

A Novel Genetic Algorithm using Helper Objectives for the 0-1 Knapsack Problem

Jun He, Feidun He and Hongbin Dong

Abstract

The 0-1 knapsack problem is a well-known combinatorial optimisation problem. Approximation algorithms have been designed for solving it and they return provably good solutions within polynomial time. On the other hand, genetic algorithms are well suited for solving the knapsack problem and they find reasonably good solutions quickly. A naturally arising question is whether genetic algorithms are able to find solutions as good as approximation algorithms do. This paper presents a novel multi-objective optimisation genetic algorithm for solving the 0-1 knapsack problem. Experiment results show that the new algorithm outperforms its rivals, the greedy algorithm, mixed strategy genetic algorithm, and greedy algorithm + mixed strategy genetic algorithm.

Index Terms

genetic algorithm, knapsack problem, multi-objective optimisation, solution quality

I. INTRODUCTION

The 0-1 knapsack problem is one of the most important and also most intensively studied combinatorial optimisation problems [1]. Several approximation algorithms have proposed for solving the 0-1 knapsack problem [1]. These algorithms always can return provably good solutions, whose values are within a factor of the value of the optimal solution.

In last two decades, evolutionary algorithm, especially genetic algorithms (GAs), have been well adopted for tackling the knapsack problem [2], [3]. The problem has received a particular interest from the evolutionary computation community for the following reason. The binary vector representation is a natural encoding of the candidate solutions to the 0-1 knapsack problem. Thereby, it provides an ideal setting for the applications of GAs [4, Chapter 4].

Empirical results often assert that GAs produce reasonably good solutions to the knapsack problems [5], [6], [7]. A naturally arising question is to compare the solution quality (reasonably good versus provably good) between GAs and approximation algorithms. There are two approaches to answer the question. One approach is to make a theoretical analysis. A GA is proven that it can produce a solution within a polynomial runtime, the value of which is within a factor of the value of an optimal solution. This is a standard approach used in the study of approximation algorithms. Another approach is to conduct an empirical study. A GA is compared with an approximation algorithm via computer experiments. If the GA can produce solutions better or not worse than an approximative algorithm does in all instances within polynomial time, the GA is able to reach the same solution quality as the approximation algorithm does.

The current paper is an empirical study of an GA which uses the *multi-objectivization* technique [8]. In multi-objectivization, single-objective optimisation problems are transferred into multi-objective optimisation problems by decomposing the original objective into several components [8] or by adding helper objectives [9]. Multi-objectivization may bring both positive and negative effects [10], [11], [12]. This approach has been used for solving several combinatorial optimisation problems, for example, the knapsack problem [13], vertex cover problem [14] and minimum label spanning tree problem [15]. This paper focusses on the 0-1 knapsack problem. A novel GA using three helper objectives is designed for solving the 0-1 knapsack problem. Then the solution quality of the GA is compared with a well-known approximation algorithm via computer experiments.

The remainder of the paper is organized as follows. The 0-1 knapsack problem, a greedy algorithm and a GA for it are introduced in Section II. In Section III we present a novel GA using helper objectives. Section IV is devoted to an empirical comparison among several algorithms. Section V concludes the article.

II. KNAPSACK PROBLEM, GREEDY ALGORITHM AND GENETIC ALGORITHM

In an instance of the 0-1 knapsack problem, given a set of n items with weights w_i and profits p_i , and a knapsack with capacity C , the task is to maximise the sum of profits of items packed in the knapsack without exceeding the capacity. More formally the target is to find a binary vector $\vec{x} = (x_1 \cdots x_n)$ so as to

$$\max_{\vec{x}} f(\vec{x}) = \sum_{i=1}^n p_i x_i, \quad \text{subject to} \quad \sum_{i=1}^n w_i x_i \leq C, \quad (1)$$

This work was supported by the EPSRC under Grant No. EP/I009809/1.

Jun He is with Department of Computer Science, Aberystwyth University, Aberystwyth, SY23 3DB, UK (email: jun.he@aber.ac.uk).

