Nash equilibrium as the minimum of a function. Application to electricity markets with large number of actors.

E.V. Beck[*], R. Cherkaoui, *Senior Member, IEEE,*[†], A. Minoia[‡] and D. Ernst[§]
[*] EPFL-STI-LRE, Switzerland, Email: elena.vdovina@epfl.ch
[†] EPFL-STI-LRE, Switzerland, Email: rachid.cherkaoui@epfl.ch
[‡] Email: anna.minoia@gmail.com
[§] Supélec, France, Email: damien.ernst@supelec.fr

***Abstract*—We introduce in this paper a new approach for efficiently identifying Nash equilibria for games composed of large numbers of players having discrete and not too large strategy spaces. The approach is based on a characterization of Nash equilibria in terms of minima of a function and relies on stochastic optimization algorithms to find these minima. The approach is applied to compute Nash equilibria of some electricity markets and, based on the simulation results, its performances are discussed.**

***Index Terms*—Electricity market, Nash equilibrium computation, combinatorial optimization**

I. Introduction

Game theory studies decisions that are made in an environment where various players interact. The most widely ”solution concept” in game theory is called Nash Equilibrium (NE), after Nobel Laureate in economics and mathematician John Nash. A Nash equilibrium in games represents a joint strategy with the property that no player can benefit by changing his strategy while the other players keep their strategy unchanged.

This concept of Nash equilibrium has been vastly adopted to analyze electricity markets. In particular, references [3], [5], [7], [8], [9], [13], [15], [16], [18], [20], [21] are only a small sample of the scientific papers which rely on some NE concepts to model the behavior of the different actors of an electricity market.

We address in this paper the problem of computation of pure Nash equilibria for electricity markets having a large number of players and for which the players have a discrete set of strategies. Identification of pure NE in such type of games can be done in principle by relying on an exhaustive search process which consists of checking whether every single joint strategy corresponds to a Nash equilibrium. However, the number of operations that are required for identifying with such a process the NEs of a game grows exponentially with the number of players, making such an approach rapidly computationally impractical.

In this paper, we introduce a new approach for efficiently identifying Nash equilibria for such type of games. The approach is based on an alternative characterization of the NEs of a game in terms of minima of a function defined over the joint strategy space. With such a characterization, the problem of finding a sample Nash equilibrium is transformed into a pure combinatorial optimization problem:

$$u^* = \arg\min_{u \in U} D(u) \tag{1}$$

where D is a cost function and U the joint strategy space. Solving such a combinatorial problem by simple enumeration has still a complexity which grows exponentially with the number of players. However, by writing the NE search problem under this form, one can exploit state-of-the-art stochastic optimization algorithms, such as genetic algorithms [12], simulated annealing [1], ant colony optimization [10], tabu search [11] or nested partitioning [19], to curb this exponential computational growth with the number of actors.

We have used Genetic Algorithms (GAs) to solve the combinatorial problems and validated the approach on a problem of computation of NEs for an electrical spot market modeled as a normal-form game.

Simulations results have shown that, even when the number of power producers bidding to the spot market is large, the approach is still able to identify NEs within reasonable computing times. In particular, we have been able to identify, within a few minutes of computation time, NEs for a spot market composed of 30 power producers and where each power producer can choose between at least 20 different bidding strategies[1].

While this alternative characterization of NEs is not unique to this paper (see e.g. [17] where several alternative characterizations of Nash equilibria are proposed), this paper is however the first one which uses this characterization together with stochastic optimization algorithms. Also, to our best knowledge, this characterization of NEs as the minima of a function has not been exploited before in the electricity market literature.

[1]This leads to a space of combined strategies composed of more than 30^{20} elements !

‡ Anna Minoia works for an electricity trading company and collaborated with the other authors for this work. This paper does not necessarily reflect the viewpoint of her company. Any error or omission is the sole responsibility of the authors.

The paper is organized as follows. In the next section (Section II), we introduce in the context of normal-form game a characterization of Nash equilibria in terms of minima of a function and describe a methodology, built upon this characterization, to identify multiple Nash equilibria of a game. In Section III, we describe the stochastic combinatorial algorithm used in our simulations. Section IV gathers the simulation results. We underline that this section will only present results related to the computation of one Nash equilibrium of a game. Finally, Section V concludes and gives directions for future research.

