Stochastic Greedy Algorithms

A Learning-Based Approach to Combinatorial Optimization

Viswa Viswanathan
Stillman School of Business
Seton Hall University
South Orange, NJ, 07079
viswa.viswanathan@shu.edu

Anup K Sen
Management Information Systems
Indian Institute of Management
Calcutta
D. H. Road, Kolkata 700104, India
sen@iimcal.ac.in

Soumyakanti Chakraborty
Information Systems Area
XLRI School of Business and HR
Jamshedpur, India
soumyakc@xlri.ac.in

Abstract **- Research in combinatorial optimization initially focused on finding optimal solutions to various problems. Researchers realized the importance of alternative approaches when faced with large practical problems that took too long to solve optimally and this led to approaches like simulated annealing and genetic algorithms which could not guarantee optimality, but yielded good solutions within a reasonable amount of computing time. In this paper we report on our experiments with stochastic greedy algorithms (SGA) – perturbed versions of standard greedy algorithms. SGA incorporates the novel idea of *learning from optimal solutions*, inspired by data-mining and other learning approaches. SGA learns some characteristics of optimal solutions and then applies them while generating its solutions. We report results based on applying this approach to three different problems – knapsack, combinatorial auctions and single-machine job sequencing. Overall, the method consistently produces solutions significantly closer to optimal than standard greedy approaches. SGA can be seen in the space of approximate algorithms as falling between the very quick greedy approaches and the relatively slower soft computing approaches like genetic algorithms and simulated annealing. SGA is easy to understand and implement -- once a greedy solution approach is known for a problem, it becomes possible to very quickly rig up a SGA for the problem. SGA has explored only one aspect of learning from optimal solutions. We believe that there is a lot of scope for variations on the theme, and the broad idea of learning from optimal solutions opens up possibilities for new streams of research.**

Keywords- greedy algorithms; stochastic approaches; approximate solutions; knapsack problem; combinatorial auctions; single-machine scheduling; machine learning

I. INTRODUCTION

"Greedy" solutions are commonplace in the field of combinatorial optimization for obtaining very quick solutions to complex problems. For example, the unconstrained knapsack problem (UKP) is known to be NP-complete, but there exists a greedy algorithm with $O(N^2)$ time complexity that yields very good solutions in practice. In general, greedy algorithms do not guarantee optimal solutions. In this paper, we elaborate on the idea of stochastic greedy algorithms first presented in [31].

In general terms, a greedy algorithm tackles a problem in several steps. At each step, the algorithm chooses the locally most attractive option with no concern for its effect on global optimality. Greedy algorithms are usually very simple and intuitive. In the Traveling Salesperson Problem (TSP) [12], the problem is to start at a city and visit n-1 other cities and return to the original city while traversing the minimal distance. A greedy algorithm for the TSP is straightforward – at each stage, simply travel to the closest unvisited city and continue this process till the tour is complete. In the Transportation Problem (TP) [17], we are given a set of requirements for goods to be satisfied from stocks available in various warehouses. The unit transportation cost from each warehouse to each demand point is also given and the problem is to satisfy the demands while incurring minimal cost. Vogel's Approximation Method [17] is a greedy algorithm that first finds the warehouse-demand point combination with the lowest unit transportation cost, satisfies the demand to the extent possible, and continues in similar vein till all demands are satisfied (or all supplies are exhausted).

Greedy algorithms are useful when the time available to solve a problem is severely limited. In the space of solution approaches to combinatorial optimization problems, greedy approaches can be seen as lying at one end of the spectrum with optimal algorithms lying at the other extreme. In the middle are approximate algorithms like genetic algorithms and simulated annealing. As we move from the greedy algorithms to optimal algorithms, the solution quality increases with a concomitant increase in the solution time.

In this paper, we elaborate on the results we presented in [31] on *stochastic greedy algorithms* (SGA). Whereas greedy algorithms choose the next step deterministically based solely on what is locally best, SGA, a variant of greedy algorithms, selects it stochastically. In other words, rather than the probability of the best available option being selected being 1, the algorithm uses a probability distribution to select the next step. It selects the next step as the n^{th} best available option with probability p(n). SGA generates many solutions and returns the best one as the output of the algorithm.

How do we determine the probability distribution that specifies p(n)? In seeking quick and good, but not necessarily optimal, solutions to combinatorial optimization problems, researchers have thus far hardly adopted the idea of learning from optimal solutions. We introduce the idea, and showcase the use of a data-mining inspired approach to learn from optimal solutions the probability distribution to use in SGA.

Learning the probability distribution involves solving many problem instances up front to optimality -- which imposes a large fixed cost. SGA is thus only good in situations where this fixed cost can be amortized over many problem instances. Therefore, SGA only makes sense when many problem instances have to be solved on an ongoing basis and the time allowed for solving each instance is small. This could arise for example, when solving the problem is part of a larger business process in a real-time transaction processing system where the on-line user cannot be kept waiting for too long and yet the response to the user's request involves solving a moderately large non-trivial combinatorial optimization problem. With the proliferation of complex web-based transactional processing systems, this scenario is only likely to become increasingly common.

We have experimented extensively with three combinatorial optimization problems – the Unbounded Knapsack Problem (UKP), Single Unit Combinatorial auction (CA) and Single machine sequencing with quadratic penalties (QPSD) , and obtained very encouraging results. These problems represent a good range because the greedy approach is extremely effective for the UKP and very ineffective for the SQP. The combinatorial auction problem falls in between.

On the UKP, we have obtained solutions consistently within 0.02% of the optimal. Our results on the other two problems are also very encouraging and establish SGA as a viable alternative in the pool of soft computing approaches. More research is definitely needed to understand the nuances and to establish performance parameters more rigorously. Nevertheless, the results we present prove conclusively the viability of the approach.

Learning from optimal solutions is a novel, useful and generic idea that opens up exciting new unions between statistics and combinatorial optimization. Whereas we have demonstrated in this paper only a small aspect of learning from optimal solutions, there is clearly unlimited scope to exploit this idea in the search for good quick solutions to combinatorial optimization problems.

In section II we discuss prior work in related areas and present, in section III a generic domain-independent description of SGA. In subsequent sections we discuss the specifics of our application of SGA to the UKP, CA and QPSD problems and the corresponding empirical findings. We conclude the paper with a summary and a discussion of the scope for further work.

II. RELATED WORK

Greedy algorithms [14] represent natural ways of quickly finding good solutions to combinatorial optimization problems [19]. In rare cases, [14], greedy approaches can even guarantee optimal solutions. Greedy algorithms use deterministic steps in that they select the next course of action by choosing the locally best option available. Stochastic algorithms ([10], [11]), on the other hand, use probabilistic elements to alter the steps of the algorithm. Blending the two approaches lies at the heart of SGA. Although researchers have looked at stochastic local search approaches ([4], [10], [11] and [13]), prior research has not explored the pros and cons of stochastic perturbations of known and new greedy approaches.