Feidun He is with School of Information Science and Technology, Southwest Jiaotong University, Chengdu, China

Hongbin Dong is with College of Computer Science and Technology, Harbin Engineering University, Harbin, China

where $x_i = 1$ if the item i is selected in the knapsack and $x_i = 0$ if the item i is not selected in the knapsack. A feasible solution is an \vec{x} which satisfies the constraint. An infeasible one is an \vec{x} that violates the constraint.

Several approximation algorithms have been proposed for solving the 0-1 knapsack problem (see [1, Chapter 2] for more details). Among these, the simplest one is the greedy algorithm described below. The algorithm aims at putting the most profitable items as many as possible into the knapsack or the items with the highest profit-to-weight ratio as many as possible, within the knapsack capacity.

- 1: **input** an instance of the 0-1 knapsack problem;
- 2: resort all the items via the ratio of their profits to their corresponding weights so that $\frac{p_1}{w_1} \geq \dots \geq \frac{p_n}{w_n}$;
- 3: greedily add the items in the above order to the knapsack as long as adding an item to the knapsack does not exceeding the capacity of the knapsack. Denote the solution by \vec{y} ;
- 4: resort all the items according to their profits so that $p_1 \geq p_2 \geq \dots \geq p_n$;
- 5: greedily add the items in the above order as long as adding an item to the knapsack does not exceeding the capacity of the knapsack. Denote the solution by \vec{z} ;
- 6: **output** the best of \vec{y} and \vec{z} .

This algorithm is a $1/2$ -approximation algorithm for the 0-1 knapsack problem [1, Section 2.4], which means it always can return a solution no worse than $1/2$ of the value of the optimal solution.

The greedy algorithm stops after finding an approximation solution, and it has no ability to seek the global optimal solution. Therefore GAs are often applied for solving the 0-1 knapsack problem.

In order to handle the constraint in the knapsack problem, we use repair methods since they are claimed to be the most efficient for the knapsack problem [4], [16]. A repair method is explained as follows.

- 1: **input** an infeasible solution \vec{x} ;
- 2: **while** \vec{x} is infeasible **do**
- 3: $i =$: **choose** an item from the knapsack;
- 4: set $x_i = 0$;
- 5: **if** $\sum_{i=1}^n x_i w_i \leq C$ **then**
- 6: \vec{x} is feasible;
- 7: **end if**
- 8: **end while**
- 9: **output** a feasible solution \vec{x} .

There are different methods available for *choosing* an item in the *repair* procedure, described as follows.

- 1) *Profit-greedy repair*: sort all items according to the decreasing order of their corresponding profits. Then choose the item with the smallest profit and remove it from the knapsack.
- 2) *Ratio-greedy repair*: sort all items according to the decreasing order of the corresponding ratios. Then choose the item with the smallest ratio and remove it from the knapsack.
- 3) *Random repair*: choose an item from the knapsack at random and remove it from the knapsack.

Thanks to the repair method, all of the infeasible solutions are repaired into the feasible ones. The following pseudo-code is a mixed strategy GA (MSGA) which chooses one of three repair methods in a probabilistic way and then applies the repair method to generate a feasible solution.

- 1: **input** an instance of the 0-1 knapsack problem;
- 2: initialize population Φ_0 consisting of N feasible solutions;
- 3: **for** $t = 0, 1, \dots, t_{\max}$ **do**
- 4: generate a random number r in $[0, 1]$;
- 5: **if** $r < 0.9$ **then**
- 6: children population $\Phi_{t,c} \leftarrow$ bitwise-mutate Φ_t ;
- 7: **else**
- 8: children population $\Phi_{t,c} \leftarrow$ one-point crossover Φ_t ;
- 9: **end if**
- 10: **if** a child is an infeasible solution **then**
- 11: choose one method from the ratio-greedy repair, random repair and value-greedy repair with probability $1/3$ and repair the child into feasible;
- 12: **end if**
- 13: the best individual in the parent and children populations is selected into population Φ_{t+1} ;
- 14: $N - 1$ individuals from the parent and children populations into population Φ_{t+1} by roulette wheel selection;
- 15: **end for**
- 16: **output** the maximum of the fitness function.

