)->bid_flag[k] = 1;
  node=node->next;
 }

 else if(demand<0.0) {

  ((PLANT*)node->data)->accept_bid_q[k]                =                ((PLANT*)node->data)-
>accept_bid_q[k]+demand;
  temp_price = ((PLANT*)node->data)->bid_price[k];
  ((PLANT*)node->data)->bid_flag[k] = 1;
  break;
 }

 else {
  ((PLANT*)node->data)->bid_flag[k] = 1;
   temp_price = ((PLANT*)node->data)->bid_price[k];
  break;
 }
}

node = (*root2)->head;

for(i=0;i<(*root2)->num;i++) {

 if (((PLANT*)node->data)->bid_flag[k] == 1)
```

```
   ((PLANT*)node->data)->accept_bid_p[k] = temp_price;
  node = node->next;
 }

 return 0;
}

//---------------------------------------------------------------------------
//Function 7

int make_plant_list_cycle(NODE *node,ROOT **c1,ROOT **c2,ROOT **c3){

 int i;

 reset_list(c1);
 reset_list(c2);
 reset_list(c3);

 if((node->data)!=NULL ){

   for(i=0;i<sc2(plant_num);i++){

     if(sc3(bid_flag[k])==1) {

      switch(sc3(cycle)) {

       case 1: dbl_insert_data(c1, (((AGENT*)(node->data))->plant[i]), 1);
               break;

       case 2: dbl_insert_data(c2, (((AGENT*)(node->data))->plant[i]), 1);
               break;

       case 3: dbl_insert_data(c3, (((AGENT*)(node->data))->plant[i]), 1);
               break;
      }
     }
   }
 }



 return 0;
}

//---------------------------------------------------------------------------
//Function 8

int sort_plant_list_cycle (ROOT **c1, ROOT **c2,ROOT **c3,float(*cmp)(void*, void*)){

 NODE *node1;
 NODE *node2;
 NODE *node3;
 NODE *node_next1;
 NODE *node_next2;
 NODE *node_next3;
 void *temp1, *temp2,*temp3;
```

```
 int i,j;

 if(*c1!=NULL){

  node1= (*c1)->head;
  node_next1 = node1->next;

  for(i=0; i<((*c1)->num); i++){
    for(j=1;j<((*c1)->num);j++){
      if(cmp((node_next1->data),(node1->data))<0.0){
       temp1 = node1->data;
       (node1->data) = (node_next1->data);
       (node_next1->data) = temp1;
      }
     node_next1 = node_next1->next;
    }
   node1 = node1->next;
   node_next1=(*c1)->head;
  }

/*  node1 = (*c1)->head;

  for(i=0;i<(*c1)->num;i++) {
   printf("%f\n", ((PLANT*)(node1->data))->bid_price[k]);
   node1 = node1->next;
  }*/
 }

//---

if(*c2!=NULL){

 node2= (*c2)->head;
 node_next2 = node2->next;

 for(i=0; i<((*c2)->num); i++){

  for(j=1;j<((*c2)->num);j++){

   if(cmp((node_next2->data),(node2->data))<0.0){

     temp2 = node2->data;
     (node2->data) = (node_next2->data);
     (node_next2->data) = temp2;
   }
  node_next2 = node_next2->next;
  }
 node2 = node2->next;
 node_next2=(*c2)->head;
 }
/* node2 = (*c2)->head;

  for(i=0;i<(*c2)->num;i++) {
   printf("%f\n", ((PLANT*)(node2->data))->bid_price[k]);
   node2 = node2->next;
  }*/
```

```
}
//--

if(*c3!=NULL){

 node3= (*c3)->head;
 node_next3 = node3->next;

 for(i=0; i<((*c3)->num); i++){

  for(j=1;j<((*c3)->num);j++){

   if(cmp((node_next3->data),(node3->data))<0.0){

     temp3 = node3->data;
     (node3->data) = (node_next3->data);
     (node_next3->data) = temp3;
   }
  node_next3 = node_next3->next;
  }
 node3 = node3->next;
 node_next3=(*c3)->head;
 }
/*
 node3 = (*c3)->head;

  for(i=0;i<(*c3)->num;i++) {
   printf("%f\n", ((PLANT*)(node3->data))->bid_price[k]);
   node3 = node3->next;
  }*/
 }


 return 0;
}

//---------------------------------------------------------------------------
//Function 9

int check_list_cycle(ROOT **c1, ROOT **c2,ROOT **c3) {

 NODE *node1, *node2, *node3;
 NODE *node_next1, *node_next2, *node_next3;
 int a=1,b=1,c=1, flag;
 int i;


 if(VALID(c1)){

  if(NUM(c1)>=2){

   node1= (*c1)->head;
   node_next1 = node1->next;

  for (i=1;i<NUM(c1);i++){
```

```
   if(       ((PLANT*)(node_next1->data))->bid_price[k]       ==       ((PLANT*)(node1->data))-
>bid_price[k]){
     node_next1 = node_next1->next;
     a = 1;
   }

   else{
    a = 0;
    break;
   }
  }
 }
}

 //---


 if(VALID(c2)){

  if(NUM(c2)>=2){

   node2= (*c2)->head;
   node_next2 = node2->next;

   for (i=1;i<NUM(c2);i++){

   if(       ((PLANT*)(node_next2->data))->bid_price[k]       ==       ((PLANT*)(node2->data))-
>bid_price[k]){
     node_next2 = node_next2->next;
     b = 1;
   }

   else{
    b = 0;
    break;
   }
  }
 }
}
 //--

if(VALID(c3)){

  if(NUM(c3)>=2){

   node3= (*c3)->head;
   node_next3 = node3->next;

   for (i=1;i<NUM(c3);i++){

   if(       ((PLANT*)(node_next3->data))->bid_price       [k]==       ((PLANT*)(node3->data))-
>bid_price[k]){
     node_next3 = node_next3->next;
     c = 1;
   }

   else{
```

```
     c = 0;
     break;
    }
   }
  }
}


 if((a==1) && (b==1) && (c==1))
  flag = 1;
 else
  flag =0;

return flag;
}

//---------------------------------------------------------------------------
//Function 10

int update_price(ROOT **c1, ROOT **c2,ROOT **c3) {

 NODE *node1, *node2, *node3;
 NODE *node_next1, *node_next2, *node_next3;


 int i;


 if(VALID(c1)){

  if(NUM(c1)>=2){

   node1= (*c1)->head;
   node_next1 = node1->next;

   for (i=2;i<NUM(c1);i++){

    if( ((PLANT*)(node_next1->data))->bid_price[k] == ((PLANT*)(node1->data))->bid_price[k])
      node_next1 = node_next1->next;
    else

    break;
   }

  while(((PLANT*)(node1->data))->bid_price[k]          !=          ((PLANT*)(node_next1->data))-
>bid_price[k]){
    ((PLANT*)(node1->data))->bid_price[k] = ((PLANT*)(node_next1->data))->bid_price[k];
    node1 = node1->next;
   }
  }
 }


 //---

 if(VALID(c2)){
```

```
  if(NUM(c2)>=2){

   node2= (*c2)->head;
   node_next2 = node2->next;

   for (i=2;i<NUM(c2);i++){

    if( ((PLANT*)(node_next2->data))->bid_price[k] == ((PLANT*)(node2->data))->bid_price[k])
      node_next2 = node_next2->next;
    else

    break;
   }

   while(((PLANT*)(node2->data))->bid_price[k]          !=          ((PLANT*)(node_next2->data))-
>bid_price[k]){
    ((PLANT*)(node2->data))->bid_price[k] = ((PLANT*)(node_next2->data))->bid_price[k];
    node2 = node2->next;
   }
  }
 }