II. Game theory and Nash equilibrium

A. The classical definition of a pure Nash equilibrium

We consider a normal-form game G whose description is the following. The game is composed of N players $\{1, 2, \ldots, N\}$. Each player i can play a strategy $u_i \in U_i$ where U_i is supposed to be a finite set. Let $u = (u_1, u_2, \ldots, u_N) \in U$ denote the players' combined strategy and $J_i(u)$ the payoff of a player i if the combined action u is played.

In such a context, the classical definition of a pure Nash equilibrium for G is:
The combined strategy $u^* = (u_1^*, \ldots, u_N^*)$ is a Nash equilibrium for G if for all $i \in \{1, 2, \ldots, N\}$ and for all $u_i \in U_i$ we have

$$J_i(u_1^*, \ldots, u_N^*) \geq J_i(u_1^*, \ldots, u_{i-1}^*, u_i, u_{i+1}^*, \ldots, u_N^*) \qquad (2)$$

B. Nash equilibrium as the minimum of a function

We introduce hereafter a function D defined over the joint strategy space U which, as shown later in Theorem 2.1, is always positive when $u \in U$ is not a NE and zero otherwise.

The function $D(u) : U \to \mathbb{R}^+$ is defined as follows:

$$D(u) = \qquad (3)$$
$$\sum_{i=1}^{N} [\max_{u_i' \in U_i} J_i(u_1, \ldots, u_{i-1}, u_i', u_{i+1}, \ldots, u_N) - J_i(u)]$$

We have the following theorem:
Theorem 2.1: The function D is strictly positive if the combined strategy u is not a Nash equilibrium and equal to zero otherwise.

The proof of Theorem 2.1 follows directly from the classical definition of a Nash equilibrium.

As direct consequence of this theorem, we can say that, in the presence of NEs, the function D has a number of global minima equal to the number of NEs.

C. A methodology for identifying multiple NEs in a game

In principle, by using a combinatorial algorithm able to identify all the minima of the function D, we could compute all the NEs of the generic game G. However, when using genetic algorithms as optimization algorithms, we found out by carrying simulations on our benchmark test problem that, in the presence of multiple Nash equilibria, they were converging with a high probability to the same equilibrium. Therefore, we suggest to adopt for identifying multiple NEs an approach which interlaces the resolution of combinatorial optimization problems with appropriate penalization of the function D. Each time a Nash equilibria u^* is found by the optimization algorithm, a neighborhood $\mathcal{N}$ of u^* is defined and strategies in this neighborhood are penalized by adding a large positive value $penalty$ to the function D (i.e., if $u \in \mathcal{N}$ then $D(u) \leftarrow D(u) + penalty$). Once the function D has been penalized, the combinatorial optimization algorithm is run again. Similar penalization schemes could also been used to mitigate the effects of local minima. Since the number of NEs is usually not known, we could for example stop the NE search process when the genetic algorithms converge several times in a raw to non-zero values of the function D.

III. Genetic Algorithm

We will in our simulation result section rely on some genetic algorithms to solve the optimization problem (1) where $D(u)$ is defined by Eqn (3).

The first paragraph of this section gives a general description of the genetic algorithms. The subsequent paragraphs define specific elements associated with the genetic algorithms used in our simulations. These are the fitness function, the genetic representation which is also called coding scheme and the genetic operators.

A. General description

Genetic algorithms are a class of heuristic search methods and computational models of adaptation and evolution based on natural selection. They became a widely recognized optimization method as a result of the work of John Holland in the early 1970s [14].

Genetic algorithms are implemented as a computer simulation in which a population of abstract representations (called chromosomes or the genotype or the genome) of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem evolves iteratively toward better solutions.

At each iteration, one uses three main operators (reproduction, crossover and mutation) to create a new population of candidate solutions whose performances are, in principle, better that those of the previous one.

A typical genetic algorithm requires three things to be defined:

- a genetic representation of the elements of the search domain
- a fitness function to evaluate the individuals of a population
- the genetic operators

B. Genetic representation of an element $u \in U$

In our simulations, we have used as representation of an element of the search domain U, a string of bits of length m, which is the standard representation [6].

