

Differential Evolution for Discrete Optimization: An Experimental Study on Combinatorial Auction Problems

Jingqiao Zhang, Viswanath Avasarala, Arthur C. Sanderson, and Tracy Mullen

Abstract: Differential evolution (DE) mutates solution vectors by the weighted difference of other vectors using arithmetic operations. As these operations cannot be directly extended to discrete combinatorial space, DE algorithms have been traditionally applied to optimization problems where the search space is continuous. In this paper, we use JADE, a self-adaptive DE algorithm, for winner determination in Combinatorial Auctions (CAs) where users place bids on combinations of items. To adapt JADE to discrete optimization, we use a rank-based representation schema that produces only feasible solutions and a regeneration operation that constricts the problem search space. It is shown that JADE compares favorably to a local stochastic search algorithm, Casanova, and a genetic algorithm based approach, SGA.

I. INTRODUCTION

EVOOLUTIONARY algorithms (EAs) are stochastic, population-based search methods that mimic the metaphor of natural biological evolution, such as mutation, recombination, selection, locality and neighborhood. They generally operate on a population of potential solutions applying the principle of survival of the fittest to produce better and better approximations to a solution. As shown by successes in various fields such as engineering, finance, biology [1, 2], evolutionary algorithms consistently perform well in searching optimal solutions to various types of problems. Differential evolution is a latest branch of EAs proposed by Storn and Price in 1995 [3, 4]. The crucial idea behind DE is a scheme of generating mutant vectors using the weighted difference of other vectors randomly chosen from the population. This approach requires no separate probability distribution to generate mutant vectors and it makes the scheme self-organizing.

Similar to other EAs, differential evolution's performance is usually sensitive to its problem-dependent control parameters. In general, there is no single parameter setting that is suitable

for various problems or even at different evolution stages of a single problem. To address this issue, various parameter adaptation [5, 6] or self-adaptation [7, 8] strategies have been introduced to improve the performance of differential evolution. The resultant DE algorithms have been found to perform better on a wide variety of test problems than traditional evolutionary algorithms.

The mutation operation of DE, like other numerical optimizers, relies on arithmetic operations (add, subtract, multiply, divide, etc.) rather than general data manipulation operations (sort, swap, permute, etc.). It cannot be directly extended to discrete combinatorial space. Therefore, DE algorithms have been traditionally developed for optimization problems where the search space is continuous, while the extensions to discrete combinatorial optimization problems are relatively small (see [4, Chapter 4.4] for related references) and are found to have some drawbacks like failing to recognize identical/equivalent solutions or generating a high proportion of infeasible solutions that need to be repaired.

A seminal combinatorial optimization problem is winner determination in combinatorial auctions. Combinatorial auctions allow agents to bid on a combination of items, and express any complementarities directly and are used in such environments with strong synergistic relationships between goods. For example, combinatorial auctions have been used for resource allocation in sensor management (SM) [9, 10], supply chain management [11] and computer grids [12]. However, winner determination in combinatorial auctions, i.e., finding out the optimal categorization of bids as either "winning" or "losing" is an NP-hard problem [13]. Various approaches have been suggested in the literature to solve this problem, including exact optimization algorithms like CPLEX [14, 15] and CABOB [16] and approximate optimization techniques Casanova [17] and SGA [18]. For many realistic distributions, CPLEX and CABOB perform extremely well with average running time being less than a second for problem sizes with thousands of bids. However, for certain complex bid distributions, neither CPLEX nor CABOB's real-time performance is adequate in the environments with strict real-time constraints. For example, for certain bid distributions, the complete algorithms took extremely long completion time even problems of for medium size (see [18] for details). Researchers have used heuristic approaches for solving the "hard" bid distributions. Two of the most recent approaches that have been shown to outperform others are Casanova and SGA. Casanova, a local stochastic search procedure, is an approximate winner determination algorithm, based on Novelty+. SGA is a standard genetic

Jingqiao Zhang and Arthur C. Sanderson are with the Center for Automation Technologies and Systems, Rensselaer Polytechnic Institute, Troy, NY 12180, USA. (Emails: zhangj14@rpi.edu; sandea@rpi.edu)

Viswanath Avasarala is with the Computing and Decision Sciences, GE Global Research Center, Niskayuna, NY 12309, USA. (Email: avasaral@ge.com).

Tracy Mullen is with the School of Information Science and Technology, Pennsylvania State University, University Park, PA 16801, USA. (Email: tmullen@ist.psu.edu).

Funding for this work was provided in part by grant number IIS-0329837 from the National Science Foundation. This work is also supported in part by the Center for Automation Technologies and Systems (CATS) under a block grant from the New York State Office of Science, Technology, and Academic Research (NYSTAR).

algorithm that uses a modified representation schema to enhance its performance.

In this paper, we apply JADE, a self-adaptive differential evolution algorithm recently proposed by the authors for continuous optimization [19], to the approximate winner determination in combinatorial auctions. As its control parameters are adjusted in a self-adaptive manner, JADE avoids users' prior knowledge about parameter settings and achieves consistently good performance for optimization problems of different characteristics. However, the algorithm has only been tested previously on continuous search spaces and its applicability to combinatorial problems has not been studied.

