

A NEW METAHEURISTIC APPROACH TO THE UNIT COMMITMENT PROBLEM

Ana Viana^{1,2}, Jorge Pinho de Sousa^{1,3}, Manuel Matos^{1,3}

¹INESC Porto, ²Instituto Superior de Engenharia do Porto, ³Faculdade de Engenharia da Universidade do Porto
Porto, Portugal

{aviana, jsousa, mmatos}@inescporto.pt

Abstract - In recent years, metaheuristics have been applied with some success to the Unit Commitment (UC) problem. Methods like Simulated Annealing, Tabu Search and Evolutionary Programming can be found in several papers, presenting results that are sufficiently interesting to justify further research in the area. In this paper, a new metaheuristic approach based on GRASP - Greedy Randomized Adaptive Search Procedure - is applied to the thermal UC problem. Due to its adaptability properties, the method is particularly suitable for solving multi-period problems, such as the UC problem. GRASP is applied, in its current version, to several instances and compared with other techniques. Computational results show the potential of the approach.

Keywords: Unit Commitment, Heuristics, Metaheuristics, Object Oriented

1 Introduction

Due to its economical importance, the Unit Commitment (UC) problem has for long been a matter of concern for generator companies (GENCOs) and, consequently, an important area of research. The UC problem is the on/off problem of selecting the power generating units to be in service, and deciding for how long will they remain in that state, for a given planning horizon (lasting from 1 day to 2 weeks, and generally split in periods of 1 hour). The committed units must satisfy the forecasted system load and reserve requirements, at minimum operating cost, subject to a large set of other system, technological and environmental constraints. Given that operating costs depend on the load assigned to each generator, the problem of committing units is directly connected to the additional problem of (roughly) assigning the load demand to the units that are on (“pre-dispatch” problem). This part of the problem can be easily solved by using, e.g., the λ -iteration method [1]. The final, actual assignment is done later, for much shorter periods (usually from 15 to 60 minutes).

However, being a combinatorial true multi-period problem (due to important start-up and shut-down costs) the UC problem is in general very hard to solve, as it is not possible to perform a separate optimisation for each time interval. The exact methods that have been used to solve the problem proved to be inefficient and, in general, incapable of finding a solution within the time available for real decision making. In fact, according to the results reported in the literature, exact methods like Dynamic Programming and Branch-and-Bound are only ca-

pable of solving small size problems. For medium and large size instances, heuristic approaches must be considered.

During the last decades, many of such approaches (capable of efficiently finding satisfactory, not necessarily optimal, solutions) have been developed to solve the UC problem. Heuristic approaches based on exact methods, e.g. Branch-and-Bound with cuts, as well as methods based on priority-lists and Lagrangian Relaxation or, more recently, metaheuristics (e.g. Genetic Algorithms, Simulated Annealing, Tabu Search) have been used [2]. Metaheuristics are particularly appealing: they are usually easier to develop and less problem dependent than a constructive heuristic; they are capable of easily incorporating new information/details in the model (being therefore easy to adapt to different problem variants); they can tackle complex cost functions and they are able to solve the very same problem for completely different objective functions, without any changes in the code.

In this work, another metaheuristic approach is used to solve the UC problem: GRASP - Greedy Randomized Adaptive Search Procedure [3]. A very interesting feature of GRASP is its *adaptability* property. Following a philosophy that is slightly different from standard metaheuristics, the decisions taken by the method, when building a solution, are somehow *adapted* according to other decisions that were previously taken. This dynamic “learning-process” often leads to very good solutions.

For the sake of generality, no instance specific characteristics were included in the implementation presented in this work. By doing so, one obtains a more general framework that can, if necessary, be adapted to each particular application context and be, in the future, applied to decentralized multiobjective environments. However, for general evaluation purposes, the instances presented in [4] (where a global management of the power units is performed) were used, allowing a comparison between GRASP, Lagrangian heuristics and an optimisation tool based on Genetic Algorithms. The computational results show that the method is robust and effective at handling centralized UC problems. It therefore deserves some further research, and can be seen as a first step for the design of more advanced tools, capable of handling the additional information that must be considered in competitive electricity markets ([5], [6]).

2 GRASP

GRASP [3] is a metaheuristic composed of two main phases: a construction phase and a local search phase. In the construction phase, an initial feasible solution is iteratively built following an adaptive reasoning, i.e., the decisions taken in previous iterations influence the decision taken in the current iteration. Then, the neighbourhood of that solution is explored, using a local search procedure, and the best overall solution is kept as the starting point for the next iteration. In Algorithm 1 the standard GRASP structure is presented.