Each component of a joint action $u \in U$ corresponds to a segment of the string of bits. Such a segment is named gene and its length is equal to $length\_gene$. Since an element of u has N components, the number of players in the game, we have therefore: $m = N \times length\_gene$.

Let s denote a binary string of length m and s_b the bth component of this string. Let S denote the set of all binary strings of length m.

Let C be the binary coding function and C^{-1} the decoding function. More specifically, $C(u)$ gives the binary string which codes u and $C^{-1}(s)$ gives the element of U to which the binary string s corresponds. We assume that:

$$C^{-1}(C(u)) = u \quad \forall u \in U. \tag{4}$$

In our test problems, every component of u corresponds to an integer. We have chosen as coding function C a standard approach for coding an integer into a string of bits. More specifically, we have chosen as a function C^{-1} defined by:

$$(C^{-1}(s))_i = \sum_{b=1+length\_gene\times(i-1)}^{length\_gene\times i} s_b \times 2^{b-1-length\_gene\times(i-1)} \tag{5}$$

$\forall i \in \{1, 2, \ldots, N\}$. From Eqn (5) one can deduce in a straightforward way C by exploiting Eqn (4).

In our simulations, every component u_i of u corresponds to a price equal to an integer number of dollars and this price has been coded by using a 7 bit string ($length\_gene = 7$).

C. Fitness function

The $fitness$ function is a function defined over S and gives 'the quality' of an individual s of the population. We have chosen this function equal to:

$$fitness(s) = -D(C^{-1}(s)) \tag{6}$$

With such a choice, the 'fitter' an individual, the 'closer' it stands from a Nash equilibrium, at least if we assume that the notion of distance between a element $u \in U$ and its closest Nash equilibrium is given by $D(u)$.

D. The operators

Let us denote by $P(t)$ the population at iteration t. The size of the population remains constant whatever the iteration t and can therefore be described throughout the iterations by a set of K individuals $\{s^1, s^2, \ldots, s^K\}$.

The genetic algorithm starts with an initial population $P(0)$. Each element of this population is the binary representation of an element u chosen at random in U.

At iteration $t \geq 0$, the algorithm creates an empty set $P(t+1)$ and uses sequentially the reproduction operator, the crossover operator and the mutation operator to fill this set with K new individuals. Usually, the quality of the individuals tends to increase with t, $-$ that is the fitness of individuals of $P(t)$ increase when t grows. In our simulation results section, we stopped only the algorithm when an individual s of $P(t)$ was such that $fitness(s) = 0$, $-$ that is when $C^{-1}(s)$ is a Nash equilibrium of the game.[2]

*1) **Reproduction operator**:* The reproduction operator selects with replacement $nb^{reproduction}$ individuals from $P(t)$. This operator is such that the chances that an individual s has to be selected grow with its fitness value ($fitness(s)$). More specifically, the operator will repeat $nb^{reproduction}$ the following sequence of instructions: (i) select an individual in $P(t)$ such that the probability of selecting s_j is equal to

$$p(s^j) = \frac{fitness(s^j)}{\sum_{s \in P(t)} fitness(s)} \tag{7}$$

(ii) copy this individual and add it to the population $P(t+1)$.

In our simulations, $nb^{reproduction}$ is always chosen equal to 2 if K is an even number and 1 otherwise.

*2) **Crossover operator**:* The crossover operator performs a partial exchange of characteristics (genetic material) between two individuals selected randomly from the current population and create from this exchange a 'new' individual which inherits the characteristics of both 'parents'.

This operator repeats $nb^{crossover}$ times the following instructions: (i) draw with replacement two strings $s1$ and $s2$ from $P(t)$ according to the probability distribution defined by Eqn (7) (ii) build two new strings ($ns1$, $ns2$) according to the following rule:

$$\begin{cases} \left.\begin{array}{l} ns1 = s1(1, cp) \oplus s2(cp+1, m) \\ ns2 = s2(1, cp) \oplus s1(cp+1, m) \end{array}\right\} & \text{with probability } p_{co} \\ \left.\begin{array}{l} ns1 = s1 \\ ns2 = s2 \end{array}\right\} & \text{with probability } 1 - p_{co} \end{cases} \tag{8}$$

where cp is the crossing point selected at random and with uniform probability in $\{1, 2, \ldots, m\}$, $\oplus$ is a concatenation operator between strings and p_{co} the 'crossover probability' (iii) add these two individuals into $P(t+1)$.