The genetic operators used in the above GA are explained below.

- *Bitwise Mutation*: Given a binary vector $(x_1 \dots x_n)$, flip each bit x_i with probability $1/n$.

- *One-Point Crossover*: Given two binary vectors $(x_1 \cdots x_n)$ and $(y_1 \cdots y_n)$, randomly choose a crossover point $k \in \{1, \dots, n\}$, swap their bits at point k . Then generate two new binary vectors as follows,

$$(x_1 \cdots x_k y_{k+1} \cdots y_n), \quad (y_1 \cdots y_k x_{k+1} \cdots x_n).$$

Like most of GAs, the MSGA may find reasonably good solutions but has no guarantee about the solution quality. Thus it is necessary to design evolutionary approximation algorithms with provably good solution quality. The most straightforward approach is that we first apply the greedy algorithm for generating approximation solutions and then take these solutions as the starting point of the MSGA. We call this approach *greedy algorithm + MSGA*. Since the MSGA starts at local optima, it becomes hard for the MSGA to leave the absorbing basin of this local optimum for seeking the global optimum. This is the main drawback of the approach.

III. GENETIC ALGORITHM USING HELPER OBJECTIVES FOR THE 0-1 KNAPSACK PROBLEMS

In this section, we propose a novel multi-objective optimisation GA (MOGA) which can beat the combination of greedy algorithm + MOGA mentioned in the previous section. The algorithm is based on the multi-objectivization technique. The original single objective optimization problem (1) is recast into a multi-objective optimization problem using helper objectives. The design of helper objectives depends on problem-specific knowledge. The first helper objective comes from an observation on the following instance.

Item	1	2	3	4	5
Profit	10	10	10	12	12
Weight	10	10	10	10	10
Capacity	20				

The global optimum in this instance is 00011. In the optimal solution, the average profit of packed items is the largest. Thus the first helper objective is to maximize the average profit of items in a knapsack. The objective function is

$$h_1(\vec{x}) = \frac{1}{\|\vec{x}\|_1} \sum_{i=1}^n x_i p_i. \quad (2)$$

where $\|\vec{x}\|_1 = \sum_{i=1}^n x_i$.

The second objective is inspired from an observation on another instance.

Item	1	2	3	4	5
Profit	15	15	20	20	20
Weight	10	10	20	20	20
Capacity	20				

The global optimum in this instance is 11000. In the optimal solution, the average profit-to-weight ratio of packed items is the largest. However, the average profit of these items is not the largest. Then the second helper objective is to maximize the average profit-to-weight ratio of items in a knapsack. The objective function is

$$h_2(\vec{x}) = \frac{1}{\|\vec{x}\|_1} \sum_{i=1}^n x_i \frac{p_i}{w_i}. \quad (3)$$

Finally let's look at the following instance.

Item	1	2	3	4	5
Profit	40	40	40	40	150
Weight	30	30	30	30	100
Capacity	120				

It is not difficult to verify that the global optimum in this instance is 11110. In the optimal solution, neither the average profit of packed items nor average profit-to-weight ratio is the largest, but the number of packed items is the largest. Thus the third helper objective is to maximize the number of items in a knapsack. The objective function is

$$h_3(\vec{x}) = \|\vec{x}\|_1. \quad (4)$$

We then come to the following multi-objective optimization problem:

$$\max_{\vec{x}} \{f(\vec{x}), h_1(\vec{x}), h_2(\vec{x}), h_3(\vec{x})\}, \quad \text{subject to } \sum_{i=1}^n w_i x_i \leq C. \quad (5)$$

Besides the above three helper objectives, it is possible to add more helper objectives, for example, to minimise the average weight of packed items.