Algorithm GRASP

```

BestSolutionFound = NULL
Iter = 0
while (Iter < Max_Iterations) do
    ConstructSolution(Solution)
    LocalSearch(Solution, BetterSolution)
    Update(BestSolutionFound, BetterSolution)
    Iter++
end while

```

Algorithm 1: GRASP structure

To obtain an initial feasible solution, at each iteration of the Construction Phase, all elements (decisions that can be taken) in a set of possible candidates, are ranked according to a *greedy function* that evaluates the contribution to the objective function obtained by choosing that particular element. If the elements in the ranked list reach a given threshold, they are accepted for future decisions and stored in a *Restricted Candidate List* (RCL). The decisions taken in each iteration of the Construction Phase are then randomly selected among those in the RCL. By following this reasoning, one allows that at each iteration of GRASP a different initial solution is obtained and that, hopefully, different regions of the searching space are explored by the Local Search procedure. Finally, at the end of each iteration, the *greedy function* is adapted, so that in the following iterations it will take into account the decisions previously taken. A brief description of the Construction Phase is given in Algorithm 2.

Algorithm Construction Phase

```

Solution = { }
while Solution is not complete do
    RCL = MakeRCL(Solution)
    s = SelectElementAtRandom(RCL)
    Solution = Solution ∪ {s}
    UpdateGreedyFunction()
end while

```

Algorithm 2: Construction Phase

3 Application of GRASP to the UC problem

In this section, the objective function and the constraints that are used in this work are presented, as well as a detailed description of the implemented GRASP algorithm.

3.1 Objective function and constraints

The most frequent objective function in the literature has been considered in this work. It aims at minimizing the total production costs over the planning horizon, expressed as the sum of:

- i) *Fuel costs* - represented by a quadratic function ($F_i = c_i P_i^2 + b_i P_i + a_i$, where F_i is the fuel cost of unit i ; a_i , b_i and c_i are the fuel cost parameters, measured in \$/h, \$/MWh and \$/MW²h, respectively, and P_i is the production level of unit i);
- ii) *Start-up costs* - given by constants that depend on the last period the unit was operating. Two constants are defined: one for hot start-up costs, that are considered when the unit has been OFF for a number of periods smaller or equal to a given value, and another for cold start-up costs, that are considered otherwise;
- iii) *Shut-down costs* - represented by a constant.

Constraints modelling the following aspects are considered:

- i) System power balance demand;
- ii) System reserve requirements;
- iii) Unit initial conditions;
- iv) Unit minimum up and down times;
- v) Unit rate limits.

3.2 GRASP implementation details

3.2.1 Construction Phase

Due to problem specific characteristics, the GRASP Construction Phase for the UC problem turns out to be, in some cases, very simple. When *must-run* and *minimum-up time* constraints are considered, it can happen that before the algorithm reaches a certain period t , the state of a unit has already been fixed to ON, for that period. In Algorithm 3 (line 2) a function that calculates the sum of the maximum production capacity of those units is used (T represents the planning horizon). If the sum, $prod(t)$, verifies the sum of load and reserve requirements, no further calculations are needed for that period. Otherwise, based on the RCL (line 4), the other units are set to ON or OFF. Finally, the operating levels that lead to a minimum operating cost are computed (line 5) and the solution current cost is obtained (line 6).

The elements to be stored in the RCL are the units that can be turned ON in the period being analyzed. The MakeRCL Algorithm (Figure 1) is used to guarantee that the solution will remain feasible. At the beginning, an intermediate list (RCLCandidates) that contains only the units that, if turned ON in period t , will verify the minimum down-time constraints, is built. If those units are not capable of verifying the demand and reserve requirements on their own, a second list (RCLNoCandidates), storing the remaining units of the system that are OFF in period t , is

built. The units in RCLCandidates are then ranked, according to a *greedy function*, and those that reach a certain threshold are stored in the RCL.

