



# Nash Equilibrium By Using A Dang's Fixed Point Algorithm

SONIYA PATEL<sup>1</sup> DR. R. S. PATEL<sup>2</sup>

<sup>1</sup>(Research Scholar) Department of Mathematics, Govt. Autonomous P.G. College Satna (M.P.) India.

<sup>2</sup> (Supervisor) Department of Mathematics, Govt. Autonomous P.G. College Satna (M.P.) India.

**Abstract**— A distributed implementation of Dang's Fixed Point algorithm is proposed for searching one Nash equilibrium of a finite  $n$  –person game in normal form. In this paper, the problem consists of two subproblems. One is changing the problem form to a mixed 0-1 linear programming form. This process is derived from applications of the properties of pure strategy and multi linear terms in the payoff function. The other subproblem is to solve the 0 – 1 linear programming generated in the former subproblem. A distributed computation network which is based on the Dang's Fixed-Point method is built to solve this 0 – 1 linear programming.

## INTRODUCTION

This paper is concerned with the distributed computation of one pure-strategy Nash equilibrium of a finite  $n$ -person game in normal form. To tackle this problem, Wu's method in [19], [20] is used to reformulate the Nash equilibrium problem to a mixed 0-1 linear programming form by exploiting the properties of pure strategy and multilinear terms in the payoff functions. One feasible solution of the mixed 0-1 linear program yields one pure-strategy Nash equilibrium. In the next step, a distributed computation network is built based on Dang's Fixed-Point algorithm [21], [22] to solve the mixed 0-1 linear programming. Numerical results show that the distributed method is promising and it can be easily extended to other NP-hard problems. In distributed computation network, a problem is divided into many subproblems, each of which can be solved in different computers which communicate with each other by message passing. The computation speed is influenced by the number of the computers in the network. Distributed models can be classified into simple model and interactive model, which are illustrated in Fig.1 and Fig.2 respectively

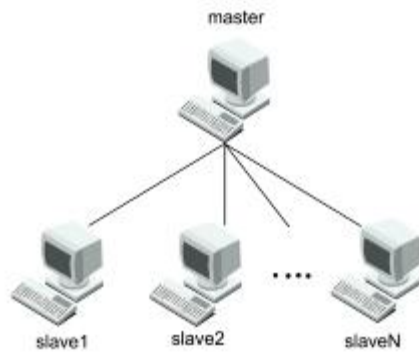


Figure 1. The simple distributed model.

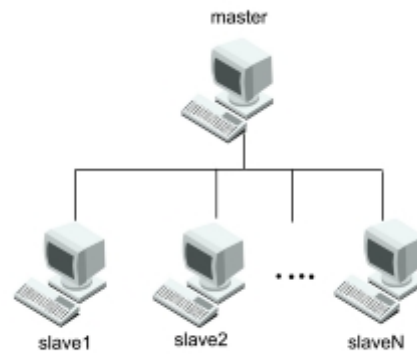


Figure 2. The interactive distributed model.

The first model is called simple distributed model. In this model, all the slave computers can only communicate with the master computer. The task is scheduled and divided into several smaller subproblems by the master computer. And then each subproblem is sent to a different slave computer by the master computer. Finally, the master computer receives the result from the slaves and output it. Compared with the simple model, the interactive model is more advanced. In this model, the slave computers also could communicate with each other by message passing. Therefore, if one slave computer finishes its work, it can help others. The details of the distributed computation will be presented in the section IV.

### CHANGING THE SEARCHING NASH EQUILIBRIUM PROBLEM TO A MIXED 0-1 LINEAR PROGRAMMING FORM

Let  $N = \{1, 2, \dots, n\}$  be the set of players. The pure strategy set of player  $i \in N$  is denoted by  $S^i = \{s_j^i \mid j \in M_i\}$  with  $M_i = \{1, 2, \dots, m_i\}$ .

Given  $S^i$  with  $i \in N$ , the set of all pure strategy profiles is  $S = \prod_{i=1}^n S^i$ .

We denote the payoff function of player  $i \in N$  by  $u^i : S \rightarrow R$ . For  $i \in N$ , let  $S^{-i} = \prod_{k \in N \setminus \{i\}} S^k$ . Then,  $s = (s_{j_1}^1, s_{j_2}^2, \dots, s_{j_n}^n) \in S$  can be rewritten as  $s = (s_{j_i}^i, s^{-i})$  with  $s^{-i} = (s_{j_1}^1, \dots, s_{j_{i-1}}^{i-1}, s_{j_{i+1}}^{i+1}, \dots, s_{j_n}^n) \in S^{-i}$ .