The multi-objective optimisation problem (5) is solved by a MOGA using bitwise mutation, one-point crossover and multi-criteria selection, plus a mixed strategy of three repair methods.

```

1: input an instance of the 0-1 knapsack problem;
2: initialize  $\Phi_0$  consisting of  $N$  feasible solutions;
3: for  $t = 0, 1, \dots, t_{\max}$  do
4:   generate a random number  $r$  in  $[0, 1]$ ;
5:   if  $r < 0.9$  then
6:     children population  $\Phi_{t,c} \leftarrow \text{bitwise mutate } \Phi_t$ ;
7:   else
8:     children population  $\Phi_{t,c} \leftarrow \text{one-point crossover } \Phi_t$ ;
9:   end if
10:  if any child is an infeasible solution then
11:    choose one repair method from the ratio-greedy repair, random repair and value-greedy repair with probability 1/3;
12:    repair the child into a feasible solution;
13:  end if
14:  population  $\Phi_{t+1} \leftarrow \text{multi-criterion select } N \text{ individuals from } \Phi_t \text{ and } \Phi_{t,c}$ ;
15: end for
16: output the maximum of  $f(\vec{x})$  in the final population.

```

The *multi-criteria selection* operator, adopted in the above MOGA, is novel and inspired from multi-objective optimisation. Since the target is to maximise several objectives simultaneously, we select individuals which have higher function values with respect to each objective function. The pseudo-code of multi-criteria selection is described as follows.

```

1: input the parent population  $\Phi_t$  and child population  $\Phi_{t,c}$ ;
2: merge the parent and children populations into a temporary population which consists of  $2N$  individuals;
3: sort all individuals in the temporary populations in the descending order of  $f(\vec{x})$ , denote them by  $\vec{x}_1^{(1)}, \dots, \vec{x}_{2N}^{(1)}$ ;
4: select all individuals from left to right (denote them by  $\vec{x}_{k_1}^{(1)}, \dots, \vec{x}_{k_m}^{(1)}$ ) which satisfy  $h_1(\vec{x}_{k_i}^{(1)}) < h_1(\vec{x}_{k_{i+1}}^{(1)})$  or  $h_2(\vec{x}_{k_i}^{(1)}) < h_2(\vec{x}_{k_{i+1}}^{(1)})$  for any  $k_i$ .
5: if the number of selected individuals is greater than  $m \frac{N}{3}$  then
6:   truncate them to  $\frac{N}{3}$  individuals;
7: end if
8: add these selected individuals into the next population  $\Phi_{t+1}$ ;
9: resort all individuals in the temporary population in the descending order of  $h_1(\vec{x})$ , still denote them by  $\vec{x}_1, \dots, \vec{x}_{2N}$ ;
10: select all individuals from left to right (still denote them by  $\vec{x}_{k_1}, \dots, \vec{x}_{k_m}$ ) which satisfy  $h_3(\vec{x}_{k_i}) < h_3(\vec{x}_{k_{i+1}})$  for any  $k_i$ .
11: if the number of selected individuals is greater than  $\frac{N}{3}$  then
12:   truncate them to  $\frac{N}{3}$  individuals;
13: end if
14: add these selected individuals into the next population  $\Phi_{t+1}$ ;
15: resort all individuals in the temporary populations in the descending order of  $h_2(\vec{x})$ , still denote them by  $\vec{x}_1, \dots, \vec{x}_{2N}$ ;
16: select all individuals from left to right (still denote them by  $\vec{x}_{k_1}, \dots, \vec{x}_{k_m}$ ) which satisfy  $h_3(\vec{x}_{k_i}) < h_3(\vec{x}_{k_{i+1}})$  for any  $k_i$ .
17: if the number of selected individuals is greater than  $\frac{N}{3}$  then
18:   truncate them to  $\frac{N}{3}$  individuals;
19: end if
20: add these selected individuals into the next population  $\Phi_{t+1}$ ;
21: while the population size of  $\Phi_{t+1}$  is less than  $N$  do
22:   randomly choose an individual from the parent population and add it into  $\Phi_{t+1}$ ;
23: end while
24: output a new population  $\Phi_{t+1}$ .