Algorithm UC Construction Phase

```

for all t ∈ T do
2:   prod(t) = CheckCurrentProduction(Pmax)
   if the reserve requirements are not verified then
4:     MakeRCL(Solution)
     CalculateDispatchValues(Solution)
6:     CalculateCurrentCost(Solution)
   end if
8: end for

```

Algorithm 3: UC Construction Phase

Equation (1) defines the *greedy function* $gf(i)$ for unit i , in a generic period of time (t). $FuelCost(P_{max}^i)$ represents the fuel cost of unit i , when it is operating at its maximum production level (P_{max}^i); $SU(i)$ represents the start-up costs of unit i (if it was OFF in period $t - 1$), and $SD(i)$ represents the shut-down costs of unit i (if it was ON in period $t - 1$). As the decision of turning ON unit i suppresses the shut-down cost in period t , this cost comes with a minus sign in equation (1).

The units that reach the established threshold are those in the interval given by equation (2), where α is a pre-defined parameter that controls the RCL size, \underline{gf}_t is the lower bound of the greedy function obtained in period t (i.e. $\underline{gf}_t = \min(gf(i))$) and \overline{gf}_t its upper bound.

One should notice that when $\alpha = 0$, GRASP turns into a normal greedy algorithm and, when $\alpha = 1$, it becomes a random walk algorithm.

$$gf(i) = \frac{FuelCost(P_{max}^i) + SU(i) - SD(i)}{P_{max}^i} \quad \forall t \quad (1)$$

$$\underline{gf}_t \leq gf(i) \leq \overline{gf}_t + \alpha (\overline{gf}_t - \underline{gf}_t) \quad (2)$$

After obtaining the RCL, the RCLCandidates are updated and the value of α is incremented. When necessary, the same process is applied to RCLNoCandidates. However, as the elements in this list do not verify the minimum down time constraints, if they are chosen, they must be turned ON, since the last period they were ON, until period t is reached.

3.2.2 Local Search Phase - Neighbourhood Structure

To explore the solution space with a Local Search procedure, a neighbourhood structure has been developed. The way it works is presented in Algorithm 4 and consists in randomly choosing one unit i , a period t and a direction dir , which indicates if the changes will be done to the left (before) or to the right (after) of t . Depending on the state of unit i in period t , this change will mean a shut-down or a start-up for a certain amount of time, decided by $ShutDownInterval(i, t, dir)$ or by $StartUpInterval(i, t, dir)$, respectively.

The rules used in the $ShutDownInterval(i, t, dir)$ procedure, for units that are always ON or OFF, over the en-

tire planning horizon, are described in Algorithm 5. They should be read in the following way: **if** State(...) **or** State(...) **then** Down(...).

State(a, b, c, d) represents the state being studied. $a \in \{ON, OFF\}$ indicates if a unit is always ON or OFF. $b \in \{Left, Right\}$ is the direction of search. $c \in \{ON, OFF\}$ gives the initial state of unit i and d is related to the value of t .

Down($t1, t2$) is the interval for which the unit is turned off. $UT(i)$ and $DT(i)$ stand for the minimum up and down time of unit i , and $per_ini(i)$ stands for the number of periods that unit i has been on/off for $t < 0$ (i.e. for the periods before the planning horizon).