//--

 if(VALID(c3)){

  if(NUM(c3)>=2){

   node3= (*c3)->head;
   node_next3 = node3->next;

   for (i=2;i<NUM(c3);i++){

    if( ((PLANT*)(node_next3->data))->bid_price[k] == ((PLANT*)(node3->data))->bid_price[k])
      node_next3 = node_next3->next;
    else

    break;
   }

   while(((PLANT*)(node3->data))->bid_price[k]          !=          ((PLANT*)(node_next3->data))-
>bid_price[k]){
    ((PLANT*)(node3->data))->bid_price[k] = ((PLANT*)(node_next3->data))->bid_price[k];
    node3 = node3->next;
   }
  }
 }


 return 0;
}

//-------------------------------------------------------------------------
//Function 11

int sum_calculations (ROOT **root2) {
```

```
 NODE *node = (*root2)->head;

 int i;

 if(k<=11){

  for(i=0;i<((*root2)->num);i++) {
   ((PLANT*)node->data)->p_sum_bid_p = ((PLANT*)node->data)->p_sum_bid_p +
((PLANT*)node->data)->bid_price[k];
   ((PLANT*)node->data)->p_sum_accept_bid_p = ((PLANT*)node->data)-
>p_sum_accept_bid_p + ((PLANT*)node->data)->accept_bid_p[k];
   node = node->next;
  }
 }
 else if(k>11) {
  for(i=0;i<((*root2)->num);i++) {
   ((PLANT*)node->data)->o_sum_bid_p = ((PLANT*)node->data)->o_sum_bid_p +
((PLANT*)node->data)->bid_price[k];
   ((PLANT*)node->data)->o_sum_accept_bid_p = ((PLANT*)node->data)-
>o_sum_accept_bid_p + ((PLANT*)node->data)->accept_bid_p[k];
   node = node->next;
  }
 }

return 0;
}


//------------------------------------------------------------------------
//Function 12

int ave_calculations (ROOT **root1, ROOT **root2) {

 NODE *node1 = (*root1)->head;
 NODE *node2 = (*root2)->head;

 int i,j;


  for(i=0;i<((*root1)->num);i++) {

   ((AGENT*)(node1->data))->col_profit = 0.0;
   ((AGENT*)(node1->data))->col_ur = 0.0;
   node1 = node1->next;
  }


  for(i=0;i<((*root2)->num);i++) {
   ((PLANT*)(node2->data))->p_ave_bid_p = ((PLANT*)(node2->data))->p_sum_bid_p / 12.0;
   ((PLANT*)(node2->data))->p_ave_accept_bid_p = ((PLANT*)(node2->data))-
>p_sum_accept_bid_p/12.0;
   ((PLANT*)(node2->data))->o_ave_bid_p = ((PLANT*)(node2->data))->o_sum_bid_p / 12.0;
   ((PLANT*)(node2->data))->o_ave_accept_bid_p = ((PLANT*)(node2->data))-
>o_sum_accept_bid_p/12.0;

    node2 = node2->next;
  }
```

```
 node1 = (*root1)->head;

 for(i=0;i<((*root1)->num);i++) {
  for(j=0;j<((AGENT*)(node1->data))->plant_num;j++){
    ((AGENT*)(node1->data))->p_sum_bid_p = ((AGENT*)(node1->data))-
>p_sum_bid_p+((AGENT*)(node1->data))->plant[j]->p_ave_bid_p;
    ((AGENT*)(node1->data))->p_sum_accept_bid_p = ((AGENT*)(node1->data))-
>p_sum_accept_bid_p +((AGENT*)(node1->data))->plant[j]->p_ave_accept_bid_p;
    ((AGENT*)(node1->data))->o_sum_bid_p = ((AGENT*)(node1->data))->o_sum_bid_p
+((AGENT*)(node1->data))->plant[j]->o_ave_bid_p;
    ((AGENT*)(node1->data))->o_sum_accept_bid_p = ((AGENT*)(node1->data))-
>o_sum_accept_bid_p+((AGENT*)(node1->data))->plant[j]->o_ave_accept_bid_p;
  }
  node1 = node1->next;
 }


 node1 = (*root1)->head;

 for(i=0;i<((*root1)->num);i++) {
  ((AGENT*)(node1->data))->p_ave_bid_p = ((AGENT*)(node1->data))->p_sum_bid_p
/(((AGENT*)(node1->data))->plant_num);
  ((AGENT*)(node1->data))->p_ave_accept_bid_p = ((AGENT*)(node1->data))-
>p_sum_accept_bid_p /(((AGENT*)(node1->data))->plant_num);
  ((AGENT*)(node1->data))->o_ave_bid_p = ((AGENT*)(node1->data))->o_sum_bid_p
/(((AGENT*)(node1->data))->plant_num);
  ((AGENT*)(node1->data))->o_ave_accept_bid_p = ((AGENT*)(node1->data))-
>o_sum_accept_bid_p /(((AGENT*)(node1->data))->plant_num);

  for(j=0;j<24;j++){
   ((AGENT*)(node1->data))->col_profit = ((AGENT*)(node1->data))->col_profit +
((AGENT*)(node1->data))->profit[j];
   ((AGENT*)(node1->data))->col_ur = ((AGENT*)(node1->data))->col_ur +
((AGENT*)(node1->data))->ur[j];
  }
  ((AGENT*)(node1->data))->col_ur = ((AGENT*)(node1->data))->col_ur /24.0;

  node1 = node1->next;
 }


 node1 = (*root1)->head;

 for(i=0;i<((*root1)->num);i++) {

  ((AGENT*)(node1->data))->p_sum_bid_p = 0.0;
  ((AGENT*)(node1->data))->p_sum_accept_bid_p = 0.0;
  ((AGENT*)(node1->data))->o_sum_bid_p = 0.0;
  ((AGENT*)(node1->data))->o_sum_accept_bid_p =0.0;
  node1 = node1->next;
 }


 node2 = (*root2)->head;
```

```
  for(i=0;i<((*root2)->num);i++) {
   ((PLANT*)node2->data)->p_sum_bid_p = ((PLANT*)node2->data)->p_sum_accept_bid_p =
0.0;
   ((PLANT*)node2->data)->o_sum_bid_p = ((PLANT*)node2->data)->o_sum_accept_bid_p =
0.0;
    node2 = node2->next;
  }


return 0;
}

//---------------------------------------------------------------------------
//Function 13

int agent(ROOT **root){

 ROOT *c1 = NULL ,* c2 = NULL ,* c3 = NULL;
 NODE *node = (*root)->head;

 float m, w, cost;
 float total_cap,accept_cap;
 float temp_price;
 int s,i,j;

 randomize();

 for(j=0;j<((*root)->num);j++){

  s1(prev_profit[k]) = s1(profit[k]);
  s1(profit[k])=0.0;
  total_cap=0.0;
  accept_cap=0.0;

  for(i=0;i<(s1(plant_num));i++) {            //the calculation of the profit & ur//

   if(s2(bid_flag[k])==1) {

    cost=(s2(cost_coef.cca))*(s2(accept_bid_q[k]))*(s2(accept_bid_q[k])) +
        (s2(cost_coef.ccb))*(s2(accept_bid_q[k]))+ (s2(cost_coef.ccc));

    s1(profit[k])= (s1(profit[k]))+ (s2(accept_bid_p[k]))*(s2(accept_bid_q[k]))- cost;
   }

   accept_cap= accept_cap + (s2(accept_bid_q[k]));

   total_cap = total_cap + s2(cap);

   s2(prev_bid_p[k])= s2(bid_price[k]);

