Computer Engineering and Applications Vol. 4, No. 1, February 2015

An Imperialist Competitive Algorithm for The Winner Determination Problem in Combinatorial Auction

Reza Mostafavi, Seyed Naser Razavi, Mohammad Ali Balafar

Faculty of Electronic and Computer Engineering, University of Tabriz, Tabriz, Iran
rezaa.mostafavi@gmail.com, razavi@iust.ac.ir, balafarila@tabrizu.ac.ir

ABSTRACT

Winner Determination problem (WDP) in combinatorial auction is an NP-complete problem. The NP-complete problems are often solved by using heuristic methods and approximation algorithms. This paper presents an imperialist competitive algorithm (ICA) for solving winner determination problem. Combinatorial auction (CA) is an auction that auctioneer considers many goods for sale and the bidder bids on the bundle of items. In this type of auction, the goal is finding winning bids that maximize the auctioneer's income under the constraint that each item can be allocated to at most one bidder. To demonstrate, the postulated algorithm is applied over various benchmark problems. The ICA offers competitive results and finds good-quality solution in compare to genetic algorithm (GA), Memetic algorithm (MA), Nash equilibrium search approach (NESA) and Tabu search.

Keywords: Winner determination problem, Combinatorial auction, Imperialist competitive algorithm

1. INTRODUCTION

The combinatorial auction is a type of auction that bidders can bid on combinations of items. Auction's winners are given all items that bid. No item is given to those who don't win. Combinatorial auction allows bidders to bid for a bundle of goods (services and resources) and the valuation of the bundles depends on synergies between the individual goods, resources or services [1]. The combinatorial auction is expressed as follows: a set of m indivisible items that are simultaneously auctioned among n bidders [2]. In combinatorial auction, the goal is finding winning bids that maximize the auctioneer's income under the constraint that each item can be allocated to at most one bidder. Determining winner in CA is a complex problem and can be formulated as an optimization problem which is NP-complete [3, 4]. The combinatorial auctions are used in transportation, resource and task allocation in multi agent systems, cloud computing and communication network [2, 5-7].

Winner determination problem in CA is expressed as follows: the WDP consists of a set of m goods $G=\{g_1,g_2,\ldots,g_m\}$ and a set of n bids $B=\{B_1,B_2,\ldots,B_n\}$. Each bid B_j includes a tuple $<M_j,P_j>$ where M_j is a set of items ($M_j \subseteq G$) and P_j is the price of M_j ($P_j \geq 0$, $P_j \in R$). Also consists a matrix $A_{m\times n}$ where $A_{ij}=1$ if the good g_i is offered in M_j by bid B_j and $A_{ij}=0$ if good G_i is not offered in M_j by bid B_j. The auctioneer's objective is to calculate an assignment $X=\{x_1,x_2,\ldots,x_n\}$, $\forall x_j \in \{0,1\}$ to determine winner. When $x_j=1$ bid B_j is accepted (a winning bid) else ($x_j=0$) this is a losing bid.

The WDP can be formulated as an integer linear program:

Maximize $\sum_{j=1}^{n} P_j \cdot x_j$ (1)

Subject to: $\sum_{j=1}^{n} A_{ij} x_j \leq 1 \; \forall i \epsilon \{1,2,\dots,m\}, x_j \epsilon \{0,1\}$ (2)

Function (1) maximizes the auctioneer's income which calculated as the sum of prices of the winning bids. Function (2) is the constraints that mean the item can be allocated to at most one bidder.