To apply JADE to the combinatorial auction problems, we consider a rank-based representation schema that transforms the discrete combinatorial problem into continuous domain and only produces feasible solutions to the problem. Also, we propose a regeneration operator to represent equivalent solutions into a unique form so as to constrict the search space that is previously enlarged due to the continuous representation of the discrete search space. Furthermore, to improve the convergence performance, we initialize the algorithm with a set of solutions that are randomly generated in the neighborhood of a seeded high-quality solution. The resultant methodology, the seeded JADE with regeneration operation, is shown to consistently achieve better solutions than Casanova and SGA for two combinatorial auction problems that are toughest for exact winner determination algorithms [14].

This paper is organized as follows. Section II introduces the winner determination problem and the schema for representing this combinatorial problem into continuous domain. Section III describes the procedure of JADE and proposes regeneration and seeding operations to improve the performance of JADE in optimizing the discrete combinatorial problem. In Section IV, we compare the performance of JADE with other approximate winner determination algorithms. The final section presents our conclusions and directions for future work.

II. PROBLEM DESCRIPTION AND CURRENT APPROACHES

This section provides an explanation of the winner determination problem and briefly describes the best solution approaches currently available.

A. Winner Determination for Resource Allocation Using CAs

Assume that the auctioneer has a set of m goods/items $G = \{g_1, g_2, \dots, g_m\}$ to sell, and a set of n bids $B = \{b_1, b_2, \dots, b_n\}$. Bidders offer bids of the form $B_i = \langle b_i, p_i \rangle$, where b_i is the bundle of goods and p_i is the price the bidder is willing to pay for. The winner determination problem is to find an allocation of goods that maximizes the overall utility, given the constraint that each good can be sold to no more than one bid. Formally, this problem can be formulated as [13]:

$$\max \sum_{j=1}^n p_j x_j \quad \text{s.t.} \quad \sum_{j \in S(i)} x_j \leq 1, \forall i \in \{1, 2, \dots, m\} \quad (1)$$

where $S(i)$ is the set of bids that contain good i , and x_j is a binary variable that equals 1 if bid j is accepted to the final allocation and 0 otherwise. A solution is feasible if and only if each good appears in at most one accepted bid.

B. Current Approaches

The various techniques that have been proposed for winner determination problem can be classified into two categories:

1. Exact winner determination: the algorithm guarantees the convergence to global optimum. Examples are CPLEX and CABOB.
2. Approximate winner determination: the algorithm does not provide optimality guarantees. Examples are Casanova and SGA.

The most successful approach for exact winner determination is solving it as an integer-programming problem using standard integer programming (IP) software like CPLEX [14]. CPLEX has been shown to perform exceedingly well on a wide-range of problem distributions (see [15] for details).

Another example of exact winner determination procedure is CABOB, which uses a specialized depth first search that branches on bids. CABOB constructs a *bidgraph* with each bid in the set of bids B as the vertex of the graph and edges between vertices only when they have items in common. CABOB then uses a separate depth-first search (DFS) to find the optimal solution. By selecting an initial bid as being in a possible allocation, then remaining bids with items that do not overlap this bid can also be in the allocation and are considered for allocation in a particular search path of the DFS. CABOB bounds the DFS by pruning paths that have already been dominated.

CABOB's performance has been found to be marginally better than CPLEX for certain hard to solve problems [16]. However, when the problem sizes are large and when the bid-distributions are not simple, both CABOB and CPLEX fail to scale up adequately [18].

For these scenarios, approximate winner determination procedures are useful. The most prominent approach in this genre has been Casanova, a local stochastic search procedure [17]. Casanova starts with an empty allocation and bids are chosen from the unsatisfied bids randomly and added to the allocation vector. The probability that a bid is chosen is determined using its normalized score, calculated as the bid price divided by the product of the number of goods the bid consumes, the bid's age, and the number of steps since it has last been chosen, to be added to the allocation vector. More recently, the authors have proposed SGA [18], a standard genetic algorithm based on a novel representation method that restricts evolutionary search only to feasible solutions. SGA performed better than Casanova for large problems.

C. Representation Schema for Discrete Optimization

Techniques such as differential evolution and particle swarm optimization [20], which function by manipulating the coordinates of solution vectors by arithmetic operations (add, subtract, multiply, divide, etc.), are not directly applicable to discrete optimization problems. Recent developments to address this drawback involve converting the search space to a continuous one based on some transformations [21, 22, 23]. A popular approach among them is proposed by Kennedy and Eberhart [21] which, instead of searching over the binary space, operates on the probability (in a continuous space) that a binary variable takes value 0 or 1. This approach is applicable to the

combinatorial auction problems as the decision variables x_j in (1) takes binary values. However, an important drawback is that it causes a high proportion of infeasible solutions where multiple bids containing a common item are marked as winning and therefore leads to poor convergence performance [18].

To avoid the problem of infeasible solutions, we derive from the idea in [18] which produces only feasible solution in discrete space using a rank-based representation schema. In view of the high efficiency of differential evolution in continuous optimization, we extend this representation schema so that the problem can be transformed into a continuous domain. Also, we propose a regeneration operation that constricts the complexity of the problem by ensuring that all equivalent solutions in the continuous space can be represented in a unique form.