We use the knapsack problem, single unit combinatorial auctions and a class of single machine sequencing problems to demonstrate the utility of SGA. Knapsack problems have been widely studied in ([16], [21]). The Unbounded Knapsack Problem (UKP) is known to be NP-hard. Greedy approaches to knapsack problems have been discussed in [16]. A new algorithm for finding exact solutions to UKP can be found in [22].

Auctions have been in use since antiquity. The commonest format has been the ascending auction, also known as the 'English' auction. The first major work on auction theory is that of Vickrey [30] who recommended the adoption of second price sealed bid auctions (later called Vickrey auctions). His ideas were extended to combinatorial auctions by Clarke and Groves ([3],[8]). In their scheme, bidders submit their valuations of packages, and the seller solves the revenue maximization problem, known as Winner Determination Problem (WDP) and allocates the bundles. Solving WDP with dynamic programming was proposed by [25]. Two approaches, CASS [5] and CABOB [26] are the prominent heuristic search techniques to solve large instances of WDP optimally for the single unit case. Both these approaches employ Depth-First Branch-and-Bound (DFBB) but they differ in the formulation of their search space. Both these algorithms may take a long time for solving large instances optimally. For the methodical evaluation and comparison of algorithms for solving WDP, Kevin Leyton-Brown et al. [15] designed a suite of distribution families called CATS 2.0 (http://cats.stanford.edu) for generating realistic, economically motivated combinatorial bids in a number of broad real world applications. With the proliferation of on-line auction situations, it is conceivable that there will be an increasing need to obtain reasonably good solutions quickly to CA and related problems.

Single machine sequencing problems [20] are generally known to be NP-hard [23]. The presence of sequence-dependent setup times makes the sequencing problem with quadratic penalties ([28], [29]) very difficult to solve [27]. Greedy approaches to the single machine sequencing problem with quadratic penalties and setup times (QPSD) are not popular yet. The best exact approach reported thus far [18] can solve problems that have only up to 22 jobs. Therefore, providing good solutions to larger instances serves to extend the envelope for this problem.

Machine learning through neural networks has been applied to optimization problems [1]. However, machine learning based on the analysis of optimal solutions to learn their characteristics and then augmenting the process of generating solutions with the resultant knowledge has not been effectively tried before. This paper shows clearly that the approach has promise.

III. GENERIC DESCRIPTION OF SGA

An instance of an *optimization problem* [19] is a pair *(F, c)* where *F* is any set, the domain of feasible points and *c* is the cost function, a mapping: $c: F \rightarrow R^1$. The problem is to

find an $f \in F$ for which $c(f) \leq c(y)$ for all $y \in F$ for a minimization problem (or to find an $f \in F$ for which $c(f) \geq c(y)$ for all $y \in F$ for a maximization problem).

When the set F has a finite number of points, the problem becomes a *combinatorial optimization problem*. A solution procedure that guarantees the best f in the above sense is an *exact* procedure; other procedures are *approximate*.

EXAMPLE 1: In the Unbounded Knapsack Problem (UKP), we are given a knapsack with weight-capacity K, and N items, with item i having weight w_i and value v_i, with each item available in unlimited quantity. The objective is to fill the knapsack with q_i units of the i^{th} item in such a way that the value of the items in the knapsack is maximized. Here

$$F = \{(q_1, q_2, \ldots, q_N) : \sum_{i=1}^{i=N} q_i w_i \leq K\} \quad (1)$$

and

$$c(q_1, q_2, \ldots, q_N) = \sum_{i=1}^{i=N} q_i v_i \quad (2)$$

EXAMPLE 2: In single unit combinatorial auctions, there is only one unit of each item. Bidders place bids on the items or the combination of items they desire, and the auctioneer determines the winning allocation, *i.e.* the set of winning bids. In determining the winning bids the objective is to select a feasible set of bids such that no item is allocated to more than one bid (no overlapping items) and revenue is maximized. Let there be M distinct items and N bids, and let bid B_i has quoted price v_i on a non-empty bundle $S \subseteq M$ of items. In this case: F = {*feasible set of bids with no overlapping items*} and

$$c(x : x \in F) = \sum_{i=1}^{N} u_i \quad (3)$$

where $u_i = v_i$ if bid $B_i \in x$ and 0 otherwise.

EXAMPLE 3: In the Single Machine Sequencing with Quadratic penalties on job completion times and Sequence Dependent Setup times (QPSD) problem [27], there are N jobs, J_i , $i = 1..N$ with J_i having processing time a_i , penalty coefficient q_i and setup times $s_{i,j}$ (being the setup time for J_j when it is immediately preceded by J_i, and $s_{0,j}$ is the setup time for J_i when it is the first job in the sequence). The objective is to find the schedule that minimizes the total cost. Each feasible schedule is a permutation of 1..N and therefore, in this case F = *{all permutations of 1 ... N} and*

$c(x) = \sum_{i=1}^{N} q_i t_i^{\,2}$ where t_i is the completion time for J_i as per permutation x.

It is common to view the solution procedure for a general optimization problem as starting from a given point in F and then moving step by step towards the final solution (optimal or otherwise). For combinatorial optimization problems, a point in the set F is usually determined through a systematic process of construction involving several stages. For example:

- In UKP, each member of F represents one feasible way of filling the knapsack. Constructing one feasible solution involves selecting items one by one and determining how many pieces of each to take. Here, we could see a feasible solution as being constructed through steps with each step involving the selection of an item and a quantity such that the weight added by this item, when combined with the weights of items already added in prior steps, does not exceed the capacity of the knapsack..
- In CA, a member of F represents a feasible set of bids with non-overlapping items. Here, constructing an element of F can be seen as involving a series of steps with each step selecting a bid which does not have any overlapping items with any bid already selected.
- In QPSD, a member of F is any valid permutation of jobs, and creating one could be seen as a series of steps with each step involving the selection of a job which has not already been selected.

Having laid down the fact that creating a member of F involves a process of constructions having several steps, it is now possible to describe abstractly both the greedy approach and SGA. In the greedy approach, we first identify an intuitive measure of attractiveness of each possible step. This measure varies from domain to domain and we will describe the actual measure used for each problem domain when we discuss the domain separately in later sections. At each step in the process of constructing a feasible solution, we choose the step that seems most attractive according to this intuitive estimate. Thus, for UKP, we first choose the item that seems most attractive and take as many units of it as will fit. We then choose as many units of the next best item as will fit and take as many units as possible and so on till no more items will fit. For CA, we first choose the most attractive bid and then choose the most attractive bid from those that remain which do not have an overlap with bids already selected. We go on like this till no more bids are available. For QPSD, we simply order the jobs by their attractiveness with the most attractive job as the first. The generic version of the greedy algorithm is shown below. It is written from the perspective of a maximization problem and can be easily modified for a minimization problem.