  }


  s1(ur[k])=(accept_cap)/(total_cap);


  if((s1(ur[k]))>=(s1(tur))){
```

```
make_plant_list_cycle(node,&c1,&c2,&c3);

sort_plant_list_cycle(&c1,&c2,&c3,compare_BIDPRICE2);

if(check_list_cycle(&c1, &c2, &c3)==0)

 update_price(&c1, &c2, &c3);

else if(check_list_cycle(&c1, &c2, &c3)==1){

        if(s1(profit[k]) <= s1(prev_profit[k])){

 if(s1(gene[k].g[0])==0) {

  s1(gene[k].g[0])=1;

  s=random(100);

  if(s>=49)
                m =1.0+ (1.0*random(11)/100.0);
         else if (s<49)
        m =1.0-(1.0*random(11)/100.0);

  for(i=0;i<(s1(plant_num));i++) {
   s2(bid_price[k]) = (s2(prev_bid_p[k]))*m;
  }
 }

 else if (s1(gene[k].g[0])==1) {

 //printf("%d\n", s1(gene[k].g[1]));
  if(s1(gene[k].g[1])==0) {

   s1(gene[k].g[1])=1;
        m =1.0+(1.0*random(11)/100.0);

   for(i=0;i<(s1(plant_num));i++) {
         s2(bid_price[k]) = s2(prev_bid_p[k])*m;
   }
  }
  else if(s1(gene[k].g[1])==1){
   s1(gene[k].g[1])=0;
        m =1.0-(1.0*random(11)/100.0);
   for(i=0;i<(s1(plant_num));i++) {
         s2(bid_price[k]) = (s2(prev_bid_p[k]))*m;
   }
  }
 }

}
 else if(s1(profit[k])>s1(prev_profit[k])){

  for(i=0;i<(s1(plant_num));i++) {
  s2(bid_price[k]) = s2(prev_bid_p[k]);
  }
```

```
    if(s1(gene[k].g[1])==0)
     s1(gene[k].g[1])=1;
    else
    if(s1(gene[k].g[1])==1)
     s1(gene[k].g[1])=0;
   }
  }
 }

 else if((s1(ur[k]))<(s1(tur))){
  s1(gene[k].g[0])=0;
  s1(gene[k].g[1])=random(2);
  w=1.0-(1.0*random(11)/100.0);
  for(i=0;i<(s1(plant_num));i++){
   s2(bid_price[k]) = s2(prev_bid_p[k])*w;
  }
 }

 //Required in order to eliminate negative profit

 for(i=0;i<(s1(plant_num));i++) {

  cost=(s2(cost_coef.cca))*(s2(bid_quantity[k]))*(s2(bid_quantity[k])) +
      (s2(cost_coef.ccb))*(s2(bid_quantity[k]))+ (s2(cost_coef.ccc));

  temp_price = cost/s2(bid_quantity[k]);

  if((s2(bid_price[k])<temp_price) || (s2(bid_price[k]) >1000.00)){

   if(s2(bid_price[k])<temp_price)
    s2(bid_price[k]) = temp_price;
   else
    s2(bid_price[k]) = 1000.0;
  }
 }

 node = node->next;
}

return 0;
}
```

(hourly_prg2.h)

```
/*MSc in Electrical Power Engineering
  Modelling of Electricity Markets using Software Agents

  Hourly Bidding - 24 hours

  ------------------------------------------------
  DBL_LIST_PRG1.H HEADER FILE
  ------------------------------------------------

  This header file contains fundamental functions:
  A. For the manipulation of doubly linked lists
  B. To aid result manipulation and presentation
  C. For auxilliary tasks*/

//  Macros
//  Macros defined to perform tasks appearing frequently
#define VALID(x) ((*x)!=NULL)
#define NUM(x) ((*x)->num)
#define NEW(x) (x*)malloc(sizeof(x))
#define sc1(x) ((((AGENT*)data)->plant[i])->x)
#define sc2(x) (((AGENT*)(node->data))->x)
#define sc3(x) ((((AGENT*)(node->data))->plant[i])->x)

//  ---------------------------------------------------------------------------
//  User Defined Type Definitions
//  ---------------------------------------------------------------------------

//  PLANT COST COEFFICIENTS - Quadrature Production Cost Function
typedef struct cost_coefficients {
 float cca,ccb,ccc;
```

```
}CC;

typedef struct genes{
 int g[2];
}GENE;

//  PLANT Type - structure
typedef struct plant{
 void *pnt_agent; //This pointer is set to point to the parent plant agent.
 int agent_id;
 int id;
 int cycle;
 float cap, mc;
 float bid_quantity[24], bid_price[24];
 float accept_bid_q[24], accept_bid_p[24];
 float prev_bid_q[24], prev_bid_p[24];
 int bid_flag[24];
 float p_sum_bid_p, p_sum_accept_bid_p;
 float p_ave_bid_p, p_ave_accept_bid_p;
 float o_sum_bid_p, o_sum_accept_bid_p;
 float o_ave_bid_p, o_ave_accept_bid_p;
 CC cost_coef;
}PLANT;

//  AGENT Type - Structure
typedef struct agent{
 PLANT *plant[BUFSIZ];
 int agent_id;
 int plant_num;
 GENE gene[24];
 float tur;
 float ur[24];
 float prev_ur[24];
 float profit[24];
 float prev_profit[24];
 float col_profit;
 float col_ur;
 float p_sum_bid_p, p_sum_accept_bid_p;
 float p_ave_bid_p, p_ave_accept_bid_p;
 float o_sum_bid_p, o_sum_accept_bid_p;
 float o_ave_bid_p, o_ave_accept_bid_p;
}AGENT;

//  TYPE DEFINTITIONS FOR DBL LINKED LISTS AND TREES

//  NODE OF A DBL LINKED LIST
typedef struct node {
 void  *data;
 struct node *next;
 struct node *prev;
}NODE;

//  ROOT OF A DBL LINKED LIST
typedef struct {
 long num;
 NODE *head;
 NODE *tail;
```

```
}ROOT;

//Function Prototypes

ROOT *make_root (void);
NODE *dbl_make_node (void *data);
int dbl_insert_data (ROOT **root, void *data, int position);
int reset_list (ROOT **root);

void display_list (ROOT **root, void (*disp) (void *data, FILE *),FILE *f1);
void display_list2(ROOT **root, void (*disp) (void *data, FILE *),FILE *f);
void display_AGENT (void *data, FILE *f1);
void display_AGENTb (void *data, FILE *filename);
void display_titles (void *data, FILE *filename);
void display_PLANT(void *data, FILE *f1);
void display_PLANTb(void *data, FILE *f1);

float compare_BIDPRICE (void *d1, void *d2);
float compare_BIDPRICE2 (void *d1, void *d2);
int save_list(char *filename, ROOT **root, size_t datasize);
int recover_list (char *filename, ROOT **root, size_t datasize);

//  ---------------------------------------------------------------------------


//  ---------------------------------------------------------------------------
//  FUNCTIONS RELATED TO THE MANIPULATION OF DOUBLY LINKED LISTS
//  ---------------------------------------------------------------------------

//1.Function to make root of a doubly linked list - Required by F3

ROOT *make_root (void) {

 ROOT *root;

 if((root=NEW(ROOT))!=NULL) {
  root->head = root->tail=NULL;
  root->num = 0;
 }

return root;
}
//  ---------------------------------------------------------------------------
//2.Function to make node of a doubly linked list - Required by F3