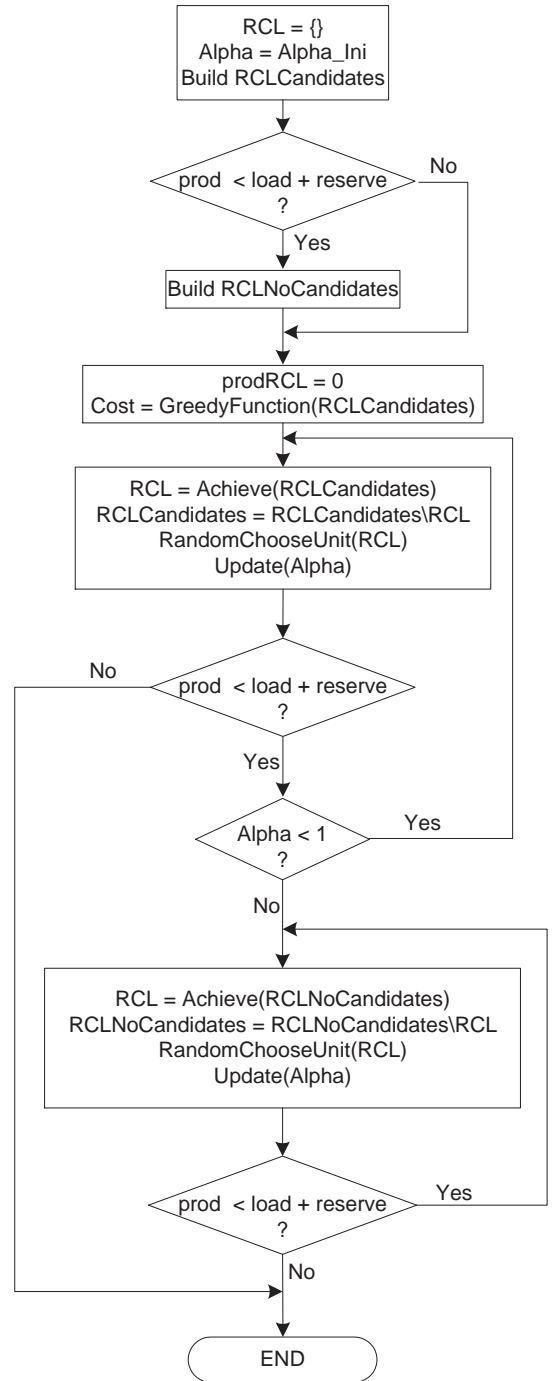


Figure 1: Flowchart for the MakeRCL procedure

Algorithm Neighbourhood

```

i = ChooseAtRandomUnit()
t = ChooseAtRandomTime()
dir = ChooseAtRandomDirection()
if (unit i is ON in period t) then
  ShutDownInterval(i, t, dir)
else
  StartUpInterval(i, t, dir)
end if

```

Algorithm 4: Neighbourhood Structure**Algorithm Rules ShutDown(*i*, *t*, *dir*)**

```

I
t1 = max(UT(i) - per.ini(i), 0)
State(ON, Left, ON, t ≥ DT(i) + t1)
State(ON, Left, OFF, t ≥ UT(i) + DT(i))
Down(t - DT(i) + 1, t)

II
t1 = max(UT(i) - per.ini(i), 0)
State(ON, Left, ON, t < DT(i) + t1)
Down(t - t1, t - t1 + DT(i) - 1)

III
State(ON, Left, OFF, t = 0)
State(ON, Right, OFF, t = 0)
State(ON, Right, ON, t > 0 and t ≥ UT(i) - per.ini)
State(ON, Right, OFF, t ≥ UT(i))
Down(t, t + DT(i) - 1)

IV
t1 = max(UT(i), t - DT(i) + 1)
State(ON, Left, OFF, t > 0 and t < UT(i) + DT(i))
t1 = max(UT(i) - per.ini(i), 0)
State(ON, Right, ON, t < t1)
Down(t1, t1 + DT(i) - 1)

V
State(ON, Right, OFF, t ≠ 0 and t < UT(i))
Down(UT(i), UT(i) + DT(i) - 1)

```

Algorithm 5: Shutting-down Rules

The shutting-down rules for those units that change their state at least once over the planning horizon are the following: if the interval containing t is equal to $UT(i)$, the unit is turned off for all periods in that interval. Otherwise, it is turned off for a number of periods chosen at random in the interval $[1, \text{SumUp} - UT(i)]$, where SumUp represents the number of consecutive periods for which the unit is ON.

Moreover, to guarantee that load and reserve requirements are still verified, additional code is used, when needed, to restore the feasibility of the solution. The feasibility recovery procedure is based on $\text{StartUpInterval}(i, t, dir)$ and turns on units, until feasibility is reached.

With an exception made to the recovery procedure, which is not needed in $\text{StartUpInterval}(i, t, dir)$, the same reasoning is followed for defining the start-up rules.

4 Computational experience

In this section, the results obtained by applying GRASP to some problem instances are reported and dis-

cussed. The code was developed in C++. All computational tests were performed on a 500MHz Pentium III PC.

4.1 Example Instances

The problem instances presented in (see [4], [7]) will be used in this work. All problems consider a 24 hour planning horizon and the number of units varies from 10 to 100 units. Another possibly interesting set of randomly generated diversified problems is that presented in [8]. However, it was not possible to obtain detailed data to reproduce the larger instances contained in that set.

In Table 1, the base problem (for 10 units) is presented. The other problems are generated based on this one, by reproducing the 10 units as many times as necessary. The load values are also multiplied by the same increasing ratio and, in all cases, the reserve requirements are considered to be 10% of the load (Table 2).