```

In the above algorithm, Steps 3-4 are for selecting the individuals with higher values of $f(\vec{x})$. In order to preserve diversity, we choose these individuals which have different values of $h_1(\vec{x})$ or $h_2(\vec{x})$. Similarly Steps 9-10 are for selecting the individuals with a higher value of $h_1(\vec{x})$. We choose the individuals which have different values of $h_3(\vec{x})$ for maintaining diversity. Steps 15-16 are for selecting individuals with a higher value of $h_2(\vec{x})$. Again we choose these individuals which have different values of $h_3(\vec{x})$ for preserving diversity. We don't explicitly select individuals based on $h_3(\vec{x})$. Instead we implicitly do it during Steps 9-10, and Steps 15-16.

Steps 5-7, Steps 11-13, Steps 17-19, plus Steps 21-23 are used to maintain an invariant population size N .

The benefit of using multi-criterion selection is its ability of making search along different directions $f(\vec{x})$, $h_1(\vec{x})$, $h_2(\vec{x})$ and implicitly $h_3(\vec{x})$. Hence the MOGA may not get trapped into the absorbing area of a local optimum.

IV. EXPERIMENTS

In this section, we implement computer experiments. According to [1], [4], the instances of the 0-1 knapsack problem are often classified into two categories.

- 1) *Restrictive capacity knapsack*: the knapsack capacity is so small that only a few items can be packed in the knapsack. An instance with restrictive capacity knapsack is generated in the following way. Choose a parameter B which is an upper bound on the weight of each item. In the experiments, set $B = n$. For item i , its profit p_i and weight w_i are generated at uniformly random in $[1, B]$. Set the capacity of the knapsack $C = B$.
- 2) *Average capacity knapsack*: the knapsack capacity is so large that it is possible to pack half of items into the knapsack. An instance with average capacity knapsack is generated as follows. Choose a parameter B which is the upper bound on the weight of each item. In the experiments, set $B = n$. For item i , its profit p_i and weight w_i are generated at uniformly random in $[1, B]$. Since the average weight of each item is $0.5B$, thus the average of the total weight of items is $0.5nB$. So we set the capacity to be the half of the total weight, that is $C = 0.25nB$.

For each type of the 0-1 knapsack problem, 10 instances are generated at random. For each instance, the number of items n is 100. The population size is $3n$. The number of maximum generations is $30n$ for the MSGA and $10n$ for the MOGA. All individuals in the initial population are generated at random. If an individual is an infeasible solution, it is repaired to feasible using random repair.

Besides the above randomly generated instances, we also consider two special instances. Special instance I is given Table I.

TABLE I
SPECIAL INSTANCE I: $n = 500$ AND $\alpha = 0.2$

Item i	$1, \dots, \lceil \frac{n}{1+\alpha} \rceil$	$\lceil \frac{n}{1+\alpha} \rceil + 1$	$\lceil \frac{n}{1+\alpha} \rceil + 2, \dots, n$
Profit p_i	1	$\frac{\alpha n}{1+\alpha}$	$\frac{1}{n}$
Weight w_i	1	$\frac{n}{1+\alpha} - \frac{\alpha}{4+4\alpha}$	$\frac{1}{2n}$
Capacity	$\frac{n}{1+\alpha}$		
Initialisation	0	1	half bits are 1

Special instance II is given in Table II.

TABLE II
SPECIAL INSTANCE II: $n = 200$

Item i	$1, \dots, \frac{n}{4}$	$\frac{n}{4} + 1, \dots, \frac{n}{2}$	$\frac{n}{2} + 1, \dots, n$
Profit p_i	$0.25n\sqrt{n} + 2$	$0.3n\sqrt{n}$	\sqrt{n}
Weight w_i	$0.25n\sqrt{n} + 1$	$0.5n\sqrt{n}$	\sqrt{n}
Capacity	$0.5n\sqrt{n}$		
Initialisation	one bit is 1, others 0	0	half bits are 1

The population size is n for Instances I and II. The number of maximum generations is $15n$ for the MSGA and $5n$ for the MOGA. The initialisation of individuals in both MSGA and MOGA refer to the above tables.

