A Comparison of GAs Penalizing Infeasible Solutions and Repairing Infeasible Solutions on the 0-1 Knapsack Problem ⋆

Jun He[1] ⋆⋆, Yuren Zhou[2]⋆⋆⋆, and Xin Yao[3]⋆⋆

[1] J. He is with the Department of Computer Science, University of Wales Aberystwyth, SY23 3DB, UK (email: j.he@cs.bham.ac.uk)
[2] Y. Zhou is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510640, China (email: yrzhou@scut.edu.cn).
[3] X. Yao is with the School of Computer Science, University of Birmingham, Birmingham, B15 2TT, UK (email: x.yao@cs.bham.ac.uk).

Abstract. Constraints exist in almost every optimization problem. Different constraint handling techniques have been incorporated with genetic algorithms (GAs), however most of current studies are based on computer experiments. An example is Michalewicz's comparison among GAs using different constraint handling techniques on the 0-1 knapsack problem. The following phenomena are observed in experiments: 1) the penalty method needs more generations to find a feasible solution to the restrictive capacity knapsack than the repair method; 2) the penalty method can find better solutions to the average capacity knapsack. Such observations need a theoretical explanation. This paper aims at providing a theoretical analysis of Michalewicz's experiments. The main result of the paper is that GAs using the repair method are more efficient than GAs using the penalty method on both restrictive capacity and average capacity knapsack problems. This result of the average capacity is a little different from Michalewicz's experimental results. So a supplemental experiment is implemented to support the theoretical claim. The results confirm the general principle pointed out by Coello: a better constraint-handling approach should tend to exploit specific domain knowledge.

1 Introduction

There is a wide gap between the theory and practice of genetic algorithms. Current theories analyze simple algorithms and simple problems, rather than algorithms and problems used in real applications. In the real world, almost every optimization problem contains more or less constraints. It is very important

⋆ Manuscript submitted on May 1, 2008.
⋆⋆ The work of J. He and X. Yao was supported by the UK Engineering and Physical Research Council under Grant No. EP/C520696/1.
⋆⋆⋆ The work of Y. Zhou was supported in part by the National Natural Science Foundation of China under Grant No.60673062, in part by the Natural Science Foundation of Guangdong Province of China under Grant No.06025686.

Dagstuhl Seminar Proceedings 08051
Theory of Evolutionary Algorithms
http://drops.dagstuhl.de/opus/volltexte/2008/1482

to handle constraints when GAs are used in solving optimization problems in practice. Different constraint handling techniques have been incorporated with GAs, and many papers have contributed to this direction, e.g. [1–6]. Due to its importance in practice, how to deal with constraints is still a hot research in the field of evolutionary computation. Many new constraint handling techniques have been designed in recent years, for example, stochastic ranking [7], co-evolutionary augmented Lagrangian methods [8], multiple Lagrange multiplier method [9], two phase algorithm, [10], α-constrained method [11], multi-objective approach [12], simple multi-membered evolution strategy [13] etc. However, most current studies are based on computer experiments. Few theoretical contributions have been made towards this research issue.

Through computer experiments, Michalewicz has compared GAs using different constraint handling techniques [14], including the penalty function method, repair method and decode method. The following phenomena were observed in the experiments [14].

1. For the average capacity knapsack problem, GAs using the penalty function method cannot get a feasible solution in 500 generations. The GA using the repair method $A_r[2]$ is better than others.
2. For the average capacity knapsack problem, the penalty function algorithm $A_p[1]$ is the best among five GAs.

However, the above observations are based on computer experiments, and no theoretical analysis has been made. It is necessary to make a theoretical analysis to explain these experimental results. This paper aims at providing such an analysis. Of course, it will be impossible to investigate all constraint-handling techniques in a single paper, so the discussion is restricted to two classical and popular methods introduced in [14]: the methods of penalizing infeasible solutions and repairing infeasible solutions.