	Unit 1	Unit 2	Unit 3	Unit 4	Unit 5
Pmax (MW)	455	455	130	130	162
Pmin (MW)	150	150	20	20	25
a (\$h)	1000	970	700	680	450
b (\$MWh)	16.19	17.26	16.60	16.50	19.70
c (\$MW ² h)	0.00048	0.00031	0.002	0.00211	0.00398
min up (h)	8	8	5	5	6
min down (h)	8	8	5	5	6
hot start cost (\$)	4500	5000	550	560	900
cold start cost (\$)	9000	10000	1100	1120	1800
cold start hours (h)	5	5	4	4	4
initial state (h)	+8	+8	-5	-5	-6
	Unit 6	Unit 7	Unit 8	Unit 9	Unit 10
Pmax (MW)	80	85	55	55	55
Pmin (MW)	20	25	10	10	10
a (\$h)	370	480	660	665	670
b (\$MWh)	22.26	27.74	25.92	27.27	27.79
c (\$MW ² h)	0.00712	0.00079	0.00413	0.00222	0.00173
min up (h)	3	3	1	1	1
min down (h)	3	3	1	1	1
hot start cost (\$)	170	260	30	30	30
cold start cost (\$)	340	520	60	60	60
cold start hours (h)	2	2	0	0	0
initial state (h)	-3	-3	-1	-1	-1

Table 1: Data for the 10 generators problem

In Table 1, a , b and c are the coefficients to be used in the fuel cost function, as mentioned in Section 3.1. In what concerns transition costs, hot start-up costs are considered when the unit has been OFF for a number of periods smaller or equal to the value presented in row *cold start hours*, and cold-start costs are considered otherwise. Shut-down costs are set to zero, for all instances. Finally, the *initial state* row is related to unit initial conditions: a positive value ($+v$) means that the unit has already been ON for v consecutive periods and a negative value indicates that it has been OFF (e.g. unit 6 has been OFF for 3 periods).

Hour	Load (MW)	Reserve (MW)	Hour	Load (MW)	Reserve (MW)
1	700	70	13	1400	140
2	750	75	14	1300	130
3	850	85	15	1200	120
4	950	95	16	1050	105
5	1000	100	17	1000	100
6	1100	110	18	1100	110
7	1150	115	19	1200	120
8	1200	120	20	1400	140
9	1300	130	21	1300	130
10	1400	140	22	1100	110
11	1450	145	23	900	90
12	1500	150	24	800	80

Table 2: Load requirements

4.2 Parameter tuning

To set up a search for GRASP, two parameters have to be tuned: the maximum number of iterations (Max_Iterations in Algorithm 1) and α . Therefore, studying multiple parameter scenarios for the same instance, with the aim of finding the “best” set of values so that the metaheuristic efficiently reaches good solutions is, in this case, relatively simple. In this work, the values of Max_Iterations and α , for each instance, were fixed through some preliminary computational experiments.

4.3 Computational results

When evaluating the performance of random based methods (like metaheuristics) it is natural to test if the methods are correctly implemented, by checking whether different seeds, that initialise the random number generator, do not influence the final results obtained with the method.

In this work, tests have been performed for the 10, 20 and 60 unit instances. For fixed values of α , five seeds were considered and the results obtained showed that they did not influence the global behaviour of the method. In fact, in all cases the same results were obtained, independently of the seed being used, a behaviour that is not necessarily true for other metaheuristics.

For each instance, several computational tests were also done, to obtain the value of α leading to a better performance of the algorithm. That value was fixed at 0.6 for the 20, 40 and 100 unit problems and 0.4 for the 60 unit problem. For the 10 unit problem the method performed equally well for values of α above 0.3. The results obtained are those presented in Table 3.

Problem size	α_{better}	Production cost	CPU time (sec)
10	-	565 825	17
20	0.6	1 128 160	571
40	0.6	2 259 340	1511
60	0.4	3 383 290	2638
100	0.6	5 669 945	4392

Table 3: Computational results for GRASP

4.3.1 Analysis of results

Tables 4 and 5 present the results reported by [4] and [7], respectively. In Table 4, DP stands for Dynamic Pro-

gramming, LH for Lagrangian Heuristic, and GA for Genetic Algorithm. The *Average time* column refers to the average CPU time necessary for convergence.