In the rank-based representation schema, each solution is represented by a vector \mathbf{x} of length n whose elements take values between 0 and 1; i.e., $\mathbf{x} = (x_1, x_2, \dots, x_n)$, with $x_i \in [0, 1]$. The vector \mathbf{x} can be decoded into an ordered *accept list* and an ordered *reject list*, based on which the solution quality can be evaluated. The decoding process is given as follows:

1. Generate a rank vector $\mathbf{z} = (z_1, z_2, \dots, z_n)$ with $z_i = k$ if x_i is the k -th smallest element in \mathbf{x} ; i.e., the i -th bid is considered to have a rank k .
2. Initialize the accept list with the lowest ranking bid, and set the reject list to be empty.
3. Consider the bid with the next lowest rank. Add it to the end of the accept list if it does not create an infeasible solution; otherwise add it to the end of reject list. Proceed to the bid with the next lowest rank.
4. Continue till the bid with the highest rank is analyzed.

A solution vector (0.91, 0.90, 0.13, 0.81) The corresponding rank vector: (4, 3, 1, 2)					
Rank	bid #	Items in the bid	State	Accept list	Reject list
1	3	a, b, c	Win	{3}	\emptyset
2	4	a, d, f	Lose	{3}	{4}
3	2	d, e	Win	{3, 2}	{4}
4	1	b, d, f	Lose	{3, 2}	{4, 1}

Table 1. A problem instance. The accept list and reject list are updated according the acceptance or rejection of bids.

For example, consider a scenario with 4 bids and 6 items (a, b, ..., f) as shown in Table 1. A solution vector (0.91, 0.90, 0.13, 0.81) is decoded in the following manner. According to the sorting result, the four elements of the vector are assigned ranks 4, 3, 1, and 2, respectively. The initial accept list is set to be {3}, as bid 3 has the lowest rank (rank 1). The next lowest ranking bid is bid 4 (rank 2). However, bid 4 has item a, which is also present in bid 3. Since bid 3 has already been labeled as winning, bid 4 is labeled as losing and added to the reject list. The procedure is repeated until the status of the highest-ranking bid (bid 1 in this case) is determined. At the end, the vector (0.91, 0.90, 0.13, 0.81) corresponds to a solution where bids 3 and 2 are marked as winning (i.e., accepted to the final allocation) and bids 4 and 1 marked as “losing”.

III. SEEDED JADE WITH REGENERATION OPERATION

In this section, we describe the procedure of JADE and propose two new operations, regeneration and seeding, to

improve the performance of JADE in optimizing the combinatorial auction problems.

A. The basic procedure of JADE

JADE follows the basic procedure of an evolutionary algorithm. The initial population of real-valued vectors is randomly generated according to a uniform distribution between the lower and upper bounds defined for each component of the vector. After the initialization, JADE enters a loop of evolutionary operations (mutation, crossover and selection) and self-adaptation operation.

Mutation: At each generation g , a set of mutation vectors $\mathbf{v}_{i,g}$ are created based on the current parent population $\{\mathbf{x}_{i,g} | i = 1, 2, \dots, NP\}$, where NP is the population size. JADE utilizes a ‘DE/current-to-p-best/1’ mutation strategy to generate mutant vectors

$$\mathbf{v}_{i,g} = \mathbf{x}_{i,g} + F_i(\mathbf{x}_{\text{best},g}^p - \mathbf{x}_{i,g}) + F_i \cdot (\mathbf{x}_{r_1,g} - \mathbf{x}_{r_2,g}), \quad (2)$$

where the indices r_1 and r_2 are distinct integers uniformly chosen from the set $\{1, 2, \dots, NP\} \setminus \{i\}$, $\mathbf{x}_{r_1,g} - \mathbf{x}_{r_2,g}$ is a difference vector to mutate the parent, $\mathbf{x}_{\text{best},g}^p$ is randomly chosen as one of the top $100p\%$, $p \in (0, 1]$, individuals in the current population, and $F_i \in (0, 1]$ is the mutation factor associated with the i -th individual $\mathbf{x}_{i,g}$ and is randomly generated by the self-adaptation operation described later.

Crossover: After mutation, a ‘binary’ crossover operation forms the final trial vector $\mathbf{u}_{i,g} = (u_{1,i,g}, u_{2,i,g}, \dots, u_{D,i,g})$:

$$u_{j,i,g} = \begin{cases} v_{j,i,g} & \text{if } \text{rand}_j(0,1) \leq CR_i \text{ or } j = j_{\text{rand}} \\ x_{j,i,g} & \text{otherwise,} \end{cases} \quad (3)$$

where $\text{rand}_j(a,b)$ is a uniform random number on the interval $(a, b]$ and newly generated for each j , $j_{\text{rand}} = \text{randint}(1, D)$ is an integer randomly chosen from 1 to D and newly generated for each i , and the crossover probability, $CR_i \in (0, 1]$, roughly corresponds to the average fraction of vector components that are inherited from the mutant vector.