A mixed strategy of player  $i$  is a probability distribution on  $S^i$  denoted by  $x^i = (x_1^i, x_2^i, \dots, x_{m_i}^i)$ .

Let  $X^i$  be the set of all mixed strategies of player  $i$ . Then,  $X^i = \{x^i = (x_1^i, x_2^i, \dots, x_{m_i}^i) \in R_+^{m_i} \mid \sum_{j=1}^{m_i} x_j^i = 1\}$ . Thus, for  $x^i \in X^i$ , the probability assigned to pure strategy  $s_j^i \in S^i$  is equal to  $x_j^i$ .

Given  $X^i$  with  $i \in N$ , the set of all mixed strategy profiles is  $X = \prod_{i=1}^n X^i$ . For  $i \in N$ , let  $X^{-i} = \prod_{k \in N \setminus \{i\}} X^k$ . Then,  $x = \{x^1, x^2, \dots, x^n\} \in X$  can be rewritten as  $x = (x^i, x^{-i})$  with  $x^{-i} = (x^1, \dots, x^{i-1}, x^{i+1}, \dots, x^n) \in X^{-i}$ .

If  $x \in X$  is played, then the probability that a pure strategy profile  $s = (s_{j_1}^1, s_{j_2}^2, \dots, s_{j_n}^n) \in S$  occurs is  $\prod_{i=1}^n x_j^i$ .

Therefore, for  $x \in X$ , the expected payoff of player  $i$  is given by  $u^i(x) = \sum_{s \in S} u^i(s) \prod_{i=1}^n x_j^i$ .

With these notations, a finite  $n$ -person game in normal form can be represented as  $\Gamma = (N, S, \{u^i\}_{i \in N})$  or  $\Gamma = (N, X, \{u^i\}_{i \in N})$ .

**Definition 1:** (Nash, [1]) A mixed strategy profile  $x^* \in X$  is a Nash equilibrium of game  $\Gamma$  if  $u^i(x^*) \geq u^i(x^i, x^{*-i})$  for all  $i \in N$  and  $x^i \in X^i$ . With the application of optimality condition, one can obtain that  $x^*$  is a Nash equilibrium if and only if there are  $\lambda^*$  and  $\mu^*$  together with  $x^*$  satisfying the system of (1)

$$\left( \begin{array}{l} u^i(s_j^i, x^{-i}) + \lambda_j^i - \mu_i = 0, \\ e^{i\top} > x^i - 1 = 0, \\ x_j^i \lambda_j^i = 0, \\ x_j^i \geq 0, \lambda_j^i \geq 0 \end{array} \right)_{j = \{1, 2, \dots, m_i\}, i = \{1, 2, \dots, n\}} \quad (1)$$

where  $e^i = (1, 1, \dots, 1)^\top \in R^{m_i}$ .

Let  $\beta$  be a given positive number such that

$$\beta \geq \max_{i \in N} \left\{ \max_{s \in S} u^i(s) - \min_{s \in S} u^i(s) \right\}.$$

Then, (1) is equivalent to (2).

$$\left( \begin{array}{l} u^i(s_j^i, x^{-i}) + \lambda_j^i - \mu_i = 0, \\ e^{i\top} > x^i - 1 = 0 \\ x_j^i \leq v_j^i \\ \lambda_j^i \leq \beta(1 - v_j^i) \\ v_j^i \in \{0, 1\} \\ x_j^i \geq 0, \lambda_j^i \geq 0 \end{array} \right)_{j = \{1, 2, \dots, m_i\}, i = \{1, 2, \dots, n\}} \quad (2)$$

Thus, finding a pure-strategy Nash equilibrium is equivalent to computing a solution of the system of (3).

$$\left( \begin{array}{l} u^i(s_j^i, x^{-i}) + \lambda_j^i - \mu_i = 0 \\ e^{i\top} > x^i - 1 = 0 \\ x_j^i \leq v_j^i \\ \lambda_j^i \leq \beta(1 - v_j^i) \\ x_j^i \in \{0, 1\}, v_j^i \in \{0, 1\}, \lambda_j^i \geq 0 \end{array} \right)_{j = \{1, 2, \dots, m_i\}, i = \{1, 2, \dots, n\}} \quad (3)$$

where  $u^i(s_j^i, x^{-i}) = \sum_{s^{-i} \in S^{-i}} u^i(s_j^i, s^{-i}) \prod_{k \neq i} x_{j_k}^k$ .

One can find a pure strategy Nash equilibrium through solving the system (3). However, it is difficult to solve the system (3) directly because of the multilinear terms. To address this issue, Wu [19], [20] developed a method which can convert the system (3) to an equivalent mixed 0-1 linear formulation. This method is based on exploiting the properties of pure strategy and multilinear terms in the payoff functions.