NODE *dbl_make_node (void *data){

 NODE *node;

 if((node = NEW(NODE))!=NULL) {
  node->data = data;
  node->next = NULL;
  node->prev = NULL;
 }

return node;
}
```

```
//  ---------------------------------------------------------------------------
//3.Function to insert data in a doubly linked list

int dbl_insert_data (ROOT **root, void *data, int position) {

 NODE *neu;

 if(!VALID(root))
  if((*root = make_root())==NULL)
   return(-1);

 if((neu = dbl_make_node (data)) == NULL){
  if(NUM(root)==0){
   free(*root);
   *root=NULL;
  }
  return (-1);
 }

 if(NUM(root)==0)
  (*root)->tail=(*root)->head= neu;

 else {

  if(position == 0) {
   neu->next = (*root)->head;
   (*root)->head->prev = neu;
   (*root)->head = neu;
  }

  else{
   (*root)->tail->next = neu;
   neu->prev = (*root)->tail;
   (*root)->tail = neu;
  }

 }

 (NUM(root))++;

return 0;
}

//  ---------------------------------------------------------------------------
//4.Function to Reset a doubly linked list

int reset_list (ROOT **root) {

 NODE *thes, *next;

 if(!VALID(root))
  return (-1);

 thes = (*root)->head;

 do{
    next = thes->next;
```

```
    free(thes);
    thes = next;
   } while(thes!=NULL);


 free(*root);
 *root = NULL;

return 0;
}

//  ---------------------------------------------------------------------------
//  DISPLAY FUNCTIONS
//  ---------------------------------------------------------------------------

//5. Function to display data pointed by the nodes of a doubly linked list

void display_list (ROOT **root, void (*disp) (void *data, FILE *),FILE *f1) {

 NODE *node;

 if(VALID(root)) {
  node = (*root)->head;

 do{
    disp(node->data, f1);
    node = node->next;
   } while (node!=NULL);

 }

 fputc('\n',f1);
}

//  ---------------------------------------------------------------------------
//5b. Function to display data pointed by the nodes of a doubly linked list

void display_list2(ROOT **root, void (*disp) (void *data, FILE *),FILE *f) {

 NODE *node;
 int i;

 if(VALID(root)) {
  node = (*root)->head;

  for(i=0;i<(*root)->num;i++){
   disp(node->data, f+(i+1));
   node = node->next;
  }

 putchar('\n');
 }
}

//  ---------------------------------------------------------------------------
void display_AGENTb (void *data, FILE *filename){
```

```
int i;
float capacity=0.0;

 for(i=0;i<((AGENT*)data)->plant_num;i++){
  fprintf(filename,"%8.2f\t",((AGENT*)data)->plant[i]->o_ave_bid_p);
  fprintf(filename,"%8.2f\t",((AGENT*)data)->plant[i]->o_ave_accept_bid_p);
  fprintf(filename,"%8.2f\t",((AGENT*)data)->plant[i]->p_ave_bid_p);
  fprintf(filename,"%8.2f\t",((AGENT*)data)->plant[i]->p_ave_accept_bid_p);
 }

 fprintf(filename,"%8.2f\t",((AGENT*)data)->o_ave_bid_p);
 fprintf(filename,"%8.2f\t",((AGENT*)data)->o_ave_accept_bid_p);
 fprintf(filename,"%8.2f\t",((AGENT*)data)->p_ave_bid_p);
 fprintf(filename,"%8.2f\t",((AGENT*)data)->p_ave_accept_bid_p);
 fprintf(filename,"%8.2f\t",((AGENT*)data)->col_profit);

 for(i=0;i<((AGENT*)(data))->plant_num;i++)
  capacity = capacity + ((AGENT*)(data))->plant[i]->cap;

 fprintf(filename,"%8.2f\t",(((AGENT*)data)->col_profit)/capacity);

 fprintf(filename,"%8.2f\t",((AGENT*)data)->col_ur);

  fprintf(filename,"\n");

}
//  ------------------------------------------------------------------------

void display_titles (void *data, FILE *filename){

 int i;

 fprintf(filename,"Agent[%d]\n",((AGENT*)data)->agent_id);

 fprintf(filename,"\n");

 for(i=0;i<((AGENT*)data)->plant_num;i++){
  fprintf(filename,"Plant[%d]\tCycle:%d                \t\t\t",i,((AGENT*)data)->plant[i]->cycle);
 }

 fprintf(filename,"AGENT SUMMARY                    \t\t\t\tProfit  \tProf_PIC\tUR      \t");

 fprintf(filename,"\n");

 for(i=0;i<((AGENT*)data)->plant_num+1;i++){
  fprintf(filename,"OFF-PEAK        \t\tPEAK            \t\t",i);
 }

 fprintf(filename,"\n");

 for(i=0;i<2*((((AGENT*)data)->plant_num)+1);i++){
  fprintf(filename,"BID     \tABP     \t");
}

 fprintf(filename,"\n");

}
```

```
/*void display_titles (void *data, FILE *filename){

 int i;

 fprintf(filename,"Agent[%d]\n",((AGENT*)data)->agent_id);

 fprintf(filename,"\n");

 for(i=0;i<((AGENT*)data)->plant_num;i++){
  fprintf(filename,"Plant[%d]         \t\t",i);
 }

 fprintf(filename,"AGENT SUMMARY                    \t\t\t\tProfit  \t");

 fprintf(filename,"\n");

 fprintf(filename,"\n");

 for(i=0;i<(((AGENT*)data)->plant_num);i++){
  fprintf(filename,"BID     \tABP     \t");
 }

 fprintf(filename,"\n");

} */
//  ------------------------------------------------------------------------
// 6.Function to display plant structure contents

void display_PLANT(void *data, FILE *f1){

 fprintf(f1,"Agent[%d],Plant[%d]->Bid Price=%f,Bid Quantity=%f\n"
            "Accepted Bid Price=%f,Accepted Bid Quantity=%f,Bid_Flag=%d\n"
           ,((PLANT*)data)->agent_id,((PLANT*)data)->id
           ,((PLANT*)data)->bid_price,((PLANT*)data)->bid_quantity
           ,((PLANT*)data)->accept_bid_p,((PLANT*)data)->accept_bid_q
           ,((PLANT*)data)->bid_flag);

 fprintf(f1,"-----------------------------------------\n");

}
//  ------------------------------------------------------------------------
void display_PLANTb(void *data, FILE *f1){

 fprintf(f1,"Agent[%d],Plant[%d]->Bid_P: %f, Bid_Q%f, A_B_P: %f, A_B_Q:%f, BF: %d\n"
           ,((PLANT*)data)->agent_id,((PLANT*)data)->id
           ,((PLANT*)data)->bid_price[k],((PLANT*)data)->bid_quantity[k]
           ,((PLANT*)data)->accept_bid_p[k],((PLANT*)data)->accept_bid_q[k]
           ,((PLANT*)data)->bid_flag[k]
        );
}

//  ------------------------------------------------------------------------
//7.Function to display data contained in an agent structure

void display_AGENT (void *data, FILE *f1){
```

```
 int i;

 fprintf(f1,"Agent ID = %d\n", ((AGENT*)data)->agent_id);
 fprintf(f1,"-----------------------------------------------\n");

 for(i=0;i<((AGENT*)data)->plant_num;i++){

  fprintf(f1,"Plant ID=%d, Marginal Cost:%f,Capacity:%f\n"
          "Cost coef cca =%f, Cost coef ccb =%f,Cost coef ccc =%f\n"
          "Initial Bid Quantity=%f,Initial Bid Price=%f\n"
          "Accepted Bid Quantity=%f,Accepted Bid Price=%f\n"
          "Previously Bid Quantity=%f,Previously Bid Price=%f\n"
          "Bid Flag = %d\n"
          ,sc1(id),sc1(mc),sc1(cap),sc1(cost_coef.cca), sc1(cost_coef.ccb)
          ,sc1(cost_coef.ccc),sc1(bid_quantity),sc1(bid_price)
          ,sc1(accept_bid_q),sc1(accept_bid_p)
          ,sc1(prev_bid_q),sc1(prev_bid_p), sc1(bid_flag));

  fprintf(f1,"-----------------------------------------------\n");
 }

 fprintf(f1,"Target Utilisation Rate:%f\n",((AGENT*)data)->tur);
 fprintf(f1,"Current Utilisation Rate:%f\n",((AGENT*)data)->ur);
 fprintf(f1,"Current Profit:%f\n",((AGENT*)data)->profit);
 fprintf(f1,"-----------------------------------------------\n");
 fprintf(f1,"-----------------------------------------------\n");