Tables III gives experiment results of comparing the greedy algorithm, MSGA, greedy algorithm + MSGA and MOGA. From the table, we observe that

- the solution quality of MSGA is better or not worse than the greedy algorithm in 20 random instances. However for Instance I, the MSGA only finds a solution whose value is about 20% of the optimal value.
- the solution quality of greedy algorithm + MSGA is better or not worse than the greedy algorithm in all instances. However for Instance II, the algorithm gets trapped into a local optimum, and is worse than the MOGA.
- the MOGA is the winner among 4 algorithms and its the solution quality is better or not worse than the greedy algorithm and MSGA in all instances.

V. CONCLUSIONS

A novel MOGA using helper objectives is proposed in this paper for solving the 0-1 knapsack problem. First the original 0-1 knapsack problem is recast into a multi-objective optimization problem (i.e. to maximize the sum of profits packed in the

TABLE III

A COMPARISON AMONG 4 ALGORITHMS IN 20 RANDOMLY GENERATED INSTANCES AND 2 SPECIAL INSTANCES. THE FIRST 10 INSTANCES BELONG TO THE RESTRICTIVE CAPACITY KNAPSACK PROBLEM. THE SECOND 10 INSTANCES BELONG TO THE AVERAGE CAPACITY KNAPSACK PROBLEM. ‘MAX’: THE MAXIMUM VALUE OF $f(\vec{x})$ PRODUCED DURING 10 RUNS. ‘AVERAGE’: THE AVERAGE VALUE OF $f(\vec{x})$ OVER 10 RUNS. ‘STDEV’: THE STANDARD DERIVATION OF $f(\vec{x})$ IN 10 RUNS.

	Greedy	MSGA			Greedy + MSGA			MOGA		
Instance		max	average	stdev	max	average	stdev	max	average	stdev
1	674	683	683	0	683	683	0	683	681.2	1.55
2	714	714	714	0	714	714	0	714	714	0
3	561	622	622	0	622	622	0	622	622	0
4	631	631	631	0	631	631	0	631	631	0
5	585	621	621	0	621	620.7	0.95	621	620.7	0.95
6	787	787	787	0	787	787	0	787	787	0
7	736	773	773	0	773	773	0	773	773	0
8	1042	1076	1076	0	1076	1076	0	1076	1076	0
9	982	994	994	0	994	993	3.16	994	993	3.16
10	906	942	942	0	942	942	0	942	942	0
11	4107	4111	4110.9	0.32	4111	4111	0	4111	4111	0
12	4090	4102	4100.6	1.43	4102	4097.1	4.41	4102	4101.2	1.03
13	4138	4169	4168.6	1.26	4169	4165.6	2.37	4169	4169	0
14	3901	3925	3923.9	1.29	3925	3922.2	2.15	3927	3923.7	1.95
15	3997	4047	4047	0	4047	4043.2	3.71	4047	4047	0
16	3984	3994	3993	1.05	3994	3992.5	0.85	3994	3993.8	0.632
17	3820	3848	3848	0	3848	3845.1	1.97	3848	3848	0
18	3914	3920	3919.4	1.90	3920	3915.2	2.53	3920	3920	0
19	4456	4471	4470.1	0.57	4470	4465.4	2.59	4471	4470.4	0.97
20	4149	4177	4175.3	1.34	4177	4171.9	2.73	4177	4176	0.82
I	416.2	83.4	83.4	0	416.2	416.2	0	416.2	416.2	0
II	1402.1	1402.1	1402.1	0	1402.1	1402.1	0	1414.2	1414.2	0

knapsack, to maximize the average profit-to-weight ratio of items, to maximize the average profit of items, and to maximize the number of packed items). Then a MOGA (using bitwise mutation, one-point crossover and multi-criterion selection plus a mixed strategy of three repair methods) is designed for the multi-objective optimization problem. Experiment results demonstrate that the MOGA using helper objectives outperforms its rivals, which are the greedy algorithm, MSGA and greedy algorithm + MSGA. The results also show that the MSGA can find reasonably good solutions but without a guarantee; and the greedy algorithm + MSGA sometimes gets trapped into a local optimum.