In our simulations, we have modified in a straightforward way step (ii) in order to have crossover points which stand only in between genes represented here by groups of 7 bits.

The crossover probability p_{co} is equal to 0.8 in our simulations while the value of $nb^{crossover}$ is chosen equal to $\frac{K - nb^{reproduction}}{2}$.

*3) **Mutation operator**:* The mutation operator introduces random modifications in the population. These modifications help to preserve the diversity and prevent the algorithm from premature convergence. The mutation operator is the last operator to be used at iteration t. For every individual s of $P(t{+}1)$, except the $nb^{reproduction}$ individuals generated by the reproduction operator, the algorithm chooses with probability p_{mo} whether to apply a mutation to the individual s of $P(t{+}1)$. If yes it selects at random a bit of the string s and changes its value.

[2]Without entering into the details, it can be shown that if a NE indeed exists, then with the operators we have adopted, our GA algorithm will identify with probability one a Nash equilibrium.

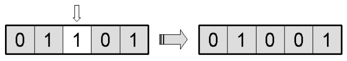

Fig. 1. Mutation operator.

In our simulations we wanted to ensure that after applying a mutation to an individual, the individual was still corresponding to an element of U. To do so, rather than to select at random a bit, we selected at random a gene i corresponding to the action u_i. Then, we selected an element at random in U_i, we converted it into a string by using the function $C(\cdot)$ and, finally, we replaced the gene i by this string.

The value of p_{mo} is chosen equal to 0.2 in our simulations.

IV. Simulation results

We assess in this section the performances of our approach when applied to the computation of one Nash equilibrium of an electricity market. We stress that, as explained in Section II-C, the approach can in principle be applied to the computation of several Nash equilibria of a game by using appropriate penalization schemes. However, since we found out that the penalization schemes we developed were not yet mature enough, we preferred to focus in this section on the computation of only one Nash equilibrium.

The first paragraph of this section details the application of our approach to the computation of one Nash equilibrium of a market having only two power producers and has mainly a didactic purpose.

In the second paragraph, we study for an electricity market whose number of generators depends on a parameter N, the performances of our approach.

A. An illustrative example

We illustrate our approach on a problem of computation of one Nash equilibrium for a spot market where two generators (generator 1 and generator 2) bid strategically to maximize their profits.

Both generators have a constant marginal production cost and a limited production capacity. These values are defined in Table I where the symbol CM_i denotes the marginal production cost of generator i and Q_i^{max} its maximum production capacity.

We assume a uniform-price spot market with a price cap of 50\$/MWh and an elastic load. The equation $Pr_{market} = -0.083 \times Q_{tot} + 58.33$ gives the amount of power Q_{tot} the load is ready to buy at a price per MWh equal to Pr_{market}. Bids are submitted in the form of "price per MWh"-"quantity" pairs. For the sake of simplicity, we consider here that the generators always bid their full capacity. The strategy spaces U_1 and U_2, which are defined in Table I, are therefore composed of elements which correspond to prices per MWh.

The profit (or payoff) of a generator i is computed by using the following expression:

$$J_i(u) = Q_i \times Pr_{market} - Q_i \times CM_i \qquad (9)$$

where Pr_{market} is the market clearing price and Q_i the electrical power the generator i is scheduled to produce.

	Q_i^{max} [MW]	CM_i [\$/MWh]	U_i [\$/MWh]
Gen. 1	200	25	$\{25, 26, \ldots, 50\}$
Gen. 2	300	30	$\{25, 26, \ldots, 50\}$

TABLE I
Generation Data and Price Strategy Space

This process of interaction between the two generators through the spot market can be modelled as a normal form game. The function D (see Eqn (3)) associated with this normal form game is drawn of Fig. 2. As we may observe, this function has seven global minima. The function $D(u)$ is equal to 0 when u corresponds to one of these global minima. They correspond therefore, by virtue of Theorem 2.1, to Nash equilibria.