Selection: In the selection operation, we choose the better one from the parent vector $\mathbf{x}_{i,g}$ and the trial vector $\mathbf{u}_{i,g}$ according to their fitness values $f(\cdot)$. For example, if we have a minimization problem, the selected vector is given by

$$\mathbf{x}_{i,g+1} = \begin{cases} \mathbf{u}_{i,g} & \text{if } f(\mathbf{u}_{i,g}) < f(\mathbf{x}_{i,g}) \\ \mathbf{x}_{i,g} & \text{otherwise,} \end{cases} \quad (4)$$

and is used as a parent vector in the next generation. If the trial vector $\mathbf{u}_{i,g}$ succeeds, the selection is considered as a *successful update* and the corresponding control parameters F_i and CR_i are called a *successful mutation factor* and *successful crossover probability*, respectively.

Self-Adaptation: The self-adaptation of the parameters F_i and CR_i is based on the principle that better values of control parameters tend to generate individuals that are more likely to survive and thus these values should be propagated. To be specific, F_i and CR_i are generated by two random processes:

$$CR_i = \text{randn}_i(\mu_{CR}, 0.1) \quad (5)$$

$$F_i = \text{randc}_i(\mu_F, 0.1) \quad (6)$$

where $\text{randn}(\mu, \sigma^2)$ denotes a random value from a normal distribution of mean μ and variance σ^2 , $\text{randc}(\mu, \delta)$ denotes a random value from a Cauchy distribution with location and

scale parameters μ and δ , respectively. The mean μ_{CR} and the location parameter μ_F are updated in a self-adaptive manner:

$$\mu_{CR} = (1-c)\mu_{CR} + c \cdot \text{mean}_A(S_{CR}), \quad (7)$$

$$\mu_F = (1-c)\mu_F + c \cdot \text{mean}_L(S_F), \quad (8)$$

where S_{CR} and S_F are the respective sets of all successful crossover probabilities and successful mutation factors obtained in the selection (4) at generation g , c is a positive constant between 0 and 1 and $\text{mean}_A(\cdot)$ is the usual arithmetic mean operation and $\text{mean}_L(\cdot)$ is the Lehmer mean

$$L(S_F) = \frac{\sum_{F \in S_F} F^2}{\sum_{F \in S_F} F}, \quad (9)$$

which plays more weight on larger mutation factor F to improve evolutionary progress.

Interested readers are referred to [19] for detailed explanations on the above self-adaptation operations.

B. Regeneration Operation

JADE, similar to other differential evolution algorithms, has been originally designed for continuous optimization problems. Though the optimization problem has been transformed into the continuous domain by the rank-based representation schema, the rank-based schema still presents a problem to JADE for the following reason. The rank-based schema is unable to detect identical or equivalent solutions, because what matters in the schema is the rank (instead of the value) of the element in a solution vector. For example, two vectors (0.3, 0.5, 0.6, 0.7) and (0.1, 0.2, 0.7, 0.9) lead to the same rank vector (1, 2, 3, 4) and thus represent an identical solution. However, their difference, i.e., (0.2, 0.3, -0.1, -0.2), is not close to zero. As a result, JADE (especially its mutation operator) may not function efficiently because the difference between two solutions in the decision space (e.g., the Euclidean distance) has a weak relation to the difference in the objective space (i.e., the difference of the fitness values).

To avoid the above problem, we propose a regeneration operation to represent all *equivalent* solutions that lead to the same accept list and reject list into a unique form. Taking a solution vector $\mathbf{u} = (0.91, 0.90, 0.13, 0.81)$ in Table 1 as example, we can describe the regeneration operation as follows:

1. Concatenate the accept list and the reject list into a vector. This vector shows the sequence of bids getting accepted or rejected. Denote this *sequence* vector as $\mathbf{s} = (s_1, s_2, \dots, s_n)$. For the example in Table 1, we have $\mathbf{s} = (3, 2, 4, 1)$.
2. Generate a new vector

$$\mathbf{u}' = (u'_1, u'_2, \dots, u'_n), \text{ with } u'_i = j/n, \quad (10)$$

if bid j is the i -th bid in the sequence vector (i.e., $s_j = i$). We have $\mathbf{u}' = (0.75, 0.50, 0.25, 1.00)$.

3. Add a small random disturbance to \mathbf{u}' , and we obtain
- $$\mathbf{u}'' = \mathbf{u}' - \text{rand}(0, 1/n), \quad (11)$$
- where $\text{rand}(0, 1/n)$ is a uniform random number on the interval $(0, 1/n]$. We have $\mathbf{u}'' = (0.67, 0.42, 0.17, 0.92)$ if, for example, $\text{rand}(0, 1/n) = 0.07$.
4. Replace \mathbf{u} with \mathbf{u}'' .

It is easy to show that both \mathbf{u}' and \mathbf{u}'' are equivalent to \mathbf{u} . Indeed, we can obtain a unique \mathbf{u}' for all equivalent solution vectors that are different in floating-point values but lead to the same accept and reject lists (and therefore the same sequence

vector \mathbf{s}). It is worth noting that two vectors are considered as equivalent if and only if their respective accept/reject lists contain the same elements and the elements are listed in the same order. Take the problem in Table 1 as example. The vector (0.91, 0.90, 0.13, 0.81) leads to an accept list {3, 2} and a reject list {4, 1}. Now, consider another vector (0.91, 0.13, 0.90, 0.81) which leads to an accept list {2, 3} and a reject list {4, 1}. In both cases, bids 2 and 3 win, and bids 1 and 4 lose. However, the two vectors are not equivalent in the sense that the former (the latter) indicates a higher priority of bid 2 (bid 3) and thus more likely produces offspring that again include bid 2 (bid 3) in the accept list.