2. RELATED WORK

All works done for this problem is classified into three classes [2]:

a) Approximation: the algorithms that find an approximately optimal allocation. The algorithms find winners quickly, but these algorithms don't guarantee optimum solution for all problems. In paper [7], authors present a general technique based on maximal in range mechanisms that converts any α-approximation non-truthful algorithm (α<1) for this problem into $\Omega(\frac{\alpha}{\log n})$ and Ω(α)-approximate truthful mechanisms. When agents have a general multi-parameter function, social welfare in the oracle model is $\Omega(\frac{\sqrt{\log m}}{m})$ [8]. In [9] is given $\Omega(\frac{1}{\sqrt{m}})$ -Approximate truthful mechanism for sub-additive valuation function.

b) Special cases: focus on special cases that can be solved efficiently. In first case, bidder request a bundle of two items. The second case is the linear order case. In this case items are ordered in a linear order and each proper bundle is for an uninterrupted segment of items. Paper [10] reviews greedy mechanisms for truthful combinatorial auction in special cases which agents are interested in sets of size at most s. Also in the paper [11], the authors present a more sophisticated search algorithm in special case.

c) Heuristics: A NP-completes are problems that we cannot write algorithms for them to run in polynomial time and obtain optimal outputs on all input instances. In this category, algorithms try to find optimal (or near optimal) answers in a reasonable time. Casanova [12] is a stochastic local search method proposed in paper [12]. In this paper Casanova is compared with the combinatorial auction structural search (CASS) [3] that is a branch-and-bound algorithm. The results show that Casanova is faster than CASS and also always find a better answer. In paper [13] is proposed a method based on hybrid simulated annealing (SAGII). SAGII includes an embedded branch-and-bound move. The SAGII is compared with the Casanova method and results show SAGII is better than Casanova. Paper [14] proposes four meta-heuristic (stochastic local search, Tabu search, genetic algorithm and memetic algorithm) for solving the winner determination problem. The results in this paper show that memetic algorithm provides competitive result and find a good-quality solution in comparison to other algorithms. But the runtime memetic algorithm is not better than other algorithms. Also, paper [15] proposes a different evolution algorithm and competitive with genetic algorithm and memetic algorithm. On paper [16] is proposed Nash equilibrium search approach (NESA) that is compared with genetic algorithm and results show that NESA is better. Also, the results show that the solution quality is near optimal. The well-known complete algorithms for the WDP are based on the branch-and-bound method. Paper [17] proposes the iterative deepening A*, the Branch-on-items (BoI), the Branch on Bids (BoB) and the combinatorial auction BoB (CABoB) [17].

3. AN IMPERIALIST COMPETITIVE ALGORITHM FOR THE WDP

The imperialist competitive (IC) is a computational algorithm that is utilized to solve optimization problem of different types [18]. The main basis of this algorithm is assimilation, imperialistic competition and revolution.

The ICA algorithm starts by generating a set of candidate random solutions in the search space of the optimization problem. The generated random points are called the initial Countries. Countries in this algorithm are the counterpart of chromosomes in GA and particles in Particle Swarm Optimization (PSO) and it is an array of values of a candidate solution of optimization problems. The cost function of the optimization problem determines the power of each country. Based on their power, some of the best initial countries (the countries with the least cost function value) become Imperialists and start taking control of other countries (called colonies) and form the initial Empires [18].

In this section, an imperialist competitive algorithm is proposed to solve the winner determination problem. The main background of proposed algorithm is presented in the following subsections:

3.1 COUNTRY REPRESENTATION

For simulation of a country, a binary vector (A) having n genome is utilized where n is equal to the number of bids. The value of each component is 0 or 1. Here 1 present accepted of bid and 0 present rejection of bid.

3.2 THE INITIALIZATION OF THE COUNTRY

We use the random key encoding (Rk) [19] for creating a new country. The random key operates as follows: we create n real number between 0 and 1 where n is the number of bids. Then, we select the bid having the maximum order value and add it in the current allocation. Then, we select the bid having the second-highest order value if it does not conflict with bids that are in the allocation. This process repeats until checking all of the bids.

3.3 THE INITIALIZATION OF THE EMPIRES

First, $N_{country}$ initial countries are generated. Then, N_{imp} emperor is selected from the best countries. The other N_{col} countries are assigned to the most similar emperor.