Problem size	Production cost			Average time (sec)
	DP	LH	GA	
10	565 825	565 825	565 825	221
20	-	1 130 660	1 126 243	733
40	-	2 258 503	2 251 911	2697
60	-	3 394 066	3 376 625	5840
100	-	5 657 277	5 627 437	15733

Table 4: Computational results reported in [4]

Problem size	Production cost	CPU time (sec)
10	564 800	518
20	1 122 622	1147
40	2 242 178	2165
60	3 371 079	2414
100	5 613 127	4045

Table 5: Computational results reported in [7]

In what concerns the results presented in [7], they outperform any results previously published.

However, when comparing the results obtained with GRASP, with those presented in [4] for the Lagrangian Heuristic (LH), they seem to “compete” equally; GRASP presenting better results for the 20 and 60 unit problems, and LH for the 40 and 100 unit ones.

Finally, the comparison with the Genetic Algorithms approach presented, in [4], shows that, though worse, the results obtained with GRASP are of the same order of magnitude of those obtained with the Genetic Algorithms. If one excludes the result obtained for the 100 unit problems, where GRASP had its worst performance (with a deviation of 0.76%, when compared with the result in [4]), for the other instances the maximum deviation was of 0.33%.

Considering that, as far as the authors know, this is the first time that GRASP is applied to the Unit Commitment problem, the results obtained in this work are encouraging for further research in the area. By being able of achieving, for some instances, better results than those obtained with a Lagrangian Heuristic (for long considered a powerful tool in solving the UC problem), and by reaching values that are similar to those obtained with Genetic Algorithms (an approach that already has some “tradition” in the area), GRASP shows that it has potential to become an additional useful technique, for solving the UC problem. In fact, if neighbourhood operations that consider some problem characteristics are developed, the performance of this technique can certainly be improved.

Although a very general approach has been followed (with the aim of producing, in the future, a generic framework, capable of handling several problem variations and of tackling different objective functions), and no problem specific operators have been considered, final results show that GRASP is capable of correctly handling this problem, leading to good results for the instances considered. Even if it did not reach the best results presented in the literature, it behaved particularly well, considering the problem instances’ structure.

Besides, being a very general approach, it has the advantage of being easily adapted to different variants of a UC problem. When necessary, additional tools that can help the searching process, may be added to the framework, with no changes in the main core of the algorithm. New information and objectives, related to the aims of the different actors present in the decision process, can therefore be incorporated in the algorithm with no important changes in its structure.

5 Conclusions

In this work, GRASP was used to solve the Unit Commitment problem. Final results show that using meta-heuristics like GRASP, that take into account decisions made in previous iterations for choosing the current movements, is a positive way of tackling true multi-period problems. As shown in the paper, the results obtained by performing several computational tests in instances from the literature, lead us to say that the method is robust and efficiently reaches good results.

Further work will focus on including hydro units in the problem. The authors will also adapt the current model to a market environment and study the behaviour of this approach for problems that reflect the particular features of such environment.

Under market conditions, and following the research priorities specified in the 99'DIMACS Workshop ("The next generation of Unit Commitment models"), presented in [5], particular attention will also be given to multiobjective versions of the problem that can include several objectives (e.g. take or pay contracts can be modeled as objective functions) and therefore allow the Decision Maker to consider further important information in the decision process.

Acknowledgements: The authors would like to thank

Prof. Kazarlis and Prof. Bakirtzis, for making available detailed information about the instances presented in this paper.

REFERENCES

- [1] A.J. Wood and B. F. Wollenberg. *Power Generation Operation and Control*. Wiley and Sons, 1996.
- [2] S. Sen and D. P. Kothari. Optimal thermal generating unit commitment: a review. *Electrical Power and Energy Systems*, 20:443–451, 1998.
- [3] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.
- [4] S.A. Kazarlis, A.G. Bakirtzis, and V. Petridis. A genetic algorithm solution to the unit commitment problem. *IEEE Transactions on Power Systems*, 11:83–92, 1996.
- [5] B. F. Hobbs, M. H. Rothkopf, R.P. O'Neill, and H-P Chao. *The Next Generation of Electric Power Unit Commitment Models*. Kluwer Academic Publishers, 2001.
- [6] G.B. Sheble. *Computational Auction Mechanisms for Restructured Power Industry Operation*. Kluwer Academic Publishers, 1999.
- [7] C-P Cheng, Liu C-W, and Liu C-C. Unit commitment by lagrangian relaxation and genetic algorithms. *IEEE Transactions on Power Systems*, 15(5):707–714, 2000.
- [8] J. F. Bard. Short-term scheduling of thermal-electric generators using lagrangian relaxation. *Operations Research*, 36(5):756–766, 1988.