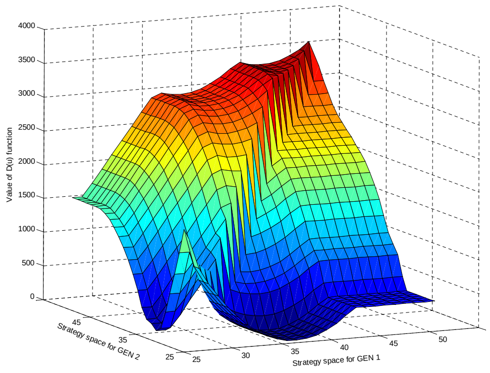

Fig. 2. A plot of the function D defined by Eqn (3).

Since the joint strategy space is here rather small (26×26 elements), there is certainly no need to use combinatorial algorithms for computing the minima of $D(\cdot)$. However, for didactic purpose, we have run on this example the genetic algorithm with a population of ten individuals. The evolution of the population in the joint strategy space U is drawn on Fig. 3. As we observe, all the individuals coincide at iteration 72 with an element $u = (u_1, u_2) = (27, 36)$ which is actually a Nash equilibrium. Actually, this Nash equilibrium was already found before iteration 72 by our approach and the algorithm could therefore have been stopped earlier.

B. Performances of the approach for markets with large joint strategy spaces

We now evaluate the performances of our approach when applied to markets having many actors.

First, we have set up a procedure which takes as input the number of actors N and defines automatically the data of a uniform-price spot market which has N generators that can bid strategically. We note that the market is cleared by computing the intersection between the supply and the demand curves. Then, we have by using this procedure generated markets of various sizes and run our algorithm to compute

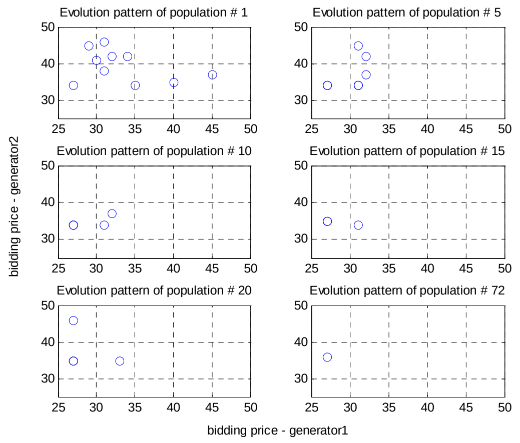

Fig. 3. Representation of the population $P(t)$ at different iterations t (1, 5 10, 15, 20, 72).

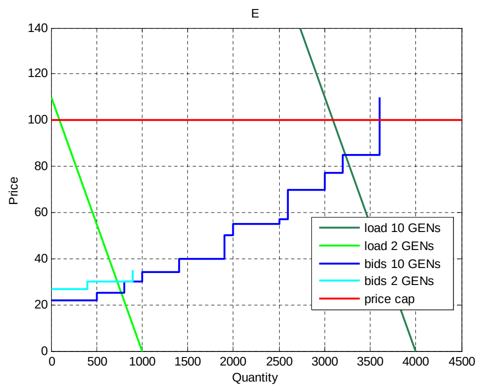

Fig. 4. Some typical elastic load demand curves and supply curves.

a Nash equilibrium for every of these markets. Finally, we have analyzed the different runs of our algorithm.

Procedure used to generate the data of a N actor electricity market. The marginal cost (maximum production Q_i^{max}) of a generator i is determined by drawing at random and with uniform probability a value in the set $\{20, 21, 22, \dots, 30\}$ ($\{100, 110, \dots, 500\}$). Its strategy space U_i is set equal to $\{CM_i, CM_i + 1, \dots, price\_cap\}$ where $price\_cap$ is the value of the market price cap. This value is chosen equal to 50 \$/MWh.

The market has an elastic load demand which responds to the following equation:

$$Pr_{market} = k1 \times Q_{tot} + b1 \tag{10}$$

where Pr_{market} is the market price, Q_{tot} is the quantity of power the load is ready to buy for a price Pr_{market}, $k1$ is a parameter chosen equal to -0.11 and $b1$ is a parameter whose value depends on the total installed capacity. This dependence has been introduced to have an elastic load demand which is correlated with the size of the generation park. More specifically, we have chosen this coefficient $b1$ equal to $-k1 \times 0.8 \times Q_{inst}$ where $Q_{inst} = \sum_{i=1}^{N} Q_i^{max}$.