}

//  ------------------------------------------------------------------------
/*void display_AGENTb (void *data, FILE *f1){

 int i;
 FILE *f2, *f3, *f4, *f5, *f6,*f7, *f8, *f9;

 if((f2=fopen("out2.dat", "a"))==NULL){
   perror("Cannot open file out.dat");
   exit(0);
1 }

 if((f3=fopen("out3.dat", "a"))==NULL){
   perror("Cannot open file out.dat");
   exit(0);
 }

 if((f4=fopen("out4.dat", "a"))==NULL){
   perror("Cannot open file out.dat");
   exit(0);
 }

 if((f5=fopen("out5.dat", "a"))==NULL){
   perror("Cannot open file out.dat");
   exit(0);
 }

 if((f6=fopen("out6.dat", "a"))==NULL){
```

```
   perror("Cannot open file out.dat");
   exit(0);
 }

  if((f7=fopen("out7.dat", "a"))==NULL){
   perror("Cannot open file out.dat");
   exit(0);
 }

  if((f8=fopen("out8.dat", "a"))==NULL){
   perror("Cannot open file out.dat");
   exit(0);
 }


 i=0;

 fprintf(f2,"%f\n", (((AGENT*)data)->plant[i])->accept_bid_p);
 fprintf(f3,"%f\n", (((AGENT*)data)->plant[i])->bid_price);

 i=1;

 fprintf(f4,"%f\n", (((AGENT*)data)->plant[i])->accept_bid_p);
 fprintf(f5,"%f\n", (((AGENT*)data)->plant[i])->bid_price);

 i=2;

 fprintf(f6,"%f\n", (((AGENT*)data)->plant[i])->accept_bid_p);
 fprintf(f7,"%f\n", (((AGENT*)data)->plant[i])->bid_price);

 fprintf(f8,"%f\n", (((AGENT*)data)->profit));
 fprintf(f9,"%f\n", (((AGENT*)data)->ur));

 fclose(f2);
 fclose(f3);
 fclose(f4);
 fclose(f5);
 fclose(f6);
 fclose(f7);
 fclose(f8);

 } */

//  ------------------------------------------------------------------------
//  COMPARE FUNCTIONS
//  ------------------------------------------------------------------------

// 8.Function to compare two prices - tree

float compare_BIDPRICE (void *d1, void *d2) {
 return ((((PLANT*)d2)->bid_price[k])-(((PLANT*)d1)->bid_price[k]));
}


float compare_BIDPRICE2 (void *d1, void *d2) {
 return ((((PLANT*)d2)->accept_bid_p[k])-(((PLANT*)d1)->accept_bid_p[k]));
}
```

```
//  ---------------------------------------------------------------------------
//  SAVING AND RECOVERING FROM A FILE FUNCTIONS
//  ---------------------------------------------------------------------------

//9.Function to save a doubly linked list to a file

int save_list(char *filename, ROOT **root, size_t datasize) {

 FILE *f1;
 NODE *node;

 if(VALID(root)) {
  if((f1=fopen(filename,"wb"))==NULL)  /*Open and write to the file in binary mode*/
   return (-1);

  node = (*root)->head;

  do{
   if(fwrite(node->data,datasize,1,f1)!=1)
   return(-1);
   node=node->next;
  } while (node!=NULL);

  return(fclose(f1));

 }

return 0;
}

//  ---------------------------------------------------------------------------
//10.Function to recover a double linked list from a file

int recover_list (char *filename, ROOT **root, size_t datasize) {

 FILE *f1;
 void *dptr;

 if((f1=fopen(filename, "rb"))==NULL)
  return(-4);

 do{
  if((dptr = malloc(datasize))==NULL)
  return (-1);

  if(fread(dptr,datasize,1,f1)!=1) {
  if(feof(f1))
   return 0;
   else {
    free(dptr);
    return (-2);
   }
  }

  if(dbl_insert_data(root,dptr,1)!=0)
   return (-3);
```

```
 } while (1);