Let  $y(s^{-i}) = \prod_{k \neq i} x_{j_k}^k$  for  $s^{-i} = (s_{j_1}^1, \dots, s_{j_{i-1}}^{i-1}, s_{j_{i+1}}^{i+1}, \dots, s_{j_n}^n) \in S - i$ .

Then,

$$u^i(s_j^i, s^{-i}) = \sum_{s^{-i} \in S^{-i}} u^i(s_j^i, s^{-i}) y(s^{-i}).$$

And the mixed 0 – 1 linear program obtained is shown in (4) as follows:

$$\left( \begin{array}{l} \sum_{s^{-i} \in S^{-i}} u^i(s_j^i, s^{-i}) y(s^{-i}) + \lambda_j^i - \mu_i = 0 \\ e^{it} > x^i - 1 = 0 \\ \lambda_j^i \leq \beta(1 - x_j^i) \\ x_j^i \in \{0, 1\}, \lambda_j^i \geq 0 \\ j = \{1, 2, \dots, m_i\}, i = \{1, 2, \dots, n\} \\ y(s^{-i}) \geq \sum_{h \neq i} x_{j_h}^h - (n - 2) \\ y(s^{-i}) \leq x_{j_k}^k, k \neq i \\ 0 \leq y(s^{-i}), s^{-i} \in S^{-i}, i = 1, 2, \dots, n \end{array} \right) \quad (4)$$

The derivation process of the formulation and the detailed proofs can be obtained in the paper [19]. Therefore, we can solve the mixed 0 – 1 linear program (4) to get a Nash equilibrium in (1). The distributed Dang's Fixed Point method which will be presented in the next sections plays well to the computation of this mixed 0-1 linear program.

#### DANG'S FIXED-POINT ITERATIVE ALGORITHM

Let  $P = \{x \in R^n | Ax + Gw \leq b, \text{ for some } w \in R^p\}$ , where  $A \in R^{m \times n}$  is an  $m \times n$  integer matrix with  $n \geq 2$ ,  $G \in R^{m \times p}$  an  $m \times p$  matrix, and  $b$  a vector of  $R^m$ .

Let  $x^{max} = (x_1^{max}, x_2^{max}, \dots, x_n^{max})^T$  with  $x_j^{max} = \max_{x \in P} x_j, j = 1, 2, \dots, n$  and  $x^{min} = (x_1^{min}, x_2^{min}, \dots, x_n^{min})^T$  with  $x_j^{min} = \min_{x \in P} x_j, j = 1, 2, \dots, n$ .

Let  $D(P) = \{x \in Z^n | x^l \leq x \leq x^u\}$ , where  $x^l = [x^{min}]$  and  $x^u = [x^{max}]$ . For  $z \in R^n$  and  $k \in N_0$ , let  $P(z, k) = \{x \in P | x_i = z_i, 1 \leq i \leq k, \text{ and } x_i \leq z_i, k + 1 \leq i \leq n\}$ .

Given an integer point  $y \in D(P)$  with  $y_1 > x_1^l$ , Dang and Ye [21], [22] developed a iterative method which is presented in Fig.3. It determines whether there is an integer point  $x^* \in P$  with  $x^* \leq_l y$ .