Figure 4 represents for several values of N some typical elastic load demand curves and supply curves.

Simulation results. By using the procedure described previously, we have generated for various values of N the data of different markets and run on each of this market our algorithm to compute a Nash equilibrium. Figure 5 reports the simulation results obtained by this procedure as well as those obtained by an approach relying on an exhaustive search process to find all the Nash equilibria of the system.

We explain hereafter the content of this figure.

Column I: Number of generators in the market (N). This number ranges from 2 to 30.

Column II: Total number of NEs in the market. These NEs have been found by relying on an exhaustive search process. We note that the total number of NEs is only reported for values of N smaller or equal to 4. Determining the number of NEs for larger values of N was leading to too high computing times.

Column III: CPU time needed to carry out this exhaustive search. The number between brackets reports the value of this CPU time divided by the number of NEs in the game. This number can in some sense be seen as the average time needed to compute a NE by relying on an exhaustive search process.

Column IV: Average CPU time needed by the genetic algorithm to find a minimum of $D(\cdot)$, i.e. to find a NE. The genetic algorithm has been run with a population size $K = 40$.

Column V: Average number of iterations needed by the genetic algorithm to find a minimum of $D(\cdot)$.

The values of Columns IV and V have been computed by averaging the results obtained over 10 runs of our algorithm.

Analysis of the simulation results. Different observations can be drawn from Figure 5. At first, one should note that the number of NEs grows with the number of generators N. At second, one should observe that the time required to find the NEs by exhaustive search grows extremely rapidly with N, making this approach rapidly too computationally demanding when N increases. Moreover, an analysis of the fourth column of this table shows that the average CPU time for finding a NE with our approach does not grow 'too rapidly' with N. This allows us to tackle much larger problems than with the exhaustive search approach. It should be noted that the growth in CPU times is the combination of two factors: (i) the increase with N of the CPU time required to carry out one iteration of our algorithm (ii) the growing number of iterations that are needed for identifying a NE when N increases, as illustrated in the last column of the table.

It is clear that the performances of our approach depend on the different parameters of the GA algorithm. For example, we found out that by using $K = 10$ rather than $K = 40$, the algorithm was performing better for small values of N. However, when N was greater than 10, it was preferable to work with $K = 40$. Numerous papers have addressed the problem of fine tuning of some stochastic optimization algorithms (see, e.g. [2], [4]). We refer the reader to them for a complement of information about this subject.

Number of GENs (N)	Number of NEs	CPU Time for finding every (a) NE by exhaustive search	Average CPU for finding a NE (GA based approach)	Average number of iter. for finding a NE (GA based approach)
2	3	85.78 (28.60)	12.058	14.3
3	6	5025.42 (837.57)	19.948	14.667
4	16	296102.31 (18506.39)	13.579	14.8
5	/	/	24.15	17.5
10	/	/	33.297	18.1
20	/	/	133.858	16.5
30	/	/	2234.415	53.5

Fig. 5. An analysis of the performances of an exhaustive search approach which computes every Nash equilibrium and of the GA based approach which computes a Nash equilibrium by finding a zero of $D(\cdot)$. The CPU times are given in seconds on an Intel Pentium 4 2.40 GHz processor.

V. Conclusions

We have proposed and evaluated in this paper a new approach for computing Nash equilibria for games having a large number of actors. While standard approaches to this problem rely on some best-response type strategies, the proposed methodology was reformulating the problem as a standard minimization problem and using stochastic optimization algorithms to solve this problem. In principle, the methodology is also able to identify every equilibrium of the game by using an iterative scheme together with a proper penalization of the cost function.

We validated our approach on electricity markets having an elastic load demand and found out that the approach was able to identify with 'reasonable computing times' a Nash equilibrium of a game, even when dealing with a large number of power producers. Some side simulations showed us that designing an appropriate penalization scheme for identifying multiple Nash equilibria was however challenging. Therefore, we propose as first research direction to design well-performing penalization schemes able to identify in an efficient way every Nash equilibrium of a game or at least a high percentage of them.