REFERENCES

- [1] S. Martello and P. Toth, *Knapsack Problems*. Chichester: John Wiley & Sons, 1990.
- [2] Z. Michalewicz and J. Arabas, “Genetic algorithms for the 0/1 knapsack problem,” in *Methodologies for Intelligent Systems*. Springer, 1994, pp. 134–143.
- [3] S. Khuri, T. Bäck, and J. Heitkötter, “The zero/one multiple knapsack problem and genetic algorithms,” in *Proceedings of the 1994 ACM Symposium on Applied Computing*. ACM, 1994, pp. 188–193.
- [4] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed. New York: Springer Verlag, 1996.
- [5] E. Zitzler and L. Thiele, “Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach,” *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.
- [6] A. Jaszkiewicz, “On the performance of multiple-objective genetic local search on the 0/1 knapsack problem—a comparative experiment,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 4, pp. 402–412, 2002.
- [7] M. Eugénia Captivo, J. Climaco, J. Figueira, E. Martins, and J. Luis Santos, “Solving bicriteria 0–1 knapsack problems using a labeling algorithm,” *Computers & Operations Research*, vol. 30, no. 12, pp. 1865–1886, 2003.
- [8] J. D. Knowles, R. A. Watson, and D. W. Corne, “Reducing local optima in single-objective problems by multi-objectivization,” in *Evolutionary Multi-Criterion Optimization*. Springer, 2001, pp. 269–283.
- [9] M. T. Jensen, “Helper-objectives: Using multi-objective evolutionary algorithms for single-objective optimisation,” *Journal of Mathematical Modelling and Algorithms*, vol. 3, no. 4, pp. 323–347, 2005.
- [10] J. Handl, S. C. Lovell, and J. Knowles, “Multiobjectivization by decomposition of scalar cost functions,” in *Parallel Problem Solving from Nature—PPSN X*. Springer, 2008, pp. 31–40.
- [11] D. Brockhoff, T. Friedrich, N. Hebbinghaus, C. Klein, F. Neumann, and E. Zitzler, “On the effects of adding objectives to plateau functions,” *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 3, pp. 591–603, 2009.
- [12] D. F. Lochtefeld and F. W. Ciarallo, “Helper-objective optimization strategies for the job-shop scheduling problem,” *Applied Soft Computing*, vol. 11, no. 6, pp. 4161–4174, 2011.
- [13] R. Kumar and N. Banerjee, “Analysis of a multiobjective evolutionary algorithm on the 0–1 knapsack problem,” *Theoretical Computer Science*, vol. 358, no. 1, pp. 104–120, 2006.
- [14] T. Friedrich, J. He, N. Hebbinghaus, F. Neumann, and C. Witt, “Approximating covering problems by randomized search heuristics using multi-objective models,” *Evolutionary Computation*, vol. 18, no. 4, pp. 617–633, 2010.
- [15] X. Lai, Y. Zhou, J. He, and J. Zhang, “Performance analysis of evolutionary algorithms for the minimum label spanning tree problem,” *IEEE Transactions on Evolutionary Computation*, 2014, (accepted, online).
- [16] J. He and Y. Zhou, “A comparison of GAs using penalizing infeasible solutions and repairing infeasible solutions II,” in *Proceedings of the 2nd International Symposium on Intelligence Computation and Applications*. Wuhan, China: Springer, 2007, C1, pp. 102–110.