The time complexity has been widely used to measure the theoretical performance of genetic algorithms, i.e. how many generations are needed to find a solution [15–19]. There were only a few theoretical analysis of running times of GAs for solving the knapsack problem. In [20], the constraint is handled by a multi-objective optimization technique. Two multi-objective evolutionary algorithms, SEMO and FEMO, are applied for a simple instance of the multi-objective 0/1 knapsack problem and the expected running time is analyzed. In [21], the problem with constraints is also transformed into a two-objective optimization problem. A multi-objective evolutionary algorithm, REMO, is proposed to solve the knapsack problem. The paper formalizes a $(1+\epsilon)$-approximate set of the knapsack problem and presents a rigorous running time analysis to obtain the formalized set. The running time on a special bi-objective linear function is analyzed. The paper [22] has analyzed the role of penalty coefficients in GAs. It is shown that in some examples, GAs have benefited greatly from higher penalty coefficients, while in other examples, they benefited from lower penalty coefficients. The runtime of GAs for solving the knapsack problem ranges from a polynomial time to an exponential time when different penalty coefficients

are used. Differently from previous research, this paper continues the previous initial study [22] and focuses on a comparison between two constraint-handling techniques and gives a theoretical analysis to the experimental results reported in [14].

The paper is organized as follows: Section 2 introduces the knapsack problem and describes GAs for solving the 0-1 knapsack problem. Section 3 analyzes the restrictive capacity knapsack problem. Section 4 estimates first hitting times of GAs for restrictive capacity problems. Section 5 analyzes the average capacity knapsack problem. Section 6 compares GAs for average capacity problems. Section 7 reports two supplemental experimental results. Section 8 gives conclusions.

2 Knapsack Problem and Genetic Algorithms

2.1 Knapsack Problem

The 0-1 knapsack problem is to

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} p_i x_i, \\ \text{subject to} \quad & \sum_{i=1}^{n} w_i x_i \leq C, \\ & x_i = 0 \text{ or } 1, \quad i = 1, \cdots, n, \end{aligned} \tag{1}$$

where

$$x_i = \begin{cases} 1 \text{ if item } i \text{ is selected,} \\ 0 \text{ else,} \end{cases}$$

and p_i is the profit of item i, w_i the weight of item i and C the capacity of the knapsack.

A detailed discussion about the knapsack problem can be referred to [23]. GAs have been applied for solving the knapsack problem [14, 24]. Since this problem is an NP-complete problem, it is hard for GAs to solve.

The difficulty of the problem is greatly affected by the correlation between profits and weights. Three randomly generated data sets are considered in experiments [23, 14]:

1. *uncorrelated*: the p_i and w_i are chosen uniformly at random in $[1, \nu]$;
2. *weakly correlated*: the w_i are chosen uniformly at random in $[1, \nu]$ and the p_i uniformly at random in $[w_i - r, w_i + r]$ (if for some i $p_i \leq 0$, then the random generation procedure should be repeated until $p_i > 0$);
3. *strongly correlated*: the w_i are chosen uniformly at random in $[1, \nu]$; and $p_i = w_i + r$;

In this paper, the parameters ν and r are set to be

$$\nu = \frac{n}{20}, \quad r = \frac{n}{20}.$$

Two values of the capacity are considered in [23, 14]:

1. *restrictive capacity knapsack:* the capacity of the knapsack is small, i.e.
$$C_1 = 2\nu.$$
2. *average capacity knapsack:* the capacity of the knapsack is large,
$$C_2 = 0.5 \sum_{i=1}^{n} w_i.$$

Due to the paper length limitations, this paper only discusses the uncorrelated capacity knapsack and leaves other problems to future work.