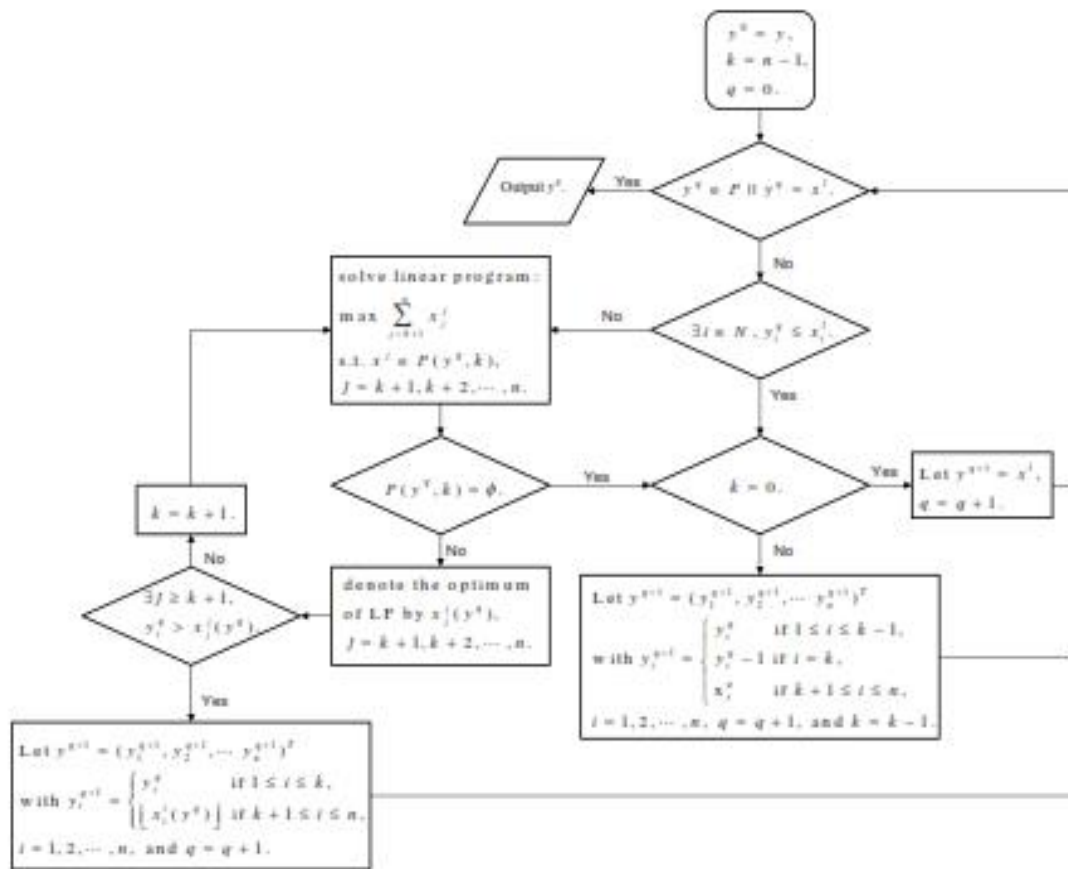


Figure 3. Flow diagram of the iterative method.

An example given below is used to illustrate the method. Consider a polytope  $P = \{x \in R^3 | Ax \leq b\}$  with

$$A = \begin{pmatrix} -1 & 0 & 2 \\ 0 & -2 & 1 \\ -1 & 0 & -2 \\ 1 & 1 & 0 \end{pmatrix}, b = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

It is easy to obtain  $x^u = (1, 0, 0)^T$  and  $x^l = (-1, -2, -2)^T$ . Let  $y = x^u, y^0 = y$ , and  $k = 3 - 1 = 2$ .  $y^1 = (1, -1, 0)$  can be obtained in the first iteration, and  $y^2 = (1, -1, -1)$  which is an integer point in  $P$  is obtained in the second iteration. An illustration of  $y^0, y^1$  and  $y^2$  can be found in Fig.4.

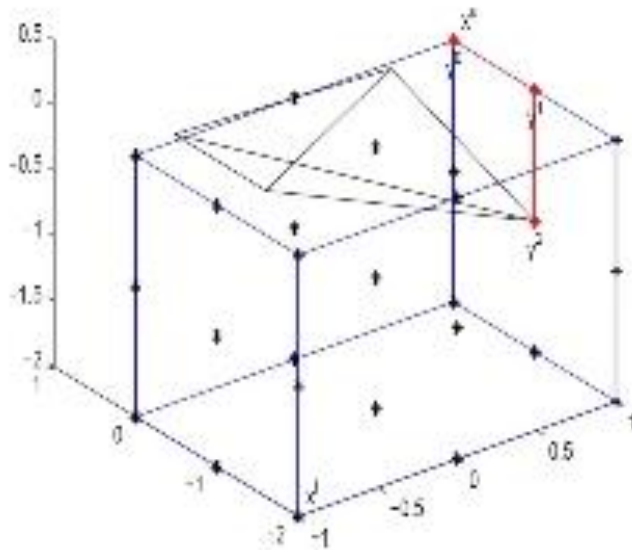


Figure 4. An illustration of the iterative method.

The idea of Dang's method [21], [22]  $y \in D(P)$  to solve integer programming is to define an increasing-mapping from a finite lattice into itself. All the integer points outside the  $P$  are mapped into the first point in  $P$  that is smaller than them in the lexicographical order of  $x^l$ . All the integer points inside the polytope are the fixed points under this increasing mapping. Given an initial integer point, the method either yields an integer point in the polytope or proves no such point exists within a finite number of iterations. For more details and proofs about this iterative method, one can consult Dang [21], [22].