3.4 REVOLUTION: SUDDEN CHANGE IN THE POSITION OF A COUNTRY

In this section, a new empire is made and a colony randomly is selected. Then a genome is chosen and its value is inverted. (i.e. if the genome bit is 1, it is changed to 0). After that, the selected colony is added to the new empire. This is repeated for a specific number of times. Also, the best country is selected for emperor of this empire.

3.5 ASSIMILATION: COLONIES MOVE TOWARDS IMPERIALIST

A new assimilation policy is proposed in the postulated policy. The number of emperor bits replaced with the colony bids and the conflict bids are removed (Figure 1).

This moves colony towards the emperor. Also, a stochastic local search algorithm (Algorithm 1) is used to improve the quality of emperors.

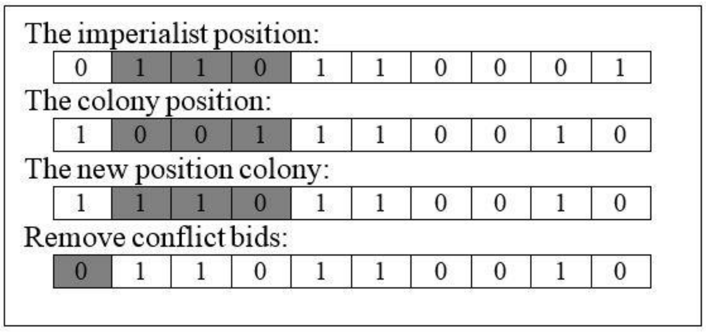

FIGURE 1. Assimilation operation

Algorithm 1. The assimilation method

Require: an allocation empires

```
1.  for i=1 to the size of empires do
2.      For each colony A in emperor(i)
3.          Move colony A toward emperor(i)
4.      End for
5.      Apply SLS on emperor(i)
6.  End for
Return the best allocation found
```

Algorithm 2. The SLS method

Require: a WDP formula, an allocation A, maxiter, wp

Ensure: an improved allocation A

```
1.  For I=1 to maxiter do
2.  R=random number between 0 and 1
3.  If R<wp the
4.      Bid=pick a random bid (*step 1)
5.  Else
6.      Bid=pick a best bid (*step 2)
7.  End if
8.  A=A with picked bid included into it
9.  Remove from A any conflicting bid
10. End for
Return the best allocation found
```

3.6 THE STOCHASTIC LOCAL SEARCH METHOD

ISSN: 2252-4274 (Print)
ISSN: 2252-5459 (Online)

To improve the quality of answers in the empire, we use a stochastic local search method [20]. The SLS method starts with a generated country A, then, it performs a certain number of local steps that consists of selecting a bid to be added in country A and removing all conflicting bids that can be occurred in the current allocation. The added bid is selected according to one of the two following ways:

a) The first way (step1 of Algorithm1) is choosing the bid in a random way with a fixed probability wp>0.
b) The second way (step2) is choosing the best bid.

The process that mentioned above is repeated. The maxiter is defined as a variable that represents the number of iterations and it is fixed empirically.

The SLS algorithm is sketched in Algorithm 2 [20].

3.7 POSITION EXCHANGE BETWEEN A COLONY AND EMPEROR

A colony with a better position than the emperor has the chance to take the control of empire by replacing the existing imperialist.

3.8 ELIIMINATE THE POWERLESS EMPIRES

Gradually, the weak empires lose their power and they will finally be eliminated.

3.9 THE ICA ALGORITHM FOR WDP

The proposed ICA algorithm for the WDP is a simple imperialist competitive algorithm. Countries generated randomly according to the random key encoding. ICA starts with an initial empires (IE) which is randomly selected from countries. Then, it applies the revolution operation to construct a new empire. Also, it applies an assimilation operation to move colonies toward empires. Then it performs exchange operation to exchange position between a colony and an emperor. Finally, it applies elimination operation to eliminate the weak empires. The ICA process is repeated a finite number. This number is obtained by an empirical study. The overall ICA algorithm for the WDP is sketched in Algorithm 3.