As second research direction, we suggest to extend our approach to games where the actors have strategy spaces described by continuous or mixed integer-continuous variables. While the philosophy behind the extension is straightforward, we expect that state-of-the art optimizers may run into difficulties when solving the corresponding minimization problem since it is, among others, generally non-convex.

Finally, we underline that it would be pertinent to compare the performances of our approach with those of some other approaches for computing Nash equilibria. In this respect, it would certainly be interesting for the power system community to define a library of benchmarks for problems of computation of Nash equilibria for electricity markets, something we found out was missing.

References

[1] E.H.L. Aarts and J.H.M. Korst. Simulated Annealing and Boltzmann Machines. *John Wiley & Sons*, 1989.

[2] B. Adenso-Diaz and Manuel Laguna. Fine-tuning of algorithms using fractional experimental design and local search. *Operation Research*, 54(1):99–114, 2006.

[3] C.A. Berry, B.F. Hobbs, W.A. Meroney, and R.P. O'Neill. Understanding how market power can arise in network competition: a game theoretic approach. *Utility Policy*, 8(3):139–158, September 1999.

[4] M. Blesa and F. Xhafa. Using parallelism in experimenting and fine tuning of parameters for metaheuristics. In *Computational science - ICCS 2004*, volume 3036 of *Lecture Notes in Computer Science*, pages 429–432. Springer Berlin / Heidelberg, 2004.

[5] D. Chattopadhyay. Multicommodity spatial sournot model for generator bidding analysis. *IEEE Transactions on Power Systems*, 19(1):267–275, February 2004.

[6] M. Coli, G. Gennuso, and P. Palazzari. A New Crossover Operator for Genetic Algorithms. *Proceeding of the IEEE International Conference on Evolutonary Computation ICEC 96*, pages 201–206, May 1996.

[7] J. Contreras, M. Klusch, and J. B. Krawczyk. Numerical solutions to Nash Cournot equilibria in coupled constraint electricity markets. *IEEE Transactions on Power Systems*, 19(1):195–206, February 2004.

[8] Pedro F. Correia, Thomas J. Overbye, and Ian A. Hiskens. Searching for noncooperative equilibria in centralized electricity markets. *IEEE Transactions on Power Systems*, 18(4):1417–1424, November 2003.

[9] Lance B. Cunningham, Ross Baldick, and Martin L. Baughman. An empirical study of applied game theory : transmission constrained Cournot behavior. *IEEE Transactions on Power Systems*, 17(1):166–172, February 2002.

[10] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, 26(1):26–41, 1996.

[11] F. Glover. Tabu search - part ii. *ORSA Journal on Computing*, 2:4–32, 1990.

[12] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[13] R. Green and D. M. Newbery. Competition in the British electricity spot market. *Journal of Political Economy*, 100(5):929–953, October 1992.

[14] J. H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, MA, USA, 1992.

[15] Kwang-Ho Lee and Ross Baldick. Solving three - player games by the matrix approach with application to an electric power market. *IEEE Transactions on Power Systems*, 18(4):1573–1580, Nov. 2003.

[16] Kwang-Ho Lee and Ross Baldick. Tuning of discretization in bimatrix game approach to power pystem market analysis. *IEEE Transactions on Power Systems*, 18(2):830–836, May 2003.

[17] R.D. McKelvey and A. McLennan. *Handbook of computational economics*, volume 1, chapter Computation of equilibria in finite games, pages 87–142. North-Holland, 1996.

[18] A. Minoia, D. Ernst, M. Dicorato, M. Trovato, and M. Ilic. Reference transmission network: a game theory approach. *IEEE Transactions on Power Systems*, 21:249–259, February 2006.

[19] L. Shi and S. Olafsson. Nested partitioning for global optimization. *Operations Research*, 48(3):390–407, 2000.

[20] You Seok Son and Ross Baldick. Hybrid coevolutionary programming for Nash equilibrium search in games with local optima. *IEEE Transactions on Evolutionary Computation*, 8(4):305–315, August 2004.

[21] W. Xian, L. Yuzeng, and Z. Shaohua. Oligopolistic equilibrium analysis for electricity markets: a nonlinear complementarity approach. *IEEE Transactions on Power Systems*, 19(3):1348–1355, August 2004.