}
//  ---------------------------------------------------------------------------
```

Appendix 4

Agent Plant Data Input File

(4 generating companies)

The data in the input file "plant3.dat" is organised as follows:

L*1*>capacity of plant 1 (float), marginal cost of plant 1 (float), cycle, plant_flag (=1)

L*2*>capacity of plant 2 (float), marginal cost of plant 2 (float), cycle, plant_flag (=1)

.

.

L*n*>capacity of plant n (float), marginal cost of plant n (float), cycle, plant_flag (=0)

Once the flag "plant_flag" is set to zero, the function will stop accepting data related to the plants of the current agent. It will proceed to read other general data related to the current agent.

L*n+1*>target utilisation rate , agent_flag (=1)

Once the agent_flag is set to one (1), the function will stop accepting data regarding the current agent. It will proceed by reading plant data for the next agent. If the agent_flag is set to zero (0), then the function will stop reading data from the file and the process will be terminated.

Sample File (In accordance with the data given in Appendix 1)

380.0 7.73 0.0095 0.5 0.0 1 1

405.0 8.19 0.0095 0.5 0.0 1 1

1972.0 17.49 0.0042 1.0 0.0 1 1

996.0 17.49 0.0083 1.0 0.0 2 1

970.0 17.64 0.0086 1.0 0.0 2 1

976.0 19.74 0.0097 1.0 0.0 2 1

945.0 20.32 0.01 1.0 0.0 2 1

90.0 33.3 0.168 3.0 0.0 3 0

0.6 1

690.0 7.44 0.005 0.5 0.0 1 1

970.0 7.58 0.0036 0.5 0.0 1 1

680.0 7.87 0.0054 0.5 0.0 1 1

500.0 7.87 0.0073 0.5 0.0 1 1

650.0 8.19 0.0059 0.5 0.0 1 1

1935.0 10.8 0.0025 1.0 0.0 2 1

1935.0 11.43 0.0027 1.0 0.0 2 1

1455.0 11.66 0.0037 1.0 0.0 2 1

2005.0 11.83 0.0027 1.0 0.0 2 1

680.0 11.91 0.0082 0.8 0.0 2 1

1960.0 12.07 0.0028 1.0 0.0 2 1

626.0 12.72 0.0096 0.8 0.0 2 1

456.0 13.52 0.014 0.8 0.0 2 1

685.0 14.13 0.0087 2.5 0.0 3 1

484.0 14.29 0.012 2.5 0.0 3 1

188.0 14.48 0.032 2.5 0.0 3 1

270.0 27.30 0.046 2.5 0.0 3 1

269.0 27.3 0.046 2.5 0.0 3 0

0.6 1

1500.0 7.44 0.0021 1.0 0.0 1 1

740.0 7.58 0.0048 0.5 0.0 1 1

940.0 8.03 0.0038 0.8 0.0 1 1

2008.0 11.55 0.0025 1.5 0.0 2 1

1940.0 11.57 0.0026 1.5 0.0 2 1

2000.0 11.62 0.0025 1.5 0.0 2 1

1470.0 11.77 0.0037 1.0 0.0 2 1

1950.0 11.78 0.0026 1.5 0.0 2 1

2025.0 15.46 0.0034 1.5 0.0 3 1

163.0 27.30 0.08 0.5 0.0 3 1

163.0 27.30 0.08 0.5 0.0 3 0

0.6 1

1050.0 1.00 0.00038 0.2 0.0 1 1

475.0 1.00 0.00084 0.2 0.0 1 1

445.0 1.00 0.0009 0.2 0.0 1 1

440.0 1.00 0.0009 0.2 0.0 1 1

430.0 1.00 0.00093 0.2 0.0 1 1

240.0 1.00 0.0017 0.2 0.0 1 1

192.0 1.00 0.002 0.2 0.0 1 0

1.0 0

Appendix 5

Demand Profile Input File

(demand.dat)

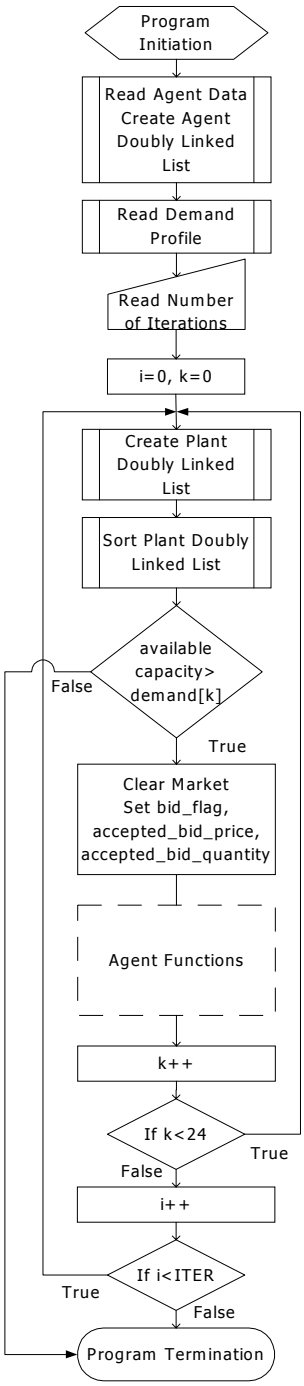
Line 1	28872.0
2	28872.0
3	28872.0
4	28872.0
5	29576.0
6	29576.0
7	29576.0
8	29576.0
9	32393.0
10	32393.0
11	32393.0
12	32393.0
13	27816.0
14	27816.0
15	27816.0
16	27816.0
17	24647.0
18	24647.0
19	24647.0
20	24647.0
21	24647.0
22	24647.0
23	24647.0
24	24647.0

Note: All demand figures are in MW

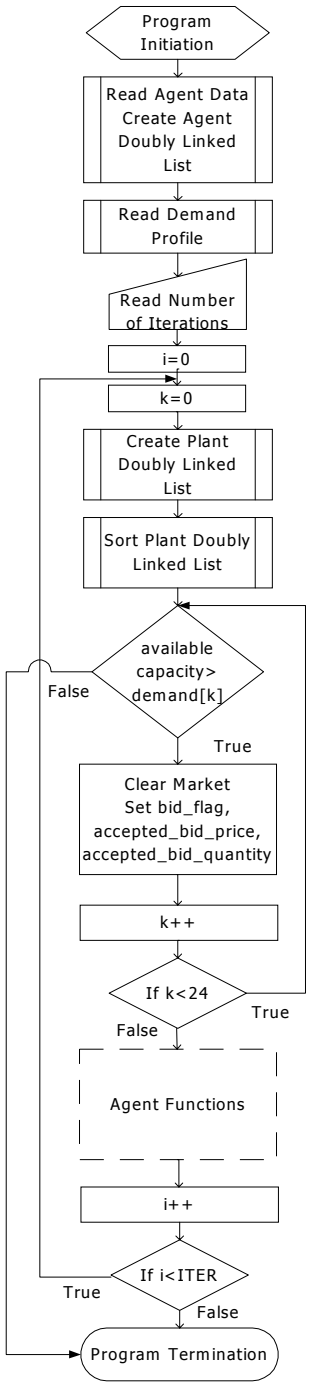
Appendix 6

Initialisation and Market Mechanism Flow Diagrams

Hourly

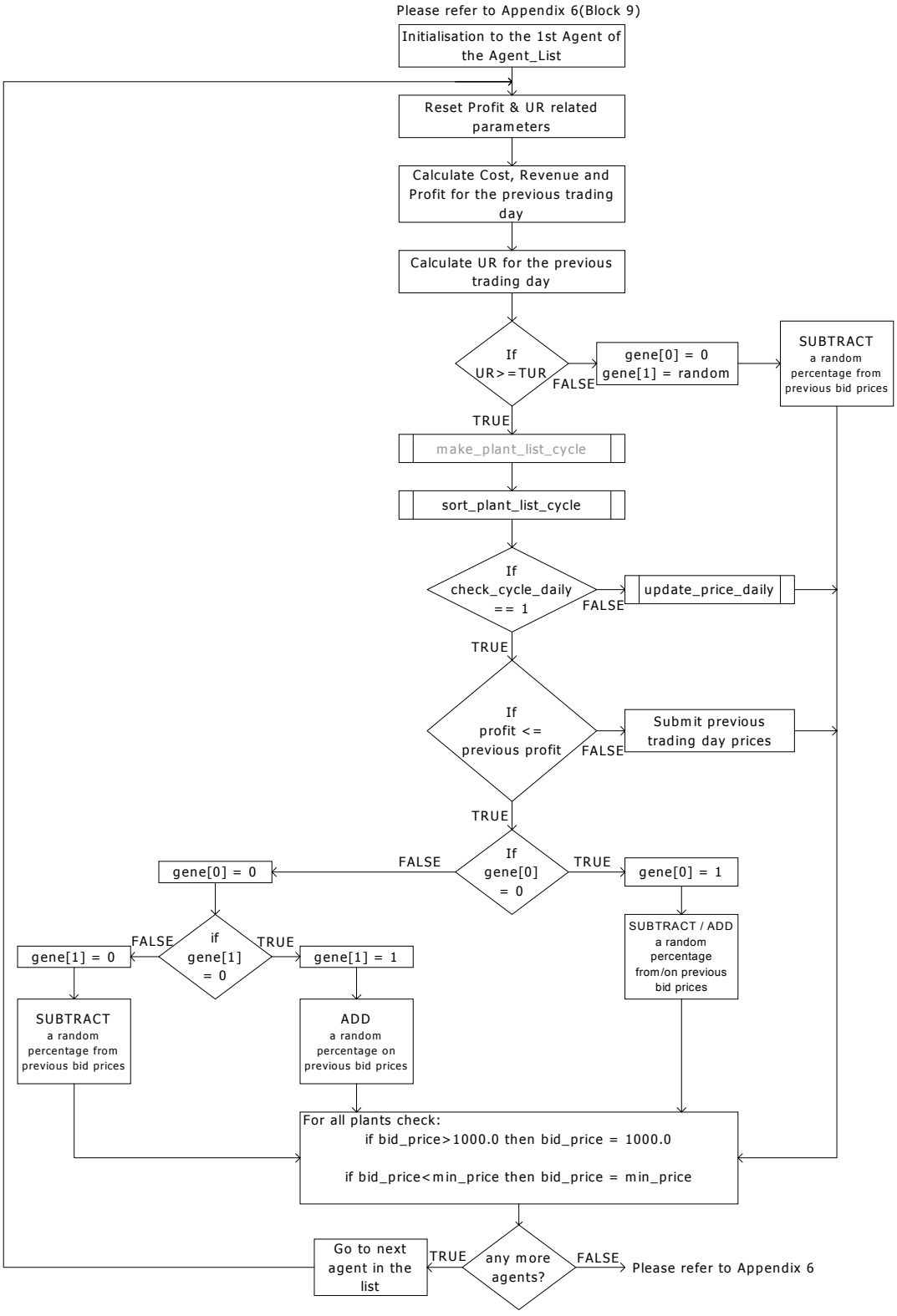


Daily



Appendix 7

Agent Flow Diagram



Appendix 8/9/10/11

Case Studies 1,2,3 and 4 – Results

NOTE

These Appendices can be found in the "Appendices" directory, which is stored in the Compact Disc. The Compact Disc is in the plastic case on the back cover of this dissertation.

Appendix 12

Agent Plant Data Input File

(6 generators)