Algorithm 3. The imperialist competitive algorithm

```
Require: a WDP formula, maxiter
Ensure: an allocation of bids that maximize the auctioneer's revenue
  1.  Create the conflict graph
  2.  Create an initial empires ica according to RK encoding
  3.  While number of imperialist equal to 1
  4.      Revolution phase
  5.      Assimilation phase (algorithm 2)
  6.      Exchange phase
  7.      Eliminate phase
  8.  End for
  Return the best country solution found
```

4. COMPUTATIONAL RESULTS

This section gives some experiment results. The source code is written in Octave on a computer with core2 Duo 2.5 GHz and 3 Gb of RAM.

We performed several experiments to evaluate the performance of the postulated algorithm on the WDP. The algorithm is compared with some other methods, including Tabu search [14], genetic algorithm (GA) [14], memetic algorithm [21] and equilibrium-based approach [16].

4.1 BENCHMARKS

To measure the performance of algorithms on the WDP problem we use the combinatorial auction test suite (CATS) [22] to generate benchmarks. We generate 50 instances for test. The instances can be divided into 5 different groups where each group contains 10 instances. If m is the number of goods and n is the number of bids, then the details of each group are given as follows:

a) REL-1000-500: 10 instances from 101 to 110: m=500, n=1000.
b) REL-1000-1000: 10 instances from 201 to 210: m=1000, n=1000.
c) REL-500-1000: 10 instances from 301 to 310: m=500, n=1000.
d) REL-1500-1000: 10 instances from 401 to 410: m=1000, n=1500.
e) REL-1500-1500: 10 instances from 501 to 510: m=1500, n=1500.

4.2 PARAMETERS TUNING

We used an experimental study to determine the parameters of the ICA algorithm. The ICA parameters are the country size ($N_{country}$) and the size of the initial emperor (N_{imp}). The SLS search in the improvement phase of ICA, performs a number of iterations at each call to achieve the best result, equal to maxiter. The parameter wp is fixed empirically to 0.2

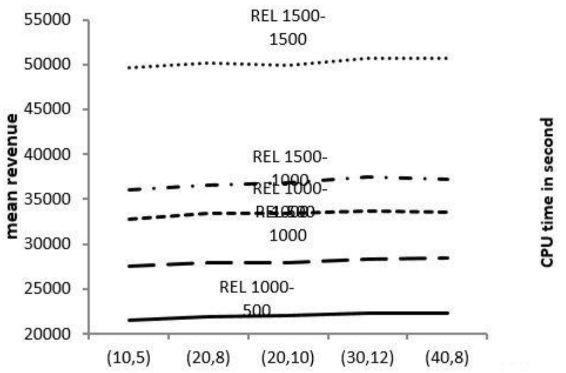

FIGURE 2. The impact of the country and initial emperor parameters on the solution quality of method ICA

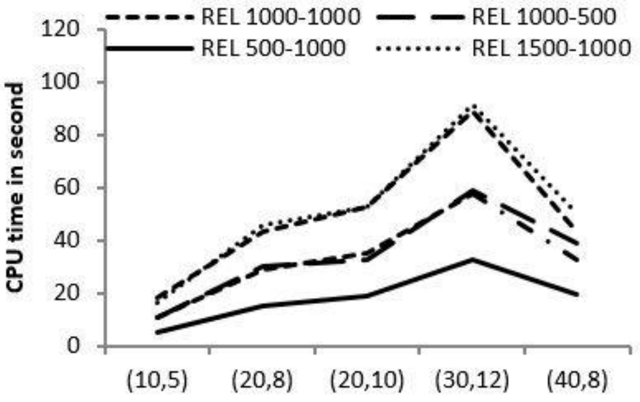

FIGURE 3. The impact of the country and initial emperor parameters on the CPU time of the ICA

4.2.1 THE IMPACT OF THE SIZE OF THE COUNTRY AND INITIAL EMPEROR ON THE ICA PERFORMANCE

We performed several experiments on the instances of the five different groups to show the impact of the size of the country and initial emperor.
Figure 2 shows the impact of the country and initial emperor parameters on the solution quality of ICA. For each group of problems, the quality of the solution is improved when the country and initial emperor parameters increase. Figure 3 shows that the CPU time of ICA becomes larger when the parameters increase.