We use the following notation:

P	A combinatorial optimization problem
F	The set of feasible solutions to P
a_i	$1 \leq i \leq N$, all possible actions which can be used to construct any feasible solution in F. Each action can be used at most once in building one element of F
r_i	A measure of attractiveness of action a_i , $1 \leq i \leq N$ *(higher is better)*
E_S	The set of eligible actions, given that the actions contained in set S have already been chosen

p_i Probability of choosing the i^{th} most attractive action from E_S (this is used only in SGA)

L Number of trials for SGA

```
Algorithm Greedy {
    S =empty set
    Initialize Es to set of eligible actions
    While Es is not empty {
        From the actions in Es select action ai that
        corresponds to the maximum ri
        Add ai to S
        Remove ai and all ineligible actions from ES
    }
    Output the actions in S
}
```

Figure 1. Algorithm Greedy

In the greedy approach we choose the next step deterministically as the best available step at that point. In SGA, we perform this step stochastically, by selecting at each stage the i^{th} best available step with probability p_i .We generate many solutions in this process and select the best of these as the output. SGA is driven by a probability distribution. The details of how the probability distribution is arrived at are specific to each problem domain and we will describe those when we look at each problem domain separately.

```
Algorithm SGA {
    best_sol = 0
    best_s = empty set
    Repeat the following L times {
        S =empty set
        Initialize Es to set of eligible actions
        While Es is not empty {
            Select a random i using probability distribution
            pi
            From the actions in Es select action ai that
            corresponds to the i^th highest ri
            Add ai to S
            Remove ai and all ineligible actions from ES
        }
        sol = objective function value corresponding to the
        actions in S
        If sol > best_sol {
                best_sol = sol
                best_s = S
        }
    }
    Output the actions in best_s
}
```

Figure 2. Algorithm SGA

IV. SGA APPLICATION TO THE UNBOUNDED KNAPSACK PROBLEM

The problem statement for UKP appears in section III. To implement the greedy approach for UKP, we need a specification of the *attractiveness* r_i, $1 \leq i \leq N$. Intuitively, the "bang for the buck" ratio of v_i/w_i looks like a good measure of the attractiveness of an item and in fact leads to good greedy solutions.

The greedy approach is to order the items in non-increasing order of the ratio v_i / w_i and then to fill the knapsack with as many units of the first item as can fit, and then as many units of the next lowest numbered item that will fit, and so on, till the knapsack is full (that is, the residual weight capacity is less than the weight of the lightest item). In doing this, at each stage we are taking the locally most attractive step, without considering its global effects. It could turn out, for example, that the greedy approach is unable to fill the knapsack completely, but that taking one less unit of one of the items currently in the knapsack would enable us to fill the knapsack completely, albeit with more units of a lower valued item, but with a larger total value. It is for this reason that the greedy solution cannot guarantee optimality.

In UKP there are N items and therefore a maximum of N possible actions at each step. In order to implement SGA for UKP, we need to specify the probability distribution, p_i , $1 \leq i \leq N$ which gives the probability with which the i^{th} most attractive action available is to be chosen. The logic of SGA is that whereas the greedy approach always picks the most attractive step available while constructing a solution, SGA determines this stochastically. Instead of always picking the most attractive item, we select the next item based on a probability distribution. Having selected the item to be used, we next need to decide on how many units of the item should be picked. It is not necessary to fill the knapsack with the maximum number of units possible for the chosen item. Once again we choose this probabilistically. Items are chosen in this fashion till no more can be added to the knapsack. This concludes a single trial. Several trials are performed and the best solution is chosen.

At the stage of selecting the next item, it seems reasonable to assume that the probability of picking items with higher attractiveness should be higher because it is expected that higher the attractiveness, higher is the chance of striking an optimal solution. Likewise, at the stage of choosing the quantity for the selected item, the chance of picking the maximum possible quantity should be highest.

We now describe the procedure we adopted for introducing stochasticity into the greedy approach for the knapsack problem. In the standard greedy approach where the next item to be allocated is chosen strictly according to the best value-to-weight or v_i / w_i ratios, and the maximum possible quantity of the selected item is used. In SGA, we make both of these choices, namely the choice of item and the quantity of the chosen item probabilistically.

We derived the probability distribution empirically by solving many problems to optimality and then learning from these optimal solutions. The dynamic programming solution

procedure for optimally solving the knapsack problem (Gilmore and Gomory [7]) exploits the Bellman Optimality principle.

As explained in the introduction, we introduce the novel idea of learning from optimal solution and using the knowledge thus derived in a stochastic process of generating solutions. This approach can be seen to be inspired by "learning" as applied to data mining. We describe the learning process in detail in the next paragraph. Broadly, the approach relies on generating optimal solutions to a large number of instances of UKP. Once we have optimal solutions to a large number of instances, we seek patterns in these. In this paper we rely on the large body of optimal solutions to calculate the probability with which the best available piece is selected, the probability of the second best available piece, and so on. We also calculate the probability of the optimal solution containing the maximum number of units of the selected piece, 1 less than the maximum and so on. Once we have these, we can then use these probabilities to generate a large number of random solutions and choose the best among them. We based the calculations on optimal solutions to a total of 500 problem instances with N varying from 50 to 250. We lumped problems with different values of N together because we did not find any significant differences in the probabilities when we calculated them separately for different values of N.

We now describe the procedure for learning the probability distributions. Consider a knapsack problem with capacity 20, and 5 items with weights w_i = {8, 3, 10, 5 and 2} in non-increasing order of their value-to-weight ratios (we ignore the actual values for this discussion). Suppose the optimal solution x_i = {1, 0, 1, 0 1} (one unit each of items 1, 3 and 5). Note that this differs from the greedy solution which would be {2, 1, 0, 0, 0}. Looking at this optimal solution, we find that initially when the knapsack is empty, all of the items are eligible for consideration and the optimal solution actually used the best available item, namely the first, although it does not use the maximum quantity possible – two units would have fitted into the knapsack, but the optimal solution uses only one unit. At the next stage, the residual knapsack capacity is 12 (having allocated one unit of item 1). Even at this stage, the residual capacity is sufficient for all the remaining items to be eligible for consideration – it can hold at least one unit of each of them. However, we see that the optimal solution for the sub-problem did not choose the best item and instead chose only the third best item (namely item 3). Only one unit of this item could fit and hence the maximum allowable number of units were used. The residual knapsack capacity now is 2 and the optimal solution now chose the best item available (only item 5 is eligible for consideration now because only it can fit) and the maximum allowable quantity, namely 1, was used.

We did the above analysis for each optimal solution and calculated the probability of the j^{th} eligible item being actually chosen, and also noted the probability of the number of units of the chosen item used in the optimal solution deviating by an amount *d*, *d* = 1, 2, 3, … from the maximum amount that would fit into the residual capacity.