```
380.0 7.73 0.0095  0.5 0.0 1 1
405.0 8.19 0.0095  0.5 0.0 1 1
1972.0 17.49 0.0042  1.0 0.0 1 1
996.0 17.49 0.0083  1.0 0.0 2 1
970.0 17.64 0.0086  1.0 0.0 2 1
976.0 19.74 0.0097 1.0 0.0 2 1
90.0 33.3 0.168 3.0 0.0  3 0
0.6 1
690.0 7.44 0.005 0.5 0.0 1 1
970.0 7.58 0.0036 0.5 0.0 1 1
1935.0 10.8 0.0025 1.0 0.0 2 1
1935.0 11.43 0.0027 1.0 0.0 2 1
1455.0 11.66 0.0037 1.0 0.0 2 1
2005.0 11.83 0.0027 1.0 0.0 2 1
685.0 14.13  0.0087 2.5 0.0 3 1
484.0 14.29  0.012 2.5 0.0 3 0
0.6 1
680.0 7.87 0.0054 0.5 0.0 1 1
500.0 7.87 0.0073 0.5 0.0 1 1
680.0 11.91  0.0082 0.8 0.0 2 1
1960.0 12.07 0.0028 1.0 0.0 2 1
626.0 12.72  0.0096 0.8 0.0 2 1
456.0 13.52  0.014  0.8 0.0 2 1
188.0 14.48  0.032 2.5 0.0 3 1
270.0 27.30  0.046 2.5 0.0 3 0
0.6 1
650.0 8.19 0.0059 0.5 0.0 1 1
1500.0 7.44 0.0021 1.0 0.0 1 1
2008.0 11.55 0.0025 1.5 0.0 2 1
1940.0 11.57 0.0026 1.5 0.0 2 1
2000.0 11.62 0.0025 1.5 0.0 2 1
269.0 27.3   0.046 2.5 0.0 3 1
163.0 27.30  0.08 0.5 0.0 3 0
0.6 1
740.0 7.58 0.0048 0.5 0.0 1 1
940.0 8.03 0.0038 0.8 0.0 1 1
1470.0 11.77 0.0037 1.0 0.0 2 1
1950.0 11.78 0.0026 1.5 0.0 2 1
945.0 20.32 0.01 1.0 0.0 2 1
2025.0 15.46 0.0034 1.5 0.0 3 1
163.0 27.30  0.08 0.5 0.0 3 0
0.6 1
1050.0 1.00 0.00038 0.2 0.0 1 1
475.0 1.00 0.00084 0.2 0.0 1 1
445.0 1.00 0.0009 0.2 0.0 1 1
440.0 1.00 0.0009 0.2 0.0 1 1
430.0 1.00  0.00093 0.2 0.0 1 1
240.0 1.00 0.0017 0.2 0.0 1 1
192.0 1.00 0.002 0.2 0.0 1 0
1.0  0
```

Appendix 13

Agent Plant Data Input File

(2 generators)

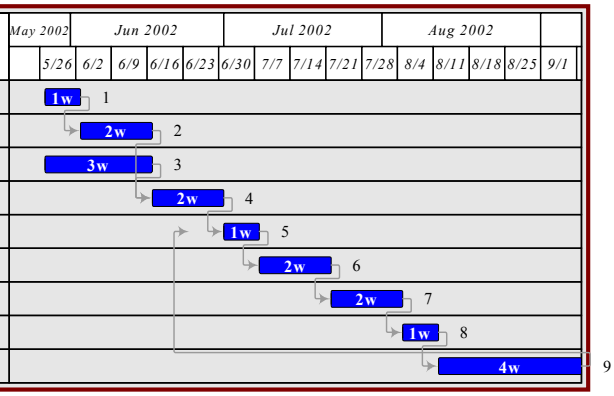
```
380.0 7.73 0.0095  0.5 0.0 1 1
405.0 8.19 0.0095  0.5 0.0 1 1
1972.0 17.49 0.0042  1.0 0.0 1 1
1500.0 7.44 0.0021 1.0 0.0 1 1
740.0 7.58 0.0048 0.5 0.0 1 1
940.0 8.03 0.0038 0.8 0.0 1 1
996.0 17.49 0.0083  1.0 0.0 2 1
970.0 17.64 0.0086  1.0 0.0 2 1
976.0 19.74 0.0097 1.0 0.0 2 1
945.0 20.32 0.01 1.0 0.0 2 1
2008.0 11.55 0.0025 1.5 0.0 2 1
1940.0 11.57 0.0026 1.5 0.0 2 1
2000.0 11.62 0.0025 1.5 0.0 2 1
1470.0 11.77 0.0037 1.0 0.0 2 1
1950.0 11.78 0.0026 1.5 0.0 2 1
90.0 33.3 0.168 3.0 0.0  3 1
2025.0 15.46 0.0034 1.5 0.0 3 1
163.0 27.30  0.08 0.5 0.0 3 1
163.0 27.30  0.08 0.5 0.0 3 1
690.0 7.44 0.005 0.5 0.0 1 1
970.0 7.58 0.0036 0.5 0.0 1 1
680.0 7.87 0.0054 0.5 0.0 1 1
500.0 7.87 0.0073 0.5 0.0 1 1
650.0 8.19 0.0059 0.5 0.0 1 1
1935.0 10.8 0.0025 1.0 0.0 2 1
1935.0 11.43 0.0027 1.0 0.0 2 1
1455.0 11.66 0.0037 1.0 0.0 2 1
2005.0 11.83 0.0027 1.0 0.0 2 1
680.0 11.91  0.0082 0.8 0.0 2 1
1960.0 12.07 0.0028 1.0 0.0 2 1
626.0 12.72  0.0096 0.8 0.0 2 1
456.0 13.52  0.014  0.8 0.0 2 1
685.0 14.13  0.0087 2.5 0.0 3 1
484.0 14.29  0.012 2.5 0.0 3 1
188.0 14.48  0.032 2.5 0.0 3 1
270.0 27.30  0.046 2.5 0.0 3 1
269.0 27.3   0.046 2.5 0.0 3 0
0.6 1
1050.0 1.00 0.00038 0.2 0.0 1 1
475.0 1.00 0.00084 0.2 0.0 1 1
445.0 1.00 0.0009 0.2 0.0 1 1
440.0 1.00 0.0009 0.2 0.0 1 1
430.0 1.00  0.00093 0.2 0.0 1 1
240.0 1.00 0.0017 0.2 0.0 1 1
192.0 1.00 0.002 0.2 0.0 1 0
1.0 0
```

Appendix 14

Gantt Chart

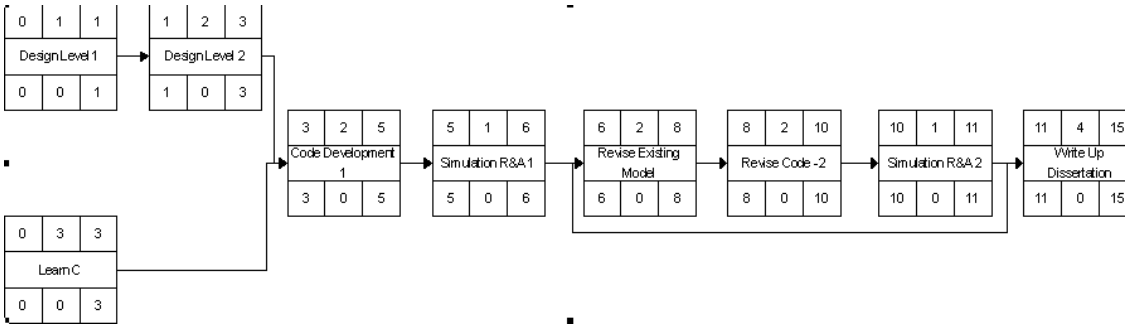
MSc Project in Electrical Power Engineering - ABS

ID	Task Name	Start	End	Duration
1	Design Level 1	27/5/2002	2/6/2002	1w
2	Design Level 2	3/6/2002	16/6/2002	2w
3	Learn C Language (Advanced)	27/5/2002	16/6/2002	3w
4	Code Development - Stage 1	17/6/2002	30/6/2002	2w
5	Simulation Results & Analysis1	1/7/2002	7/7/2002	1w
6	Revise Existing Model - 2	8/7/2002	21/7/2002	2w
7	Revise Code - Stage 2	22/7/2002	4/8/2002	2w
8	Simulation Results & Analysis2	5/8/2002	11/8/2002	1w
9	Write Up Dissertation	12/8/2002	8/9/2002	4w



Appendix 15

Activity-On-Node Diagram



References

[1] Day, C.J.; Bunn, D.W.: 'Generation Asset Divestment in the England and Wales Electricity Market: A Computational Approach to Analysing Market Power', (Decision Technology Centre, London Business School, April 1999)

[2] Luger, G.F.: 'Artificial Intelligence; Structures and Strategies for Complex Problem Solving', (Addison-Wesley, USA, 2002) 4th edn.

[3] Durfee, E.H.; Lesser, V.:' Negotiating Task Decomposition and Allocation using Partial Global Planning', in Gasser, L. ; Huhns, M.: 'Distributed Artificial Intelligence', (Vol II, Morgan Kauffman ,San Francisco) pp.229-244

[4] Simon, H.A.: 'Why Should Machines Learn?', in Michalski, R.S.; Carbonell, J.G.; Mitchell, T.M.: 'Machine Learning: An Artificial Intelligence Approach',(Vol.1, Palo Alto, CA, 1983)

[5] Satton, R.S.; Burton, A.G.: 'Reinforcement Learning', (MIT Press, Cambridge)

[6] Le Baron, B.: 'Building Financial Markets with Artificial Agents; Desired Goals and Present Techniques', in Karakoulas, G: 'Computation Markets', (MIT Press, 1999)