4.2.2 THE IMPACT OF THE PARAMETERS ON ICA

Table 1 shows the results found for ICA by applying five groups of instances. column sol presents average revenue and column time shows average CPU time of the algorithm in second. This table shows an overview of results with different value of parameters. We can see that by increasing the value of the parameters the solution quality is improved, but the CPU time for the ICA process is increasing.

TABLE 1.
The results of ICA on instances for different parameters.

Test set	ins	$N_{country}$	N_{imp}	Maxiter	Time	sol
REL 500-1000	10	10	5	300	4.1598	172003.10
REL 1000-500	10	10	5	300	8.1593	121198.8
REL 1000-1000	10	10	5	300	9.1342	208077.67
REL 1500-1000	10	10	5	300	14.3393	211518.24
REL 1500-1500	10	10	5	300	14.6269	287171.30
REL 500-1000	10	20	8	60	6.7751	172908.38
REL 1000-500	10	20	8	60	15.1648	122269.12
REL 1000-1000	10	20	8	60	16.5704	208950.86
REL 1500-1000	10	20	8	60	27.2228	213469.48
REL1500-1500	10	20	8	60	27.7536	289634.85
REL 500-1000	10	40	10	300	**21.1246**	**173592.14**
REL 1000-500	10	40	10	300	**39.1037**	**123129.76**
REL 1000-1000	10	40	10	300	**45.7174**	**210858.21**
REL 1500-1000	10	40	10	300	**67.9613**	**215717.91**
REL 1500-1000	10	40	10	300	**64.7756**	**291600.82**
REL 500-1000	10	30	12	500	32.6712	173465.33
REL 1000-500	10	30	12	500	58.9308	134734.65
REL 1000-1000	10	30	12	500	58.4658	211262.15
REL 1500-1000	10	30	12	500	91.5021	233062.31
REL 1500-1000	10	30	12	500	88.5099	317529.71

4.3 EXPERIMENTAL RESULTS

We perform several experiments to evaluate the performance of ICA on the WDP by comparing over utilized state-of-arts methods. The parameters for other methods are:

a) The Tabu search parameters: the maximum number of iterations (maxiter) is 5000, the λ is equal to 40 and the parameter d is set to 10.
b) The genetic algorithm parameters: max generation is set to 150, popsize is equal to 30, crossover rate is set to 0.8 and the mutation rate is set to 0.1.
c) The memetic parameters: the size of collection C is fixed to (8,8), the size of generation is 100, maxiter is equal to 300 and wp is set to 0.3.

4.3.1 A COMPARISON BETWEEN TABU, MEMETIC, GA, NESA AND ICA

We can see that the results of Tabu, memetic, GA, NESA and ICA in tables 2 to 6. In the tables, sol is the solution found by the algorithm and time is CPU time of the algorithm in second.

TABLE 2.
GA, Memetic, Tabu, NESA and ICA on some REL 500-1000 instances

Instances	GA		Memetic		Tabu		NESA		ICA	
	time	sol	time	sol	time	sol	time	sol	time	sol
In301	2.57	5916.18	86.69	6712.09	6.28	6712.09	16.79	5848.77	19.53	6712.09
In302	9.73	930710	78.60	930710	6.08	930710	76.31	178001.17	20.85	930710
In303	66.16	300240.4	107.54	358439.59	7.10	354902.92	498.82	307034.60	25.35	358425.87
In304	29.54	2515.14	92.77	2707.68	7.22	2640.94	264.82	2624.21	24.76	2691.26
In305	34.86	272490.67	92.04	348221.49	7.52	343619.91	245.98	264832.96	20.98	345893.69