## DISTRIBUTED IMPLEMENTATION

As an appeal feature, Dang's method can be easily implemented in a distributed way. Some distributed implementation techniques to Dang's method will be discussed in this section. The simple distributed model as described in Fig.1 has been used in our implementation. There are one master computer and a certain number of slave computers in this distributed computing system. The master computer takes charge of computing the solution space of the polytope, dividing the solution space to segments, sending the segments to the slave computers, receiving the computation result from the slave computers and exporting the computation result. Each slave computer receives the segment, judges whether there exists an integer point in its segment using Dang's Fixed-Point iterative method and sends its result to the master. The outline of the distributed computation process in this paper can be explained in the Fig.5.

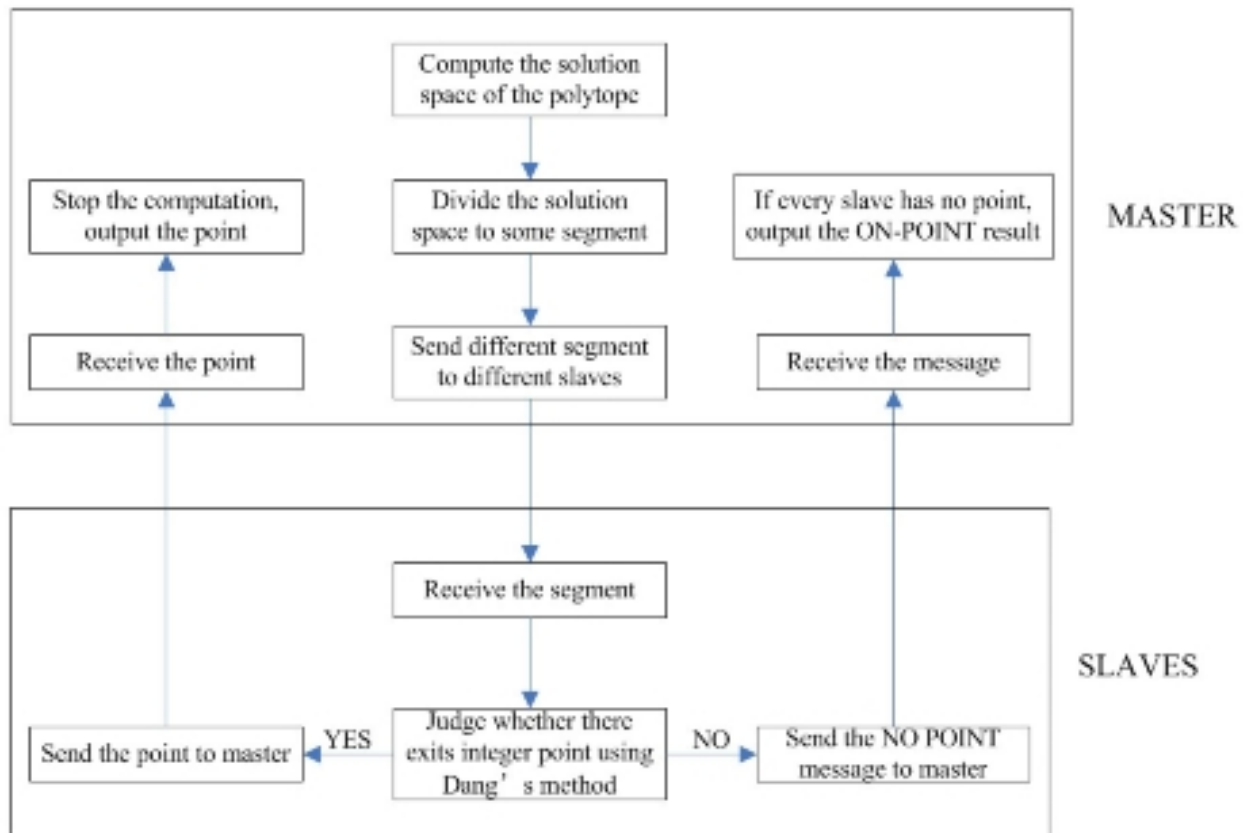


Figure 5. Flow diagram of the distributed computation process.