In this way we calculated the probability p_j of the item with the j^{th} highest ratio being chosen as the next item. Similarly we also calculated the probability $q_{j,k}$ of the number of units of the selected item *j* being less than the maximum possible number by k units.

Figure 3 shows the algorithm for applying SGA to UKP.

Algorithm SGA_UKP

Re-order the *N* items such that $v_1/w_1 \geq v_2/w_2 \geq .... \geq v_N/w_N$

$best\_val \leftarrow 0$

$best_k \leftarrow 0, k = 1, 2, ... N$

Repeat *numtrials* times {

  $capacity \leftarrow K$

  $curpos \leftarrow 1$

  $sga\_value \leftarrow 0$

  while $(capacity \geq \min(w_i), i = 1..N))$ {

    Randomly select a position *j* according to the chosen probability distribution for the position of the next item relative to *curpos*

    Starting from *curpos* skip the first *j* items whose weights are not greater than *capacity*. Let *k* be the index of the next item whose weight is not greater than *capacity*. If this causes a spillover beyond *N*, then search backwards for the first item whose weight is not greater than *capacity*

    $maxunits \leftarrow floor(capacity/w_k)$

    Randomly select a number *m* according to the chosen probability distribution for the quantity of the next item relative to *maxunits*

    Set sol_k the number of units of the k^{th} item in the solution to *max(1, maxunits - m)*

    $curpos \leftarrow k - 1$

    $capacity \leftarrow capacity - sol_k * w_k$

    $sga\_value \leftarrow sga\_value + sol_k * v_i$

  }

  if (*sga_value* > *best*_val)

    $best\_val = sga\_value$

    $best_k \leftarrow sol_k, k = 1..N$

}

Figure 3. Algorithm SGA-UKP

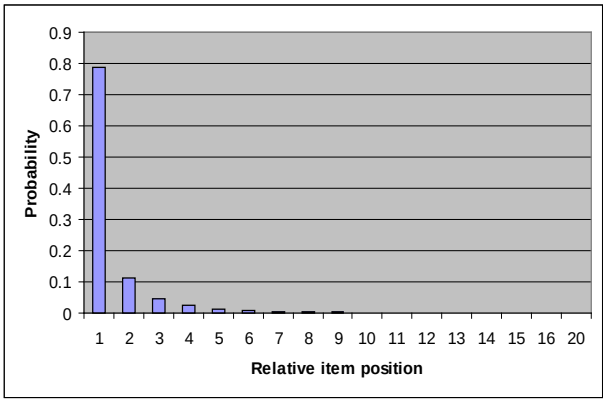

Figure 4. Example of learned probability distribution for position of next item relative to current item (based on 500 random instances)

Figures 4 and 5 show the probability distributions we obtained experimentally for p_j and q_{jk} respectively. The first bar on Figure 4 shows for example that almost 80% of the time the optimal solution chooses the next item as the one with the highest value-to-weight ratio. The second bar shows that there is a close to 10% chance that this is the second best item. Similarly, the first bar in Figure 5 shows that about 36% of the time the optimal solution will utilize the maximum number of units of the selected item. The second bar shows that about 18% of the time, the optimal solution will use one unit less than the maximum possible and so on. Given the probability distributions being used, and an optimal solution, it is easy to calculate the probability that SGA will generate the given optimal solution.

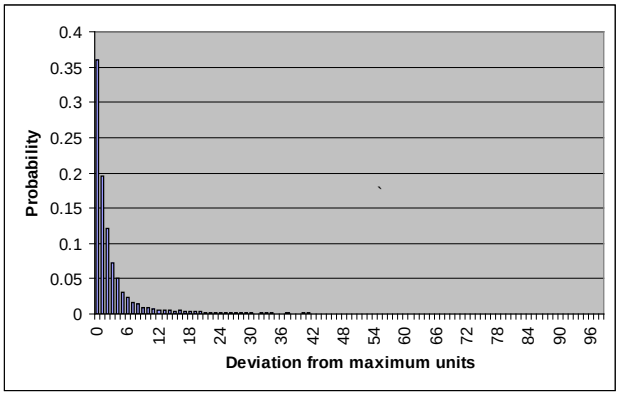

Figure 5. Example of learned probability distribution for extent of deviation of quantity used from maximum possible (based on 500 random instances)

Suppose the probability that SGA will generate an optimal solution in a single trial is *p*, then the probability that it will generate a non-optimal solution in a single trial is *(1-p)*. If there are *L* trials, the probability that each trial generates a non-optimal solution is $(1-p)^L$. Therefore the probability that at least one of the trials generates an optimal solution is

$$1-(1-p)^L$$

As is well known, this number can be surprisingly close to 1 for even fairly low values of *p*. This probability estimate is somewhat lower than then real value, as a problem could have multiple optimal solutions. Also, it is possible for a given solution to be generated in more than one way by our algorithm.

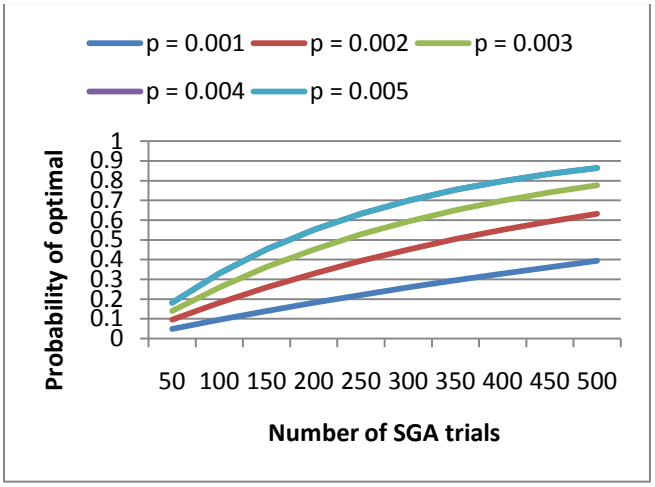

Figure 6. Probability of finding optimal solutions in SGA as number of trials increses

To demonstrate the probability calculation, we revert to the example used earlier. Suppose we have a knapsack problem with capacity 20 and 5 items with weights w_i = {8, 3, 10, 5 and 2} in non-increasing order of their value-to-weight ratios (we ignore the values for this discussion). Suppose the probability of distribution for item position is {0.6, 0.3, 0.1}. This means that the best item available was chosen 60% of the time, the second best 30% of the time and the third best 10% of the time. Suppose the probability distribution for the deviation from the maximum is {0.7, 0.25, 0.05}. This means that the maximum number of units possible would be used 70% of the time, one less than the maximum would be used 25% of the time and two less than the maximum would be used 5% of the time.