By representing all equivalent solution in a unique form \mathbf{u}' , the problem complexity, which is inevitably enlarged due to the continuous representation of the discrete search space, can be greatly constricted. In (11), we add a small random disturbance to \mathbf{u}' , because otherwise the difference of individual vectors only take values that are a multiple of $1/n$, causing degraded performance in the mutation operation. The random disturbance is small enough not to affect the rank of the elements (and thus \mathbf{u}' and \mathbf{u}'' are equivalent), so the problem's complexity is not changed due to (11).

C. Seeding JADE

If a high quality solution is available before the optimization, we are able to boost the performance of an algorithm by initializing solutions in the neighborhood of the high quality solution. The following solutions can be considered as the candidates of high quality solutions:

1. The solution obtained by a less-efficient yet fast approximate winner determination algorithm.
2. The solution obtained by calculating a quick lower bound for the winner determination problem using linear programming. For example, for CAT distributions and weighted random distribution [16], a linear programming solution is usually close in value to the optimum.
3. The optimal or high quality solution previously obtained in an incremental winner determination process. Assume that the auctioneer has calculated an optimal or high quality solution for the bids received. If certain bids are retracted or new bids are added (e.g., in the sensor management scenario [9]), the auctioneer can use the previous solution as a high quality solution to the new problem.

In this paper, we consider a simple heuristic method that finds a high quality solution \mathbf{x}^{hq} based on the unit price of a bid (i.e., the price of a bid divided by the number of items in it). The basic reasoning is that the higher the unit price of a bid, the higher the priority of this bid being considered in the auction. To be specific, the i -th element x_i^{hq} of \mathbf{x}^{hq} is assigned a value j/n , if bid i has the j -th highest unit price among all the n bids.

We next randomly generate a set of seed solutions in the neighborhood of the high-quality solution obtained above. The k -th seed solution $\mathbf{x}^k = (x_1^k, x_2^k, \dots, x_n^k)$ is obtained in the following manner:

$$x_i^k = x_i^{\text{hq}} + \text{rand}_i(0, k/n), \text{ for } i = 1, 2, \dots, n. \quad (12)$$

It is clear that the larger the k , the more disturbance is added to the elements of \mathbf{x}^{hq} . In the case of $k = 1$, \mathbf{x}^1 is equivalent to \mathbf{x}^{hq}