All the programs are coded in C++, and run on Microsoft Windows platform. Two algorithms have been considered to the bounding linear program in Dang's iterative method. One is the simplex algorithm [23] which is the most popular algorithm for linear programming. We can call the API function in CPLEX Concert Technology to carry out this method. The other algorithm is the self-dual embedding technique presented in [24]. This algorithm detects LP infeasibility based on a proved criterion, and it is the best method to solve the linear programming in Dang's algorithm to our knowledge. The MPICH2, which is a freely available, portable implementation of Message Passing Interface(MPI), is used to send and receive message between the master computer and slave computers in this implementation. More information about the Message Passing Interface can be obtained in the literature [25]. It is important to assign equal amount of work to each slave computer. Two methods have been implemented for the assignment. One is to divide the interested space into a number of more or less equal regions. The other is to randomly divide the interested space into a number of regions according to the Latin Squares method in [26]. The first method has a good performance when the number of slave computers is small. When the number of slave computers increases big enough, the later method performs better. The slave computers may complete their work allocations at different times. So the interactive distributed model(in Fig.2), in which the slave computers could communicate with each other by message passing, is our next work. By doing so, a slave computer can help others if it completes its work earlier. This will be quite helpful to enhance efficiency of the method.

## NUMERICAL RESULTS

In this section, some numerical results will be presented. The distributed computation network consists of 3 computers of OptiPlex 330 with 2 processors. All programs are coded in C++, and CPLEX Convert Technology is used to solve the linear programming in the Fixed-Point algorithm in this distributed network. Each subsegment divided by the master computer is independent to each other. Therefore, each slave computer can conduct each subsegment simultaneously. Message Passing Interface (MPI) is used to establish a communication network between the master computer to the slave computers. And the master computer takes charge of outputting the computation result. In the presentation of numerical results, some symbols are explained as follows.

*N*: The number of players in the instance.

*S*: The number of strategies for each player in the instance.

*Niteration*: The number of iteration of the Fixed-Point algorithm.

*Result*: “Yes” appears if the method finds a Nash equilibrium and “No” otherwise.

Only three-player game needs to be considered since any n-player game can be reduced to a three-player game in polynomial time as shown in [27]. In this paper, the computation examples which are generated randomly are given as follows.

**Example 1:** Consider a three-player game  $\Gamma = (N, S, \{u^i\}_{i \in N})$ , where  $N = \{1, 2, 3\}$ . The number of strategies for each player, Num *S*, is generated from 2 to 15 randomly. The  $\{u^i\}_{i \in N}$  are generated randomly too. There are four different ranges for  $\{u^i\}_{i \in N}$  in this example, which are from 0 to 1, from 0 to 10, from 0 to 50, and from 0 to 100. Let  $\beta = 1000$ . For different range of  $\{u^i\}_{i \in N}$ , 80 instances have been solved by this distributed computation network. Numerical results of this distributed computation network are given in Table I, Table II Table III and Table IV respectively.

Table I presents the numerical results when the payoff function  $u^i(s_j^i, s^{-i})$  is generated from 0 to 1 randomly.



| Pro | N | S  | Niteration | Result |
|-----|---|----|------------|--------|
| 1   | 3 | 2  | 23         | Yes    |
| 2   | 3 | 3  | 44         | Yes    |
| 3   | 3 | 3  | 44         | Yes    |
| 4   | 3 | 4  | 98         | Yes    |
| 5   | 3 | 4  | 71         | Yes    |
| 6   | 3 | 4  | 70         | Yes    |
| 7   | 3 | 5  | 104        | Yes    |
| 8   | 3 | 7  | 188        | Yes    |
| 9   | 3 | 9  | 433        | Yes    |
| 10  | 3 | 9  | 296        | Yes    |
| 11  | 3 | 9  | 296        | Yes    |
| 12  | 3 | 10 | 470        | Yes    |
| 13  | 3 | 10 | 491        | Yes    |
| 14  | 3 | 10 | 360        | Yes    |
| 15  | 3 | 11 | 429        | Yes    |
| 16  | 3 | 12 | 697        | Yes    |
| 17  | 3 | 13 | 781        | Yes    |
| 18  | 3 | 15 | 765        | Yes    |
| 19  | 3 | 15 | 1019       | Yes    |
| 20  | 3 | 15 | 770        | Yes    |

Table II gives the numerical results when the payoff function  $u^i(s_j^i, s^{-i})$  is generated from 0 to 10 randomly.

| Pro. | N | S  | Niteration | Result |
|------|---|----|------------|--------|
| 1    | 3 | 2  | 25         | No     |
| 2    | 3 | 2  | 15         | Yes    |
| 3    | 3 | 2  | 22         | Yes    |
| 4    | 3 | 3  | 51         | Yes    |
| 5    | 3 | 3  | 44         | Yes    |
| 6    | 3 | 4  | 80         | Yes    |
| 7    | 3 | 4  | 80         | Yes    |
| 8    | 3 | 5  | 124        | Yes    |
| 9    | 3 | 6  | 166        | Yes    |
| 10   | 3 | 7  | 203        | Yes    |
| 11   | 3 | 8  | 243        | No     |
| 12   | 3 | 8  | 274        | Yes    |
| 13   | 3 | 8  | 276        | Yes    |
| 14   | 3 | 9  | 303        | Yes    |
| 15   | 3 | 11 | 435        | Yes    |
| 16   | 3 | 11 | 446        | Yes    |
| 17   | 3 | 12 | 519        | Yes    |
| 18   | 3 | 13 | 625        | Yes    |

|    |   |    |     |     |
|----|---|----|-----|-----|
| 19 | 3 | 14 | 688 | Yes |
| 20 | 3 | 15 | 779 | Yes |