Suppose the optimal solution x_i = {1, 0, 1, 0 1} (one unit each of items 1, 3 and 5). With these numbers, the probability of SGA finding the optimal solution in a single trial is the product of the probability of selecting each of the actual items chosen and the probabilities of the correct quantities being chosen. The first element of the optimal solution is a choice of one unit of the best item. The probability of this happening is the probability of the first item being chosen – which is 0.6 times the probability that the deviation from the maximum number of units being 1 (since 2 units will fit, but only one unit is represented in the optimal solution) which is 0.25. Calculating in this way we find the probability as (0.6*0.25)*(0.1*0.7)*(0.6*0.7) = 0.0041. Therefore the probability of generating an optimal solution in 250 trials will be about 0.63. Since the computation and the results are similar for other problem

domains, we have not shown this probability for CA and QPSD problems.

Table I and II show the results obtained on UKP instances with varying numbers of items, as well as different instance types based on [21], where the weights and values are weakly correlated (easy) and strongly correlated (harder),

TABLE I. AVERAGES OF 100 RUNS FOR UKP USING LEARNED PROBABILITY DISTRIBUTIONS IN WEAKLY CORRELATED CASE

N	Weakly correlated (easy)						
	Greedy %dev	*SGA_UKP*					
		50 trials		*100 trials*		*200 trials*	
		%dev	*Prob*	*%dev*	*Prob*	*%dev*	*Prob*
50	0.410	0.097	0.321	0.061	0.367	0.012	0.564
100	0.401	0.112	0.243	0.079	0.310	0.013	0.545
150	0.425	0.100	0.266	0.073	0.300	0.015	0.496
200	0.435	0.100	0.263	0.063	0.290	0.018	0.439
250	0.458	0.103	0.277	0.063	0.275	0.019	0.418

TABLE II. AVERAGES OF 100 RUNS FOR UKP USING LEARNED PROBABILITY DISTRIBUTIONS IN STRONGLY CORRELATED CASE

N	Strongly correlated (hard)						
	Greedy %dev	*SGA_UKP*					
		50 trials		*100 trials*		*200 trials*	
		%dev	*Prob*	*%dev*	*Prob*	*%dev*	*Prob*
50	0.430	0.172	0.180	0.110	0.171	0.061	0.256
100	0.490	0.200	0.151	0.146	0.141	0.090	0.251
150	0.521	0.210	0.128	0.171	0.130	0.113	0.183
200	0.541	0.225	0.113	0.183	0.222	0.142	0.121
250	0.580	0.251	0.107	0.175	0.104	0.150	0.100

All the data are based on an average over 100 problem instances. For each instance, we also calculated the probability of SGA obtaining the optimal solution, and the tables show these as well. Across all the figures in Tables I-II, the deviation from optimal for the greedy solution is, at the minimum, 2.5 times the SGA deviation and the maximum is 35 times.

V. APPLICATION TO COMBINATORIAL AUCTIONS

In single unit combinatorial auctions, there is only one unit of each item. Bidders place bids on the items or the combination of items they desire, and the auctioneer determines the winning allocation, *i.e.* the set of winning bids. In determining the winning bids the objective is to select a feasible set of bids such that no item is allocated to more than one bid (no overlapping items) and revenue is maximized. The formal description of the problem is given in section III.

Individual items have no prices associated with them. Prices are only associated with bids and each bid can be for many items. Accordingly a useful measure of attractiveness of a bid is its price per item. Thus suppose a bid has price 200 and is for four different items. The price per item for this bid is 50. Suppose there is another bid whose price is 80, but is for just a single item. Then the second bid is in some sense preferable to the first because its price per item is higher. Table III below shows an example of CA with 10 items and 5 bids.

TABLE III. SINGLE UNIT COMBINATORIAL AUCTION WITH 10 ITEMS AND 5 BIDS

Bid no	Price	Items in bid	Attractiveness
1	100	{8, 9, 10}	33.33
2	125	{6, 9, 2, 1}	31.25
3	75	{4, 6}	37.5
4	80	{5, 7, 1}	26.66
5	30	{6}	30

The greedy approach for CA therefore is very straightforward. Simply pick the most attractive bid first and then continue to pick the most attractive remaining bid which has no overlapping items with any bids already chosen. In the above example, first we would choose bid 3. Then we can choose bid 1. Now, since items 4, 6, 8, 9 and 10 have already been chosen, only bid 4 can be chosen because of item overlap considerations. The greedy solution is 255, which also happens to be the optimal solution,

For learning the probabilities, we ran CA to optimality using CASS [5]. We then analyzed the optimal solutions generated by CASS. For each optimal solution generated by CASS, we first considered the bids in the optimal solution in their order of their attractiveness. We then tallied the number of times the optimal solution picked the best admissible bid, the second best admissible bid and so on. We calculated the probability with which CASS chose the most attractive bid at each stage. Using the above problem as an example, we would see that the optimal solution selected the best available bid at each stage. It is important to note that while analyzing the optimal solutions, we consider only the admissible bids at any stage. For example, it is possible that at some stage the optimal solution uses the fifth best bid overall. However, if at that stage this bid happens to be the best among the *admissible* bids at that stage based on overlaps with bids already selected, then we will consider that the best bid has been chosen. Suppose we perform this analysis over a large number of problems and see that the i^{th} best available bid was chosen n_i times across all the problem instances. Then the probability of SGA choosing the i^{th} best available bid at any point is (N being the number of bids)

$$p_i = \frac{n_i}{\sum_{i=1}^{N} n_i}$$

```
Algorithm SGA_CA {
    best_val = 0;
    best_bids = empty set
    repeat num_trials times {
        selected_bids = empty set
        S = set of all bids
        val = 0;
```

```
        while the set S of bids is not empty {
            randomly select i based on the learned
                probability distribution p_i
            select the i^th most attractive bid b_i from S
            add b_i to selected_bids
            remove b_i and all bids overlapping with b_i from
                S
            val = val + p_i
        }
        If val > best_val {
            best_val = val
            best_bids = selected_bids
        }
    }
    Output best_sol and best_bids
}
```

Figure 7. Algorithm SGA_CA

The probability distribution that we gleaned from optimal solutions is shown in Figure 8.

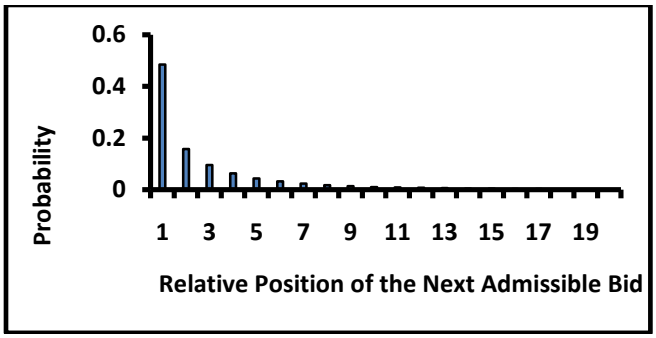

Figure 8. Example of learned probability distribution for relative position of next admissible bid (based on 500 random instances with number of goods varying from 10 to 40 and number of bids between 50 and 200)

The results of running SGA on CA are shown in Table IV. Each row shows the average of 100 random problem instances generated using the standard problem suite generator CATS 2.0 [15]. The results indicate very clearly that even with 50 trials, SGA is able to drastically improve on the greedy solution.