In306	13.64	26576.04	83.67	30334.23	8.10	28606.19	109.52	25748.60	19.28	29096.06
In307	12.93	25277.95	83.08	28204.62	8.24	26691.17	110.97	25314.95	17.08	28046.87
In308	26.79	321.15	75.82	321.15	7.08	321.15	34.97	321.15	22.97	321.15
In309	13.51	54.33	83.23	59.06	7.22	56.82	126.37	54.22	25.13	58.29
In310	11.67	29661.06	80.53	35470	7.52	34744.81	133.36	29827.25	15.27	34966.13

TABLE 3.
GA, Memetic, Tabu, NESA and ICA on some REL 1000-500 instances

Instances	**GA**		**Memetic**		**Tabu**		**NESA**		**ICA**	
	time	**sol**	**time**	**sol**	**time**	**sol**	**time**	**sol**	**time**	**sol**
In101	6.67	16892.71	155.96	18908.04	17.27	18908.04	30.03	18314.08	27.43	18832.1
In102	12.68	472331.66	133.97	488882	14.01	488882	24.15	75155.59	39.06	488882
In103	88.65	287894.74	178	359269.84	19.51	337548.52	593.64	288393.03	51.04	352396.75
In104	26.82	2124.49	148.95	2469.83	16.81	2312.3	224.74	2302.20	41.06	2361.53
In105	44.28	224920.30	153.99	302346.96	18.55	290608.52	274.21	228814.53	34.01	298918.2
In106	13.54	18398.49	144.8	22731.15	17.9	20464.63	95.09	18247.12	41.43	21765.04
In107	13.99	18780.93	147.27	22189.85	18.16	20953.8	106.79	18978.79	36.2	22066.02
In108	36.24	451.24	139.17	477.81	17.15	460.57	127.73	451.5	43.69	475.9
In109	19.01	67.50	141.93	72.19	16.02	69.46	134.44	67.31	31.53	71.01
In110	14.26	20007.2	136.07	25865.21	16.83	25183.61	122.86	21583.96	45.54	25530.04

TABLE 4.
GA, Memetic, Tabu, NESA and ICA on some REL 1000-1000 instances

Instances	**GA**		**Memetic**		**Tabu**		**NESA**		**ICA**	
	time	**sol**	**time**	**sol**	**time**	**sol**	**time**	**sol**	**time**	**sol**
In201	6.41	19100.48	154.75	20277.56	17	20277.56	36.8	19120.6	33.44	20277.56
In202	33.37	952030	128.34	965723	12.58	965723	1831	726301.78	39.44	965723
In203	135.6	421996.06	206.47	532577.38	20.36	504628.94	1702	433633.84	63.81	524950
In204	44.8	3225.34	165.09	3753.44	17.66	3694.23	545.04	3593.35	58.1	3747.01
In205	64.35	340127.66	167.54	483069.57	19.33	449419	639.28	358558.65	41.57	475163.04
In206	19.78	33854.09	145.92	38184.07	18.03	37184.47	227.88	36872.99	44.12	38359.94
In207	20.47	28648.47	146.31	34141.25	18.67	31766.1	218.57	29810.36	35.09	33789.44
In208	53.32	586.66	135.22	598.92	18.41	590.84	135.38	586.32	48.16	597.80
In209	23.09	74.41	144.73	85.43	16.27	80.51	248.91	78.44	45.91	83.61
In210	18.73	36044.24	139.52	46944.75	16.83	46309.86	243.07	38142.29	47.49	45990.67