2.2 Genetic Algorithms

The GA [14] is described as in Figure 1. The encoding is a binary representation $(x_1 \cdots x_n)$, where $x_i = 1$ if the i-th item is chosen in the knapsack, $x_i = 0$ if the i-th item is not in the knapsack.

begin
  $t := 0$;
  initialize $\xi^{(0)}$;
  evaluate $\xi^{(0)}$;
  while (termination-condition does not hold) **do**
    mutate $\xi^{(t)}$;
    evaluate $\xi^{(t)}$;
    $t := t + 1$;
    select $\xi^{(t)}$ from $\xi^{(t-1)}$;
  od
end

Fig. 1. Genetic Algorithm. t is the generation counter; $\xi^{(t)}$ is the t-th generation population.

The genetic operators used in the GA are bitwise mutation with a mutation rate $1/n$ and an elitist selection. Crossover is not considered in the GA since the analysis of a crossover is too complex.

The fitness of an individual is dependent on the technique used for handling constraints. A general introduction of constraint-handling techniques incorporated with GAs can be found in many references, e.g. [1, 14, 5].

In this paper, two of them are considered: the methods of penalizing infeasible solutions and repairing infeasible solutions.

For the method using a penalty function, the fitness of an individual consists of two parts:
$$f(x) = \sum_{i=1}^{n} x_i p_i - g(x),$$

where the penalty term $g(x)$ is set to 0 for all feasible solutions, otherwise it is assigned a positive value.

There are many different ways to assign the penalty value. Michalewicz [14] used three types of penalty functions in experiments.

1. $A_p[1]$: the penalty function is

$$g_1(x) = \ln(1 + \rho(\sum_{i=1}^{n} x_i w_i - C)) \tag{2}$$

2. $A_p[2]$: the penalty function is

$$g_2(x) = \rho(\sum_{i=1}^{n} x_i w_i - C) \tag{3}$$

3. $A_p[2]$: the penalty function is

$$g_3(x) = \rho^2(\sum_{i=1}^{n} x_i w_i - C))^2 \tag{4}$$

where

$$\rho := \max_{i=1,\cdots,n} \{\frac{p_i}{w_i}\}.$$

GAs using the repair method are almost the same as GAs using the penalty method. The only difference is at one point: if an infeasible individual is generated, then it will be repaired and then become a feasible solution.

The fitness function is determined as follows [14]:

$$f(x) = \sum_{i=1}^{n} x'_i p_i,$$

where x' is a feasible individual obtained by repairing the infeasible individual x.

The repairing procedure [14] is described in Figure 2.

Two repairing methods are used in [14] which are described as follows:

1. $A_r[1]$: the algorithm uses a random repairing approach, i.e. the **select** procedure is to choose an item from the knapsack randomly.
2. $A_r[2]$: the algorithm uses a greedy repairing approach, i.e first the items are sorted according to the order of p_i/w_i, then the **select** procedure is to choose the smallest item.

3 Restrictive Capacity Knapsack

3.1 Initial Solutions and First Hitting Times

Let us consider a GA using a penalty function for the restrictive capacity knapsack problem.

```
begin
  knapsack-overfilled:=false;
  x' := x;
  if (∑_{i=1}^{n} x_i w_i > C)
    then knapsack-overfilled:=true;
         while (knapsack-overfilled) do
                i =: select an item from the knapsack;
                set x_i = 0;
                if (∑_{i=1}^{n} x'_i w_i ≤ C)
                  then knapsack-overfilled:=false;
                fi
         od
  fi
end
```

Fig. 2. Repairing Procedure [14].

Lemma 1. *If an initial individual is chosen at random, then the probability for the individual of being an infeasible solution is not less than* $1 - e^{-n/16}$.