We wanted to see if this impressive performance of SGA was merely the result of the fact that the greedy solution was quite poor in the problem instances tested. We wanted to create a situation where the greedy solution is a lot closer to the optimal solution and then see if SGA can provide benefits even under this scenario that tests SGA more rigorously. We hypothesized that if the number of bids in relation to the number of items is drastically increased, then the greedy solution is likely to come a lot closer to the optimal solution on the average. We expected this because the drastically increased number of bids will make available many more attractive bids than would have been possible with fewer bids. Accordingly we generated random problem instances with a significantly larger number of bids. The results on running SGA on this set are shown in Table V. As we expected, the greedy solution was indeed a lot closer to the optimal. Encouragingly, SGA still managed to improve significantly upon the greedy solution.

TABLE IV. AVERAGES OF 100 RUNS FOR CA USING LEARNED PROBABILITY DISTRIBUTIONS IN CASE OF WEAK GREEDY SOLUTIONS

No. of Goods	No. of Bids	Greedy % dev	SGA			
			50 Trials	100 Trials	200 Trials	500 Trials
			% dev	% dev	% dev	% dev
10	50	19.97	2.39	0.86	0.70	0.29
10	200	11.18	2.28	1.43	0.95	0.29
10	500	6.96	2.01	1.50	1.07	0.49
10	1000	6.75	3.08	2.45	2.14	1.68
12	50	21.50	2.18	1.08	0.41	0.23
12	200	12.58	2.70	1.79	1.09	0.55
12	500	7.34	2.56	1.68	1.19	0.82
12	1000	8.49	4.05	3.50	2.86	2.50
15	50	18.45	2.12	1.22	0.63	0.27
15	200	11.80	2.51	1.83	1.25	0.60
15	500	6.65	2.63	1.95	1.55	1.06
15	1000	9.24	4.82	4.36	3.83	3.25
20	50	27.73	4.08	2.49	1.32	0.46
20	200	15.62	4.56	3.03	2.15	1.46
20	500	10.98	4.54	3.75	3.08	2.43
20	1000	13.38	6.92	6.30	5.58	4.89
26	50	27.07	5.19	4.28	2.85	2.40
26	200	19.09	5.44	4.06	3.15	2.01
26	500	15.25	6.65	5.43	4.54	3.32
26	1000	20.79	11.08	9.98	9.05	8.00
30	50	27.92	6.13	4.59	2.81	1.91
30	200	19.11	6.53	4.85	3.84	2.92
30	500	13.22	5.85	5.16	4.08	3.31
30	1000	19.68	11.39	10.70	9.57	8.81
40	50	31.74	7.71	5.75	3.96	2.67
40	200	20.93	8.99	7.49	5.91	4.61
40	500	15.34	7.65	6.42	5.47	4.68
40	1000	21.18	13.07	12.13	11.28	10.41

We were curious to see if the probability distributions for the problems with lower number of bids and those for the problems with a huge number of bids would be significantly different. It turned out that they were very stable. The probability distribution is shown in Figure 9. Thus, while it might be a good idea to re-learn the probability distributions when the problem parameters change a lot, this finding indicates that in a time crunch nothing much would be lost in using a probability distribution obtained from a set of problem instances with different characteristics.

VI. APPLICATION TO SINGLE MACHINE SEQUENCING

We also studied the performance of SGA on a very hard single machine sequencing problem with quadratic penalties on job completion times and sequence dependent setup times (QPSD) [27]. This is also described in section III. In QPSD, there are N jobs , J_i i = 1..N with J_i having processing time a_i, penalty coefficient q_i and setup times $s_{i,j}$ (being the setup time for J_j when it is immediately preceded by J_i, and $s_{0,j}$ is the setup time for J_j when it is the first job in the sequence). We assume that all values are non-negative integers.

Figure 9. Example of learned probability distribution for relative position of next admissible bid on problems with large nunmber of bids (500 and 1000)

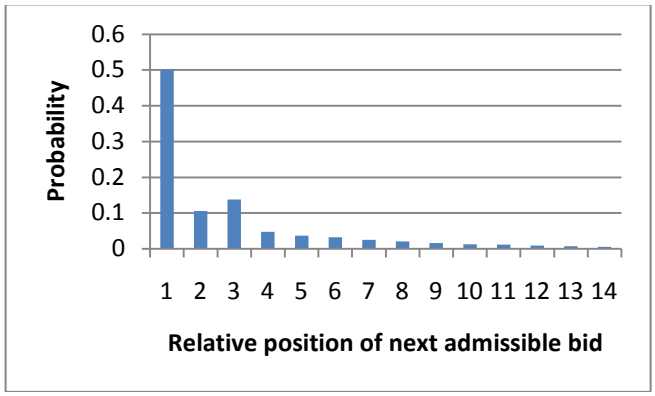

TABLE V. AVERAGES OF 100 RUNS FOR CA USING LEARNED PROBABILITY DISTRIBUTIONS IN CASE OF STRONG GREEDY SOLUTIONS

No. of Goods	No. of Bids	Greedy % dev	SGA			
			50 Trials	100 Trials	200 Trials	500 Trials
			% dev	% dev	% dev	% dev
10	3000	1.27	0.72	0.59	0.44	0.31
10	5000	0.67	0.53	0.43	0.35	0.26
10	8000	0.54	0.38	0.31	0.26	0.18
10	10000	0.37	0.25	0.20	0.16	0.14
12	5000	0.76	0.53	0.46	0.37	0.27
12	8000	0.62	0.47	0.40	0.33	0.23
12	10000	0.31	0.23	0.21	0.19	0.13
15	8000	0.64	0.55	0.50	0.39	0.33
20	10000	1.34	1.10	0.99	0.88	0.78
26	10000	2.15	1.79	1.67	1.59	1.39
26	12000	1.67	1.55	1.47	1.36	1.20
26	15000	1.56	1.36	1.28	1.17	1.01
26	20000	0.91	0.80	0.77	0.70	0.64
30	8000	5.78	3.70	3.44	3.00	2.60
30	10000	3.49	2.76	2.51	2.26	2.03
30	12000	2.31	1.99	1.88	1.77	1.55
30	15000	1.76	1.62	1.53	1.42	1.20
40	4000	6.55	4.61	4.33	3.99	3.63