TABLE 5.
GA, Memetic, Tabu, NESA and ICA on some REL 1500-1000 instances

Instances	**GA**		**Memetic**		**Tabu**		**NESA**		**ICA**	
	time	**sol**	**time**	**sol**	**time**	**sol**	**time**	**sol**	**time**	**sol**
in401	6.88	10999.09	222.2	12563.62	23.7	12563.62	36.39	10416.42	48.5	12240.55
in402	18.88	920281.3	190.8	972951	23.4	972951	137.4	177141.8	56.5	972951
in403	208	508356.4	316.3	660188.7	37.8	636119.5	2992	518298.4	101.1	653802.17
in404	56.8	3603.04	245.42	4214.07	29.95	4051.75	884.48	3977.15	74.86	4076.68
in405	102.92	398347.46	258.25	567193.03	34.57	556290.67	1407	418859.6	60.25	559704.69
in406	28.54	37828.12	222.2	44107.94	30.57	41192.46	352.51	37340.61	67.92	43418.38
in407	29.16	31431.18	222.71	37429.59	31.56	35782.32	549.83	31996.92	71.5	36726.97
in408	76.49	1401.04	214.23	1438.39	31.43	1420.09	530.56	1401.1	81.72	1432.32
in409	30.01	100.01	215.95	109.23	27.5	105.55	581.96	104.88	69.22	106.91
in410	30.05	37131.41	210.39	48617.41	28.65	46358.71	826.61	40966.71	68.34	47446.71

TABLE 6.
GA, Memetic, Tabu, NESA and ICA on some REL 1500-1500 instances

Instances	**GA**		**Memetic**		**Tabu**		**NESA**		**ICA**	
	time	**sol**	**time**	**sol**	**time**	**sol**	**time**	**sol**	**time**	**sol**
in501	5.45	6427.45	215.6	8159.5	21.6	8159.5	38.67	6529.83	28.1	8159.5
in502	25.99	1453583	188	1486440	23.3	1486440	784.8	534453.2	53.1	1486440
in503	259.6	607988.2	344.6	798177.1	38.3	788158.3	5538	630143.9	97.8	793414.8
in504	80.1	4793.66	266.1	5529.08	32.2	5422.7	2345	5407.38	80.9	5349.34
in505	117.7	495142.04	274.25	714650.73	35.54	673762.95	2581	526419.81	63.1	685191.77

in506	38.89	51965.42	224.56	60820.64	30.41	9960.65	768.1	51925.16	60.92	60777.02
in507	33.27	43262.29	222.54	50679.66	31.34	49970.8	721.46	44062.03	42.37	50138.41
in508	94.19	1187.32	212.37	1212.3	33	1201.14	468.12	1187.15	80.47	1206.93
in509	31.68	100.4	112.42	216.59	27.56	111.28	698.39	104.29	64.83	109.14
in510	35.31	56161.08	213.94	72484.29	28.58	69315.57	978.41	62072.23	72.37	70375.38

The result of tables 2 to 6 shows that memetic algorithm, Tabu and ICA methods find a good-quality answer for all instances while, the GA and NESA usually fail to find a good-quality answer for all instances. We can see that the Tabu is the fast algorithm. Also, the memetic algorithm usually finds best answer, but the CPU time is high. The ICA algorithm usually finds a best second answer, but the CPU time is better than memetic algorithm. The ICA algorithm has a worse-quality response 0.83% than the memetic algorithm. But the ICA algorithm 71.2% are faster than algorithm algorithm. Also, The ICA algorithm has a better-quality response 1.17 % than tabu algorithm.

5. CONCLUSION

We proposed an imperialist competitive algorithm (ICA) for the winner determination problem in combinatorial auction. We evaluated this method on the different instances of problems, is compared to genetic algorithm (GA), memetic algorithm (MA), Tabu search and Nash equilibrium search (NESA). The results show that the ICA algorithm is competitive algorithm. Though cannot find best answer, but find a good-quality answer in modest time.

ACKNOWLEDGEMENTS

This paper would not have been possible without the support of many people. The author wishes to express her gratitude to her incentives, Leila Habibpour who was abundantly helpful and offered invaluable as sistance, support and guidance. The author would also like to convey thanks to the Ministry and Faculty for providing the financial means and laboratory facilities. The author wishes to express her love and gratitude to her beloved families; for their understanding endless love, through the duration of her studies.

REFERENCES

[1] L. M. Ausubel, P. Cramton, and P. Milgrom, "The clock-proxy auction: A practical combinatorial auction design," 2006.

[2] L. Blumrosen and N. Nisan, "Combinatorial auctions," *Algorithmic Game Theory*, vol. 267, p. 300, 2007.