For the payoff function  $u^i (s_j^i, s^{-i} )$  is generated from 0 to 50 randomly, the numerical results are shown in Table III.

| Pro | . N | S  | Niteration | Result |
|-----|-----|----|------------|--------|
| 1   | 3   | 2  | 21         | Yes    |
| 2   | 3   | 4  | 74         | Yes    |
| 3   | 3   | 4  | 76         | Yes    |
| 4   | 3   | 4  | 73         | No     |
| 5   | 3   | 5  | 110        | Yes    |
| 6   | 3   | 6  | 179        | Yes    |
| 7   | 3   | 7  | 199        | Yes    |
| 8   | 3   | 8  | 244        | Yes    |
| 9   | 3   | 9  | 300        | Yes    |
| 10  | 3   | 10 | 424        | Yes    |
| 11  | 3   | 10 | 364        | Yes    |
| 12  | 3   | 11 | 496        | Yes    |
| 13  | 3   | 12 | 508        | No     |
| 14  | 3   | 13 | 589        | Yes    |
| 15  | 3   | 13 | 609        | Yes    |
| 16  | 3   | 13 | 591        | Yes    |
| 17  | 3   | 13 | 597        | Yes    |
| 18  | 3   | 14 | 678        | Yes    |
| 19  | 3   | 14 | 757        | Yes    |
| 20  | 3   | 15 | 779        | Yes    |

Table IV is for the range of 0 to 100.

| Pro. | N | S  | Niteration | Result |
|------|---|----|------------|--------|
| 1    | 3 | 2  | 22         | Yes    |
| 2    | 3 | 3  | 52         | Yes    |
| 3    | 3 | 3  | 38         | Yes    |
| 4    | 3 | 4  | 74         | Yes    |
| 5    | 3 | 4  | 72         | Yes    |
| 6    | 3 | 5  | 102        | Yes    |
| 7    | 3 | 6  | 152        | Yes    |
| 8    | 3 | 6  | 153        | Yes    |
| 9    | 3 | 7  | 191        | No     |
| 10   | 3 | 7  | 191        | Yes    |
| 11   | 3 | 9  | 300        | No     |
| 12   | 3 | 9  | 302        | No     |
| 13   | 3 | 11 | 432        | Yes    |
| 14   | 3 | 11 | 434        | Yes    |
| 15   | 3 | 12 | 516        | Yes    |
| 16   | 3 | 13 | 692        | Yes    |
| 17   | 3 | 13 | 655        | Yes    |
| 18   | 3 | 14 | 677        | Yes    |
| 19   | 3 | 14 | 675        | Yes    |
| 20   | 3 | 15 | 770        | Yes    |

With compare of the numerical results generated by Fixed-Point method in the paper [20], one can see that the distributed network could obtain the same computation results for the same example. The comparison of the computation time has no meaning since two methods are run on different computers. Limited by the number of computers in experiment, dimension in examples above is relatively small. However, with hundreds or thousands computers one can image that large dimension problem can be solved easily by this distributed computation network. There are two problems have to be settled for the improvement of this network. One is the solver of linear programming. The advanced self-dual embedding technique presented in [24] has to take the place of CPLEX Concert Technology for solving linear programming. The other one is to upgrade the simple distributed model to the interactive distributed model. With these two problems solved, the performance