Let

$$e_{i,j} = s_{i,j} + a_j \forall i \in \{0..N\}, j \in \{1..N\} \quad (4)$$

be the effective processing time for J_j when it is immediately preceded by J_i. Let M be a number such that

$$M > \sum_{j=1}^{N} \max(e_{i,j}), i \in \{0..N\}. \quad (5)$$

The objective is to minimize the total penalty across all jobs, that is, to minimize the weighted sum of the square of completion times. When the setup times are sequence-dependent, the quadratic penalty problem becomes extremely difficult to solve. Drawing from Balas [2], the problem of minimizing the total penalty was formulated by [31] as:

$$Min \sum_{j=1}^{N} q_j t_j^2$$

Subject to:

$$t_j \geq \min\{e_{0,j}, j \in \{1..N\}\} \quad (6)$$

$$t_j + e_{j,k} \leq t_k + M(1 - x_{j,k}),\ j < k,\ \ j,k \in \{1..N\} \quad (7)$$

$$t_k + e_{k,j} \leq t_j + M x_{j,k},\ \ j < k,\ j,k \in \{1..N\} \quad (8)$$

$$t_j \in \left\{0..\sum_{j=1}^{N} \max(e_{i,j}), i \in \{0..N\}\right\} \quad (9)$$

$$x_{j,k} \in \{0,1\},\ j < k,\ \ j,k \in \{1..N\} \quad (10)$$

Constraint 1 addresses the completion time for the first job in the sequence. Constraints 7 and 8 ensure that for any pair of jobs j and k, either j precedes k or k precedes j. We use "j < k" in constraints 7, 8 and 10 to reduce the number of x-variables by half. As in the case of Traveling Salesman Problem, such a formulation may not be efficient to solve in practice using IP solvers.

In [27], it has been shown that the search space for sequencing problems can be modeled as a tree, or as a graph, and those algorithms using the graph search space run faster. For the QPSD problem under the tree formulation, two nodes with the same set of jobs but in different orders and having the same last job will generally not have the same cost because the setup times for the jobs could differ. Nevertheless, the sub trees below them are identical in terms of the structure. Algorithms using the tree search space cannot take advantage of this fact and might wastefully traverse these identical sub-trees more than once. The graph search space has far fewer nodes and offers the potential for faster search. The node count reduction results from the fact that unlike in the tree search space, there could be multiple paths from the root node to any given node, and this helps to avoid replicating the identical sub trees. However, sequence-dependent setup times complicate traditional graph search because the identical sub trees may not have the same costs.

The main feature of graph search algorithms like the graph version of A* [9] is that when these reach the same node through different paths, they retain the path having the lowest cost, discarding any other paths from the root to the node. This approach works fine when the incremental cost from a given node to a goal node is independent of the path by which the node was reached. This is the same as the principle of optimality on which the dynamic programming formulations [12] are based. However, this does not hold for sequence-dependent setup times[18]. For example, consider the following 4 job problem given in Table VI below.

TABLE VI. 4 JOB QPSD PROBLEM

Job	Setup Times				Proc. Times	Penalty Coeff
	1	2	3	4		
1	-	1	1	3	1	2
2	1	-	3	2	4	1
3	5	4	-	10	3	1
4	3	6	9	-	10	1

In this example, it is assumed that the setup time for a job is zero if it is the first in the sequence. Consider the ordered sequence of jobs (1, 2, 3) and (2, 1, 3). Under the graph formulation, a node is represented by the set of completed jobs without regard to the ordering, except for the last job in the sequence. Because the set of jobs and the last job in the two ordered sequences in question are the same, the two are represented by a single node $(\{1,2\},3)$, where the first two jobs form an (unordered) set and the last job is shown separately. The cost when the node is reached through the sequence 1, 2, 3 is 182 and through the sequence 2, 1, 3 is 188. If a traditional graph search algorithm reaches the node through the two different paths considered, it would simply discard the higher cost path 2, 1, 3. However, if we look below this node, we see that the sequence 1, 2, 3, 4 has a cost of 1206, which is higher than the cost of the sequence 2, 1, 3, 4 which is 1088. A traditional graph search algorithm thus runs the risk of missing the optimal solution.

In [18], solutions for QPSD only up to 22-job problems using a memory constrained graph search algorithm have been reported. Increasing the memory limit to 512K nodes, we could solve 30-job problems using PC running Windows XP. We wanted to study how SGA performs on this hard problem. For a simpler problem not involving setup times, Townsend [29] had proposed two sufficient conditions for a given sequence of jobs to be optimal. The first of these involves ordering the jobs by non-ascending order of their p_i/a_i ratios. Being only one of two sufficient conditions for optimality, this ordering cannot guarantee optimal solutions for the simpler problem, but it does provide the basis for very good greedy solutions for that problem. In the absence of any other known greedy approaches to QPSD, we chose to adopt Townsend's heuristic.

Our SGA application to QPSD orders the jobs as above, and at each stage, chooses the job with the highest p_i/a_i ratio. Since UKP and CA have already established the benefit of learning from optimal solutions, we wanted to check and see how a standard discrete probability distribution with the right shape would perform for SGA. The benefit of doing this is that the up-front cost of solving many problem instances to optimality can then be avoided. Accordingly, in our experiments with QPSD, instead of learning the probability distribution from the solutions to optimal solutions, we experimented with both the Geometric and the Binomial distributions (since they can have the proper shape with suitably chosen parameters) and found that the Binomial distribution with a low value for its parameter performed better. The results are given in Table VII. It shows that the results for QPSD are good, but not as impressive as for UKP. It is intuitively clear that SGA can give good results only when the underlying greedy algorithm is reasonably good. Results of SGA application to QPSD - based on 100 trials and averaged over 100 random problem instances for each value of N.

TABLE VII. RESULTS OF SGA APPLICATION TO QPSD (BASED ON 100 TRIALS AND AVERAGED OVER 100 RANDOM PROBLEM INSTANCES FOR EACH VALUE OF N)

N	% deviation from optimal		
	Greedy	SGA – Binomial (p = 0.025)	SGA – Geometric (p=0.8)
10	7.70	1.67	2.07
12	10.02	2.81	3.48
14	11.13	3.70	4.65
16	11.05	4.01	5.14
18	12.50	4.93	6.20
20	13.91	6.10	7.61
22	14.45	7.11	8.34
24	14.41	7.43	8.93
26	15.64	8.31	9.92
28	15.54	8.53	9.75
30	15.71	9.22	10.56

For QPSD we based the greedy approach on a result obtained for a far simpler problem, and its performance was not very good. Nevertheless, we find that SGA is able to improve upon the solution significantly. We need to experiment with learned distributions in this domain too.

VII. CONCLUSIONS

We have proposed a new approximate approach called the Stochastic Greedy Algorithm and presented the results of its application to the Unbounded Knapsack Problem, Combinatorial Auctions and a hard Single Machine Sequencing Problem.