[3] Y. Fujishima, K. Leyton-Brown, and Y. Shoham, "Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches," in *IJCAI*, 1999, pp. 548-553.

[4] D. Krych, "Calculation and Analysis of Nash Equilibria Af Vickery-based Payment Rules for Combinatorial Exchanges," Harvard University, 2003.

[5] J. Collins, R. Sundareswara, M. Gini, and B. Mobasher, "Bid selection strategies for multi-agent contracting in the presence of scheduling constraints," in *Agent Mediated Electronic Commerce II*, ed: Springer, 2000, pp. 113-130.

[6] N. Nisan, "Bidding and allocation in combinatorial auctions," in *Proceedings of the 2nd ACM conference on Electronic commerce*, 2000, pp. 1-12.

[7] X. Wang, J. Sun, H. Li, C. Wu, and M. Huang, "A reverse auction based allocation mechanism in the cloud computing environment," *Appl. Math,* vol. 7, pp. 75-84, 2013.

[8] R. Holzman, N. Kfir-Dahav, D. Monderer, and M. Tennenholtz, "Bundling equilibrium in combinatorial auctions," *Games and Economic Behavior,* vol. 47, pp. 104-123, 2004.

[9] S. Dobzinski, N. Nisan, and M. Schapira, "Approximation algorithms for combinatorial auctions with complement-free bidders," in *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, 2005, pp. 610-618.

[10] A. Borodin and B. Lucier, "On the limitations of greedy mechanism design for truthful combinatorial auctions," in *Automata, Languages and Programming*, ed: Springer, 2010, pp. 90-101.

[11] T. Sandholm and S. Suri, "BOB: Improved winner determination in combinatorial auctions and generalizations," *Artificial Intelligence,* vol. 145, pp. 33-58, 2003.

[12] H. H. Hoos and C. Boutilier, "Solving combinatorial auctions using stochastic local search," in *AAAI/IAAI*, 2000, pp. 22-29.

[13] Y. Guo, A. Lim, B. Rodrigues, and Y. Zhu, "Heuristics for a bidding problem," *Computers & operations research*, vol. 33, pp. 2179-2188, 2006.

[14] D. Boughaci, "Metaheuristic approaches for the winner determination problem in combinatorial auction," in *Artificial Intelligence, Evolutionary Computing and Metaheuristics,* ed: Springer, 2013, pp. 775-791.

[15] D. Boughaci, "A Differential Evolution Algorithm for the Winner Determination Problem in Combinatorial Auctions," *Electronic Notes in Discrete Mathematics,* vol. 36, pp. 535-542, 2010.

[16] C. Tsung, H. Ho, and S. L. Lee, "An Equilibrium-Based Approach for Determining Winners in Combinatorial Auctions," in *Parallel and Distributed Processing with Applications (ISPA), 2011 IEEE 9th International Symposium on,* 2011, pp. 47-51.

[17] T. Sandholm, "Optimal winner determination algorithms," *Combinatorial auctions*, pp. 337-368, 2006.

[18] E. Atashpaz-Gargari and C. Lucas, "Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition," in *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, 2007, pp. 4661-4667.

[19] J. C. Bean, "Genetic algorithms and random keys for sequencing and optimization," *ORSA journal on computing*, vol. 6, pp. 154-160, 1994.

[20] D. Boughaci, B. Benhamou, and H. Drias, "Stochastic local search for the optimal winner determination problem in combinatorial auctions," in *Principles and Practice of Constraint Programming*, 2008, pp. 593-597.

[21] D. Boughaci, B. Benhamou, and H. Drias, "A memetic algorithm for the optimal winner determination problem," *Soft Computing*, vol. 13, pp. 905-917, 2009.

[22] K. Leyton-Brown, M. Pearson, and Y. Shoham, "Towards a universal test suite for combinatorial auction algorithms," in *Proceedings of the 2nd ACM conference on Electronic commerce*, 2000, pp. 66-76.