## REFERENCES

- [1] J. Nash, "Non-cooperative games," *The Annals of Mathematics*, vol. 54, no. 2, pp. 286–295, 1951.
- [2] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou, "The complexity of computing a nash equilibrium," in *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, 2006, pp. 71–78.
- [3] X. Chen and X. Deng, "Settling the complexity of twoplayer nash equilibrium," in *Proceedings of the 47<sup>th</sup> Annual Symposium on Foundations of Computer Science (FOCS)*, 2006, pp. 261–272.
- [4] C. E. Lemke and J. T. Howson, "Equilibrium points of bimatrix games," *Journal of the Society for Industrial & Applied Mathematics*, vol. 12, no. 2, pp. 413–423, 1964.
- [5] J. Rosenmüller, "On a generalization of the lemke-howson algorithm to noncooperative n-person games," *SIAM Journal on Applied Mathematics*, vol. 21, no. 1, pp. 73–79, 1971.
- [6] R. Wilson, "Computing equilibria of n-person games," *SIAM Journal on Applied Mathematics*, vol. 21, no. 1, pp. 80–87, 1971.
- [7] H. E. Scarf, "The approximation of fixed points of a continuous mapping," *SIAM Journal on Applied Mathematics*, vol. 15, no. 5, pp. 1328–1343, 1967.
- [8] C. B. Garcia, C. E. Lemke, and H. Luethi, "Simplicial approximation of an equilibrium point of noncooperative n-person games," *Mathematical Programming*, vol. 4, pp. 227–260, 1973.
- [9] T. M. Doup and A. J. J. Talman, "A continuous deformation algorithm on the product space of unit simplices," *Mathematics of Operations Research*, vol. 12, no. 3, pp. 485–521, 1987.
- [10] G. Van der Laan, A. J. J. Talman, and L. Van der Heyden, "Simplicial variable dimension algorithms for solving the nonlinear complementarity problem on a product of unit simplices using a general labelling," *Mathematics of Operations Research*, vol. 12, no. 3, pp. 377–397, 1987.
- [11] J. C. Harsanyi, "The tracing procedure: A bayesian approach to defining a solution for n-person noncooperative games," *International Journal of Game Theory*, vol. 4, no. 2, pp. 61–94, 1975.
- [12] P. J. J. Herings and A. Van den Elzen, "Computation of the nash equilibrium selected by the tracing procedure in n-person games," *Games and Economic Behavior*, vol. 38, no. 1, pp. 89–117, 2002.
- [13] P. J. J. Herings and R. J. A. P. Peeters, "A differentiable homotopy to compute nash equilibria of n-person games," *Economic Theory*, vol. 18, no. 1, pp. 159–185, 2001.
- [14] P. J. J. Herings and R. J. A. P. Peeters, "Homotopy methods to compute equilibria in game theory," *Economic Theory*, vol. 42, no. 1, pp. 119–156, 2010.
- [15] R. D. McKelvey and A. McLennan, "Computation of equilibria in finite games," *Handbook of Computational Economics*, vol. 1, pp. 87–142, 1996.
- [16] D. Avis, G. D. Rosenberg, R. Savani, and B. Von Stengel, "Enumeration of nash equilibria for two-player games," *Economic Theory*, vol. 42, no. 1, pp. 9–37, 2010.
- [17] R. S. Datta, "Finding all nash equilibria of a finite game using polynomial algebra," *Economic Theory*, vol. 42, no. 1, pp. 55–96, 2010.
- [18] Y. Yang, Y. Zhang, F. Li, and H. Chen, "Computing all nash equilibria of multiplayer games in electricity markets by solving polynomial equations," *Power Systems, IEEE Transactions on*, vol. 27, no. 1, pp. 81–91, 2012.
- [19] Z. Wu, C. Dang, and C. Zhu, "A mixed 0-1 linear programming approach to the computation of all pure-strategy nash equilibria of a finite n-person game in normal form," *submitted to Optimization*, 2013.
- [20] Z. Wu, C. Dang, and C. Zhu, "Finding the pure-strategy nash equilibrium using a fixed-point algorithm," in *Proceedings of 2013 International Conference on Computer Science, Electronics Technology and Automation*, 2013.

- [21] C. Dang, “An increasing-mapping approach to integer programming based on lexicographic ordering and linear programming,” in *Proceedings of the 9th International Symposium on Operations Research and Its Applications*, 2010, pp. 55–60.
- [22] C. Dang and Y. Ye, “A fixed-point iterative approach to integer programming and distributed computation,” *City University of Hong Kong*, vol. TR-1, 2011.
- [23] G. B. Dantzig, *Linear programming and extensions*. Princeton university press, 1998.
- [24] Y. Ye, *Interior point algorithms: theory and analysis*. John Wiley & Sons, 2011, vol. 44.
- [25] *Message Passing Interface Forum. MPI: A message-passing interface standard, version 2.2.*, <http://www.mpiforum.org/docs/mpi-2.2/mpi22-report.pdf>, 2009.
- [26] Y. Wang and C. Dang, “An evolutionary algorithm for global optimization based on level-set evolution and latin squares,” *Evolutionary Computation, IEEE Transactions on*, vol. 11, no. 5, pp. 579–595, 2007.
- [27] V. Bubelis, “On equilibria in finite games,” *International Journal of Game Theory*, vol. 8, no. 2, pp. 65–79, 1979.