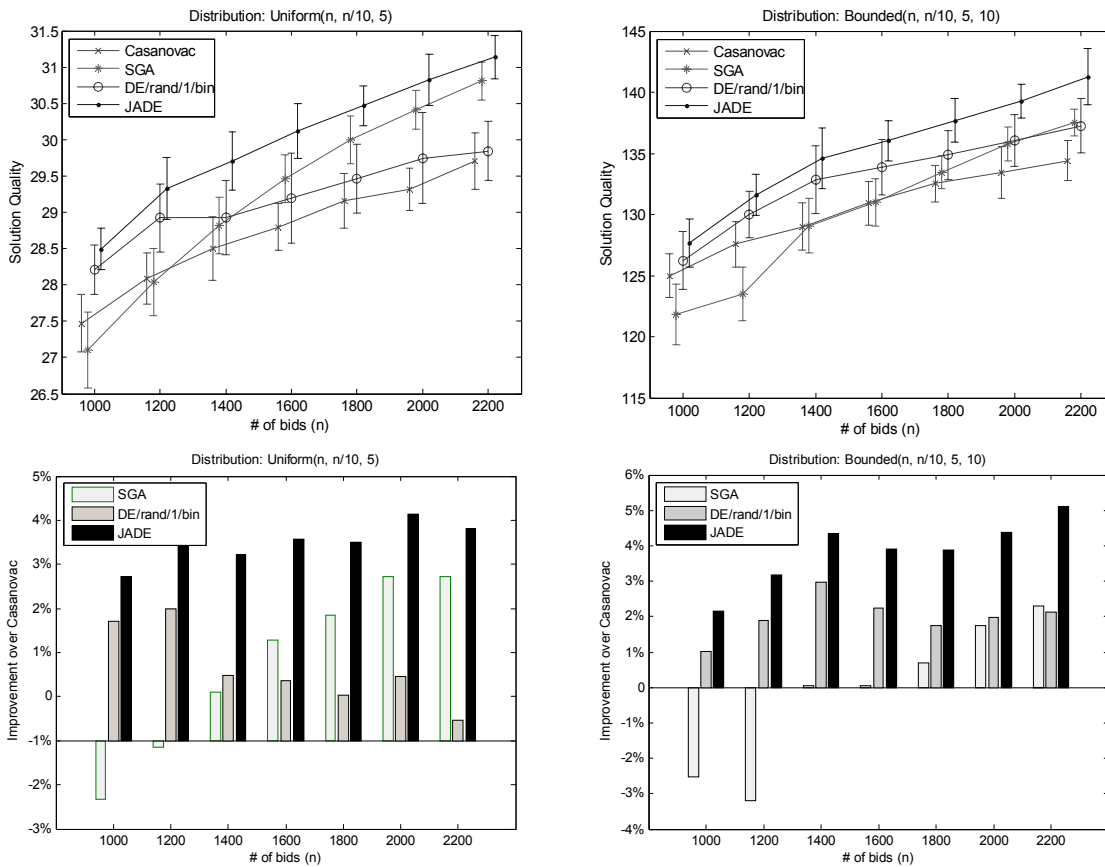


Fig. 1: A performance comparison among JADE, Casanova, SGA, and DE/rand/1/bin. The top two subplots show the mean and standard deviation of the best solutions obtained by each algorithm. For clarity, the error bars are slightly shifted for each algorithm in the subplots. The bottom two subplots show the improvement of JADE, SGA and DE/rand/1/bin over Casanova.

because the random disturbance is always less than $1/n$, the minimum difference between the elements of \mathbf{x}^{hq} , and thus does not affect the rank of the elements.

IV. PERFORMANCE COMPARISON

In this section, we compare JADE with other prominent approximate winner determination algorithms, Casanova and SGA. Also, we include a classic differential evolution algorithm DE/rand/1/bin [3, 4] in the comparison so as to identify the benefit of the features introduced in JADE.

A. Experimental Setting

We perform two sets of experiments on the uniform and bounded bid distributions, as they are shown to be the toughest for the exact winner determination algorithms [14].

1. Uniform (n, m, λ) distribution which consists of n bids, each with λ items chosen without replacement from the m items. Price is chosen randomly from a uniform distribution on $[0, 1]$. In our experiments, we use $m = n/10$ and $\lambda = 5$, as in [16].
2. Bounded $(n, m, \lambda^1, \lambda^2)$ distribution where the problem consists of n bids. The number of items λ is randomly

chosen between a lower bound λ^1 and an upper bound λ^2 . The price is chosen from a uniform distribution on $[0, \lambda]$. In our experiments, we use $m = n/10$ and $\lambda^1 = 5$ and $\lambda^2 = 10$, as in [16].

The parameter values of JADE are selected as $c = 1/10$, and $p = 5\%$, as suggested in [19]. In DE/rand/1/bin, we set $F = 0.5$ and $CR = 0.9$, as recommended in [3, 24, 25]. For fair comparison, the regeneration and seeding operations proposed in JADE is also applied to DE/rand/1/bin. In both algorithms, the number of generations is set to be 500, and the population size is $NP = n/2$, which is much smaller than the value $3n \sim 10n$ as usually suggested in the literature of DE [3]. The small population size speeds up the convergence rate at the expense of the exploration capability of the algorithm. However, as shown below, the final performance JADE is still significantly better than other algorithms.

For Casanova, we follow the parameters suggested in the original paper [17]. However, we find that changing the walk probability to 0.4 gives better results and hence $w_p = 0.4$ is used. The SGA parameters are chosen from [18]: the population size is set to 6000 if $n \leq 1400$ and 6500 otherwise; the tour size is

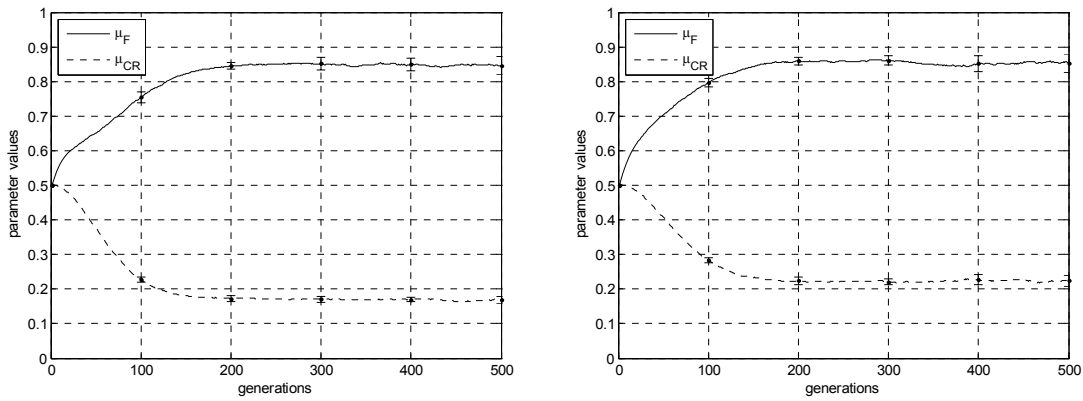


Fig. 2: The evolutions of μ_F and μ_{CR} in the optimization of problems with uniform (left plot) or bounded (right plot) bid distributions. The number of bid is 2200.