The two major contributions of SGA are

- its combining greedy approaches with stochastic approaches
- its introduction of the idea of learning from the characteristics of optimal solutions to incorporate in a generative approach

In all three domains, SGA provides significant improvements over the greedy solution. Of the three, the results for the single machine sequencing problem are perhaps relatively weak, and one reason for this is that no good greedy approach is known for the problem as of now. One important finding is that standard discrete probability distributions perform quite well and that, if necessary, the costly step to learn the underlying probability distribution can be avoided on occasion. Furthermore, our findings seem to hint that probability distributions are pretty stable and need not necessarily be re-learned when the problem characteristics change.

Our results explore the potential for learning patterns from optimal solutions and applying this learning in the process of generating solutions. There is obviously much more scope to extend this "supervised learning" approach for combinatorial optimization. While analyzing optimal

solutions to learn characteristics, it is possible to assign several other descriptors to each decision point. For example, in the knapsack problem, we could attach the percentage difference between the best available option and the next best one as a descriptor. Once the decisions made in the optimal solution are thus tagged, we effectively have different probability distributions for different states and SGA could sample from a more fine-grained and situation-specific probability distribution. Another approach would be to study numerous optimal solutions and impute decision rules and see how solutions based on such rules perform. Broadly speaking, this learning metaphor can be exploited in numerous ways and certainly opens up new avenues for further work.

REFERENCES

[1] A. Cochocki, R. Unbehauen, Neural Networks for Optimization and Signal Processing, New York: John Wiley, 1993,

[2] Balas, E., 1985. On the facial structure of scheduling polyhedra, *Mathematical Programming*, 24, 179-218

[3] E. H. Clarke, "Multipart pricing of public goods," *Public Choice* (11) 1971, pp 17 - 33.

[4] A. Feldman, G. Provan and A. V. Gemund, Computing minimal diagnoses by greedy stochastic search**,** In *Proc. AAAI* 2008, pp. 911-918.

[5] Y. Fujishima, K. Leyton-Brown, and Y. Shoham, "Taming the Computational Complexity of Combinatorial Auctions: Optimal and Approximate Approaches," in: *International Joint Conference on Artificial Intelligence*, Stockholm, 1999, pp. 548 - 553.

[6] E. C. Freuder, R. Dechter, B. Ginsberg, B. Selman. and E. P. K. Tsang, 1995. Systematic versus stochastic constraint satisfaction. In *Proc. IJCAI 95*, volume 2.

[7] P. C. Gilmore and R. E. Gomory, 1966, The theory and computation of knapsack functions, Operations Research, 14(6), pp 1045-1074

[8] T. Groves, "Incentives in Teams," *Econometrica* (41) 1973, pp 617 - 631.

[9] P. Hart, N. Nilsson and B. Raphael, 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths, *IEEE Trans. Syst. Science and Cybernetics*, SSC4(2):100-107

[10] H. H. Hoos and T. Stutzle, *Stochastic Local Search Foundations and Applications*, 2004, Elsevier.

[11] J. Hromkovic, R. Královič, M. Nunkesser, and P. Widmayer (Eds.), *Stochastic Algorithms: Foundations and Applications*, Proceedings of 4th International Symposium, SAGA 2007, Lecture Notes in Computer Science, Zurich, Switzerland, Sept 13-14, 2007.

[12] D. S. Johnson and L. A. McGeoch, The Traveling Salesman Problem: A Case Study in Local Optimization, In *Local Search in Combinatorial Optimization*, E. H. L. Aarts and J.K. Lenstra (Eds), John Wiley and Sons Ltd, 215-310, 1997.

[13] K. Kask, and R. Dechter, 1999, Stochastic local search for Bayesian networks. In *Proc. AISTAT'99*, 113–122.

[14] J. Kruskal, Greedy algorithm for the minimum spanning tree problem, *Proceedings of the American Mathematical Society*, 48-50, 1956.

[15] K. Leyton-Brown, M. Pearson, and Y. Shoham, Y. "Towards a Universal Test Suite for Combinatorial Auction Algorithms," in: *ACM Conference on Electronic Commerce*, 2000a, pp. 66 -76.

[16] S. Martello and P. Toth, Knapsack Problems: Algorithms and Computer Implementations, John Wiley & Sons, 1990.

[17] M. Mathirajan, and B. Meenakshi, Experimental Analysis of some Variants of Vogel's Approximation Method, *Asia-Pacific Journal of Operational Research* 21(4), 447-462, 2004.

[18] S. A. Mondal and A. K. Sen, 2000. TCBB scheme: Applications to single machine sequencing problems, *Proc AAAI-2000*, pp. 792-797.

[19] K. Papadimitriou and K. Steiglitz K. Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall, Englewood Cliffs, NJ, 1982.

[20] M. Pinedo, Scheduling: Theory, Algorithms and Systems. Prentice Hall. 1995.

[21] D. Pisinger, *Algorithms for Knapsack Problems*, Ph. D. Thesis, Department of Computer Science, University of Copenhagen, Denmark, 1995.

[22] V. Poirriez, N. Yanev and R. Andonov, "A hybrid algorithm for the unbounded knapsack problem", Discrete Optimization, volume 6, 2009, pp. 110-124.

[23] A. H. G. Rinooy Kan, Machine Complexity Problems: Classification Complexity and Computations. Nijhoff, The Hague, 1976.

[24] F. Rossi, P. v. Beek and T. Walsh, Constraint Programming, In *Handbook of Knowledge Representation*, Edited by B. Porter, V. Lifschitz and F. van Harmelen, 2008 , Elsevier B.V.

[25] M. H. Rothkopf, A. Pekec and R. M Harstad. Computationally Manageable Combinatorial Auctions. Management Science, 44(8):1131 – 1147, 1998.

[26] T. Sandholm, "Algorithm for Optimal Winner Determination in Combinatorial Auctions," *Artificial Intelligence* (135) 2002, pp 1 - 54.

[27] A. K. Sen, and A. Bagchi, Graph Search Methods for Non-order-preserving Evaluation Functions: Applications to Job Sequencing Problems, *Artificial Intelligence*, 86(1), 43-73, 1996.

[28] W. Szwarc, M. E. Posner and J. J. Liu, "The single machine scheduling problem with quadratic penalty function of completion times", Management Science. Volume 34, no 2, 1988, pp. 1480-1488.

[29] W. Townsend, The Single Machine Scheduling Problem with Quadratic Penalty Function of Completion Times: A Branch-and-bound Solution, *Management Science*, 24(5), 530-534, 1978.

[30] W. Vickrey, "Counterspeculation, Auctions, and Competitive Sealed Tenders," *Journal of Finance* (16) 1961, pp 8 - 37.

[31] K. V. Viswanathan and A. K. Sen, Greedy by Chance – Stochastic Greedy Algorithms, Proceedings of the Sixth International Conference on Autonomic and Autonomous Systems (ICAS 2010), March 7-13, 2010, Cancun, Mexico, published by IEEE CPS, pp. 182-187.