[7] Bunn, D.W. ; Oliveira, F.S.: 'Agent-Based Simulation: An Application to the New Electricity Trading Arrangements of England and Wales', Energy Markets Group, London Business School (available from http://www.london.edu)

[8] Green, R.: 'Increasing Competition in the Electricity Spot Market', Journal of Industrial Economics, Vol. 44(2), pp. 201-216, 1996.

[9] Rudkevitch, A., Duckworth, R.R.: 'Modelling Electricity Pricing in a Deregulated Generation Industry: The Potential for Oligopoly Pricing in the Pool', The Energy Journal , Vol. 19(3), pp. 19-48, 1998.

[10] H.M. von de Fehr, N.; Harbord, D.: 'Spot Market Competition in the UK Electricity Industry', The Economic Journal, Vol. 103 (418), pp.531-546, 1993.

[11] Bower, J. ; Bunn, D.: 'A Model-Based Comparison of Pool and Bilateral Market Mechanisms for Electricity Trading', Energy Markets Group, London Business School (available from http://www.london.edu)

[12] Tesfatsion, L.: 'Agent-Based Computational Economics: Growing Economies from the Bottom-Up', Economics Working Paper No.1, Iowa State University, March 2002

Bibliography

A. NETA – Market Mechanisms, Role Playing Simulations

1 Ofgem. (1999, July). The new electricity trading arrangements

http://www.ofgem.gov.uk

2. Ofgem/DTI.(1999, October). The new electricity trading arrangements, Ofgem/DTI Conclusions Document

http://www.ofgem.gov.uk

3. Ofgem. (2000, March). Settlement Administration Agent User Requirements Specification

http://www.ofgem.gov.uk

4. Ofgem. (2000, April). NGC Incentives under Neta

http://www.ofgem.gov.uk

5. London Economics. (1999, October). Role Playing Simulations of the new electricity trading arrangements, A report to the RETA programme

http://www.ofgem.gov.uk

B. Introduction to Agent Based Computational Economics (ABC)

1. Tesfatsion, L.: "Agent-Based Computational Economics: Growing Economies from the Bottom Up", Economics Working Paper No. 1, Iowa State University, March 2002

2. Tesfatsion, L.: “Agent-Bassed Computational Economics: A Brief Guide to the Literature”, Reader’s Guide to Social Sciences, Fitzroy-Dearborn, London, UK, March 2001

3. Le Baron, B.: "Agent Based Computational Finance: Suggested Readings and early research", Journal of Economic Dynamics and Control

C. AGENTS MODELLING - As Applied in Restructured Electricity Markets

1. Bunn, W.D.; Oliveira, F.S.: "Agent-based Simulation: An application to the New Electricity Trading Arrangements of England and Wales"
http://www.london.edu
2. "A Model-Based Comparison of Pool and Bilateral Market Mechanisms for Electricity Trading"
http://www.london.edu
3. Bower, J.; Bunn, D.: "Experimental Analysis of the efficiency of uniform price versus discriminatory auctions in the England and Wales electricity Market", Journal of Economic Dynamics & Control, 25(2001) 561-562
4. Bunn, D. W.; Bower, J.: "ABS investigation of generator market power in the England and Wales electricity market using an Excel/VBA platform", Proceedings of Agent 2000, Argonne National Laboratory, Chicago Illinois, USA
5. Bunn, D. W.: " Forecasting loads and prices in competitive power markets", Proceedings of the IEEE, 2000, Vol. 88:2, pp. 163-169
6. Bunn, D.W.; Day, C.J.: "Divestiture of generation assets in the electricity pool of England and Wales: A computational approach to analyzing market power", Journal of Regulatory Economics 2001, Vol. 19:2, pp. 123-141

7. Bunn, D.W.; Day, C.J.: “Computational Modelling of Price Formation in the Electricity Pool of England and Wales”, May 2001

D. Auctions – Bidding Strategies

1. Bernard, C.J. et al: “Alternative Auction Institutions for Purchasing Electric Power”, Cornell University
2. Cason, T.N.; Friedman, D.: “Price Formation in Double Auction Markets”, Journal of Economic Dynamics & Control, 20 (1996), 1307-1337
3. Cox, I.; Smith, V.; Walker, J.: “ Theory and Behaviour of multiple unit discriminative auctions”, Journal of Finance, 39 (4) (September), pp. 983-1010, 1984
4. Rothkopf, M.H.; Harstad, R.M.: “Modelling Competitive Bidding: A Critical Essay”. Management Science, Vol 40 June, No.3, pp. 364-384, 1994
5. Rothkopf, M.H.: “Daily Repetition: A neglected factor in the analysis of electricity auctions”, 1997
6. Wolfram, C.D.: “Strategic Bidding in a Multiunit Auction: An Empirical Analysis of Bids to Supply Electricity in England and Wales”, 1997
7. Shelb⌈, G.B.: “Computational Auction Mechanisms for Restructured Power Industry Operation”, Boston; London: Kluwer Academic, 1999 (The Kluwer International Series in Engineering)