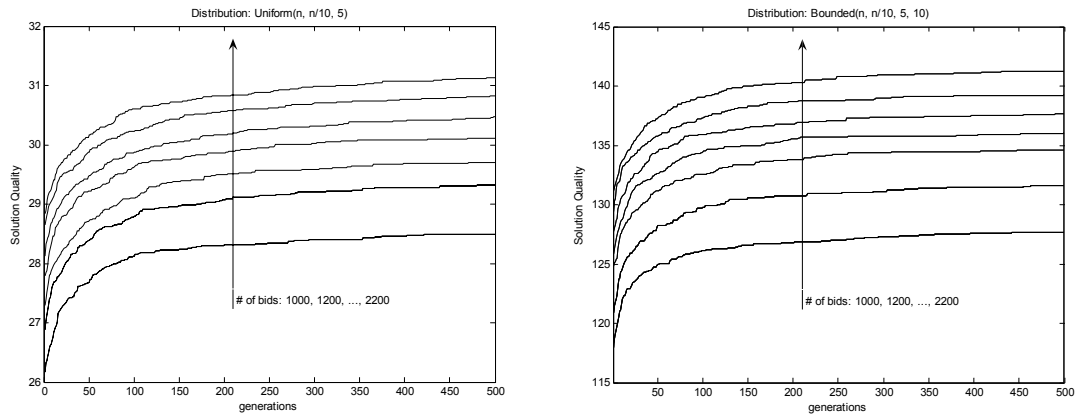


Fig. 3: The convergence graph of JADE (the mean of the results in 30 independent runs).

equal to 2; and the crossover and mutation probabilities equal 0.995 and 0.02, respectively. Note that the population size of SGA is much greater than that of JADE.

We compare the four algorithms on large problems, varying the number of bids from 1000 to 2200 bids. In each case, the results reported below are calculated based on the evaluation of each algorithm on 30 problem instances randomly generated according to the uniform or bounded bid distribution.

B. Comparison Result

Fig. 1 illustrates the performance comparison between JADE and other algorithms. JADE shows consistently better performance than Casanova for the problems with both uniform and bounded bid distribution. The benefit is slightly increasing (from 2~3% to 4~5%) as the problem size goes up from 1000 to 2200. In addition, there is no increasing trend in the variation of results, indicating similar robustness of JADE and Casanova for problems of different sizes.

JADE performs better than SGA as well. As shown in Fig. 1, SGA is worse than Casanova in the case of small problems but is better than Casanova when the problem size becomes greater than 1200 or 1400 in the case of uniform or bounded bid distributions, respectively. JADE outperforms Casanova in all

the conducted experiments.² The performance of SGA gets closer to that of JADE as the problem size increases: while it trails JADE by a large margin (about 5%) for small problems, SGA becomes 1% or 3% worse than JADE for large problem in the case of the uniform or bounded bid distributions, respectively. However, there is no trend that SGA could beat JADE for even larger problems. It is worth noting that these comparison results are obtained with JADE using a much smaller population than SGA. In addition, considerable extra experiments have been conducted to select the optimal set of parameters used in SGA for these combinatorial auction problems. As a comparison, parameter selection for JADE is very simple owing to the ability of parameter self-adaptation.

As JADE gets better performance than SGA, it is interesting to investigate if its benefit comes solely from the underlying differential evolution strategy or stems from its other unique features as well. The comparison of JADE and SGA with the classic differential evolution algorithm DE/rand/1/bin provides some information in this aspect. As shown in Fig. 1, DE/rand/1/bin achieves much better results than SGA for small

² This is the case even when the problem size is as low as 200. For even smaller problems, both JADE and Casanova easily find solutions competitive to the results of exact winner determination algorithms.

problems but does not continue performing efficiently for large problems. In all cases, however, DE/rand/1/bin is clearly worse than JADE, indicating the effectiveness of the new features (parameter self-adaptation and DE/current-to- p -best mutation strategy) introduced in JADE.

The effect of parameter self-adaptation in JADE is illustrated in Fig. 2. We show the evolution process of μ_F and μ_{CR} with mean curves and error bars. The error bars show a small variance of μ_F and μ_{CR} at different stages of the optimization, indicating a clear evolution trend of μ_F and μ_{CR} over 30 independent experiments for both bid distributions. As the evolution process is automatic, JADE is able to find the optimal or near optimal control parameter values without users' prior knowledge about parameter settings or interaction during the evolution search.

It is also interesting to investigate the scalability of JADE for combinatorial auction problems. Illustrated in Fig. 3 is the convergence graph of JADE where the mean of the 30 random problems are plotted at each generation. It shows that JADE usually reaches a performance plateau after 100 to 200 generations, regardless of the problem size. In addition, we observe that JADE achieves better results than Casanova after only 20 ~ 50 generations by comparing the convergence graphs to the results of Casanova in Fig. 1.

V. CONCLUSIONS

Differential evolution has been successfully applied to a wide range of continuous optimization problems. In this paper, we apply a self-adaptive differential evolution algorithm JADE to a discrete optimization problem, the winner determination in combinatorial auctions. As the combinatorial auction is a discrete optimization problem, we transform the problem into continuous domain by a rank-based representation method and solve it using JADE. As the transformation into continuous domain inevitably enlarges the search space, we further propose a regeneration operation to represent all equivalent solutions in a unique form so as to reduce the problem complexity. JADE outperforms both the local stochastic approach Casanova and the genetic algorithm based approach SGA, indicating its efficiency in the domain of discrete optimization.

Although the proposed methodology for representing a discrete optimization problem in continuous domain is applicable directly only to combinatorial auctions, we believe that this technique can be extended to a large set of discrete optimization problems that involve the sorting or permutation of elements.

REFERENCES

- [1] C. A. Coello-Coello, and G. Lamont, *Applications of Multi-Objective Evolutionary Algorithms*. World Scientific: Singapore,; 2004.
- [2] Dasgupta D, and Michalewicz Z. *Evolutionary Algorithms in Engineering Applications*. Springer: 1st edition; 2001.
- [3] R. Storn and K. Price, "Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces," *Journal of Global Optimization*, vol. 11, pp. 341-359, 1997.
- [4] K. V. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization*, 1st Ed., Springer, Dec., 2005

- [5] J. Liu and L. Lampinen, "Adaptive Parameter Control of Differential Evolution," *Proc. of the 8th International Mendel Conference on Soft Computing*, pp. 19-26, 2002.
- [6] F. Xue, A. C. Sanderson, P. P. Bonissone, and Graves RJ, "Fuzzy logic controlled multi-objective differential evolution," *Proc. of the IEEE International Conference on Fuzzy Systems*, pp. 720-725, 2005.
- [7] A. K. Qin and P. N. Suganthan, "Self-adaptive differential evolution algorithm for numerical optimization," *Proc. of the IEEE Congress on Evolutionary Computation* pp. 1785-1791, 2005.
- [8] J. Brest, S. Greiner, B. Boskovic, M. Mernik and V. Zumer, "Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 6, pp. 646-657, 2006.
- [9] V. Avasarala, T. Mullen., D. L. Hall., and A. Garga, "MASM: market architecture or sensor management in distributed sensor networks", *SPIE Defense and Security Symposium*, Orlando FL, 2005, pp 5813 - 30.
- [10] T. Mullen, V. Avasarala, and D.L. Hall, "Customer-Driven Sensor Management", *IEEE Intelligent Systems*, vol. 21, no. 2, pp 41 - 49, 2006.
- [11] W. E. Walsh, M. Wellman and F. Ygge, "Combinatorial Auctions for supply chain formation", *Proc. of ACM Conf on Electronic Commerce*, pp 260-269, 2000.
- [12] A. Das, and D. Grosu, "A Combinatorial Auction-Based Protocols for Resource Allocation in Grids", *19th IEEE International Parallel and Distributed Processing Symposium*, 2005.
- [13] M. Rothkopf, A. Pekec, and R. Harstad, "Computationally manageable combinatorial auctions", *Management Science*, pp 1131-1147m vol. 44, no.8, 1998.
- [14] ILOG CPLEX, <http://www.ilog.com/products/cplex/>, ILOG Inc.
- [15] A. Andersson, T. Mattias, Y. Fredrik, "Integer programming for combinatorial auction winner determination", *Proc. Of the Fourth International Conf. Multi-Agent Systems*. pp 39-46, Boston, MA, 2000.
- [16] T. Sandholm., S. Subhash., A. Gilpin, and D. Levine, "CABOB: A fast optimal algorithm for winner determination in combinatorial auctions," *International Joint Conferences on Artificial Intelligence*, pp. 1002-1008, 2001.
- [17] H. Hoos and C. Boutilier, "Solving combinatorial auctions using stochastic local search", *Proc. of the Seventeenth National Conf on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pp 22-29, 2000.
- [18] V. Avasarala, H. Polavarapu, T. Mullen, "An Approximate Algorithm for Resource Allocation using Combinatorial Auctions", *Proc. of the IEEE/WIC/ACM international conference on Intelligent Agent Technology*, pp. 571-578, 2006.
- [19] Jingqiao Zhang and Arthur C. Sanderson, "JADE: Self-Adaptive Differential Evolution with Fast and Reliable Convergence Performance," *Prof. of the IEEE Congress on Evolutionary Computation*, Sept 2007, Singapore.
- [20] J. Kennedy and R. C. Eberhart. Particle swarm optimization. *Proc. of IEEE International Conference on Neural Networks*, Piscataway, NJ. pp. 1942-1948, 1995.
- [21] J. Kennedy and R. C. Eberhart, "A Discrete Binary Version of Particle Swarm Optimization," *Proc. of the Conf. on Systems, Man, and Cybernetics*, pp. 4104-4109. Piscataway, NJ. 1997.
- [22] A. P. Engelbrecht, and G. Pampara, "Binary Differential Evolution Strategies," *Proc. of the IEEE Congress on Evolutionary Computation*, Sept 2007, Singapore.
- [23] K. Veeramachaneni, L. Osadciw, G. Kamath, "Probabilistically Driven Particle Swarms for Discrete Multi Valued Problems: Design and Analysis", IEEE Swarm Intelligence Symposium, Hawaii, April 1-5, 2007
- [24] E. Mezura-Montes, J. Velázquez-Reyes and C. A. Coello Coello, "A Comparative Study of Differential Evolution Variants for Global Optimization," *Genetic and Evolutionary Computation Conference*, pp. 485-492, Seattle, Washington, 2006.
- [25] J. Vesterstroem and R. Thomsen, "A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems," *Proc. of the IEEE Congress on Evolutionary Computation*, pp. 1980-1987, Portland, 2004.