Proof: Since w_i is chosen at random in $[1, \cdots, \nu]$, then $w_i \geq 1$ and its expectation is

$$\mathrm{E}[w_i] = \nu/2.$$

Since for each bit in a binary string, the probability of $x_i = 1$ is 0.5, then its expectation is

$$\mathrm{E}[\sum_{i=1}^{n} x_i] = \frac{n}{2}.$$

Each bit is chosen independently at random, hence from Chernoff bounds, it is known that

$$\mathrm{P}\left(\sum_{i=1}^{n} x_i > \frac{n}{4}\right) < e^{-n/16}.$$

This means with no less than a probability $1 - e^{-n/16}$,

$$\sum_{i=1}^{n} w_i x_i \geq \sum_{i=1}^{n} x_i > \frac{n}{4}.$$

Since the capacity is set to be

$$C_1 = \frac{n}{10},$$

with a probability of $1 - e^{-n/16}$, an initial individual satisfies:

$$\sum_{i=1}^{n} w_i x_i > C_1,$$

i.e. the individual is an infeasible solution. ■

Corollary 1. *Given a population with N individuals, the probability of all individuals in the initial population of being infeasible is not less than*

$$1-(1-e^{-n/16})^N.$$

This probability is still very small if $N = O(n)$.

Based on the above lemma and corollary, is is reasonable to assume that any individual in the initial population is infeasible and

$$\sum_{i=1}^{n} w_i x_i > \frac{n}{4}.$$

Denote S to be the set of all individuals. Let the feasible solution set be

$$S_f := \{x;\ \sum_{i=1}^{n} w_i x_i \leq C_1\},$$

and the infeasible solution set be

$$S_i := \{x;\ \sum_{i=1}^{n} w_i x_i > C_1\}.$$

Furthermore, for the population space S^N, denote the feasible population set as

$$S_f^N := \{X;\ \exists x \in X : x \in S_f\},$$

and the infeasible population set as

$$S_i^N := \{X;\ \forall x \in X : x \in S_i\}.$$

If an individual is feasible, then it satisfies the condition

$$\sum_{i=1}^{n} x_i w_i \leq C_1 = 2\nu.$$

Since $w_i \geq 1$, so $\forall x \in S_f$, it holds that:

$$\sum_{i=1}^{n} x_i \leq \frac{n}{10}.$$

The population sequence $\{\xi^{(t)}, t = 0, 1, 2, \cdots\}$ usually is a Markov chain [25, 26]. The first hitting time is a useful concept to measure the computation time of GAs [27, 28]. Several first hitting times are defined in the following.

Let τ_f be the mean number of generations for a GA to find a feasible solution. The following number is a first hitting time:

$$\tau_f := \{t \geq 0;\ \xi^{(t)} \in S_f^N.\}$$

Let τ_n the number of generations for the GA to find a near-feasible solution to be

$$\tau_n := \{t \geq 0;\ H(\xi^{(t)}, S_f) \leq 1\}.$$

where $H(x, y)$ is the Hamming distance between x and y, and

$$H(A, B) := \min\{H(x, y);\ x \in A, y \in B\}.$$

Furthermore, let τ, the first hitting time to optimal solutions, be

$$\tau := \{t \geq 0;\ f(\xi^{(t)}) = f_{\max}\}.$$

where

$$f_{\max} := \max_{x \in S}\{f(x) \mid \sum_{i=1}^{n} w_i x_i \leq C_1\}.$$

3.2 Types of Instances

Given an individual x, define its neighbour with distance δ as follows:

$$N(x, \delta) := \{y;\ H(x, y) \leq \delta\}.$$

Define its left neighbour with distance δ as below:

$$N_l(x, \delta) := \{y;\ H(x, y) \leq \delta, \mid y \mid \leq \mid x \mid\}$$

where $\mid x \mid := \sum_{i=1}^{n} x_i$.

For convenience in the analysis, the instances of the knapsack problem are grouped into several types. The idea originates from the classification of narrow-gap and wide-gap fitness landscapes appearing in [27, 19], but with a small modification: the distance function here is the Hamming distance.

- **T1: non-equivalent narrow-gap problem.** A fitness function $f(x)$ satisfies that $\exists y \in S_i$,

$$f(y) > f_{\max}. \tag{5}$$

  This means that the problem of maximizing $f(x)$ is not equivalent to the original knapsack problem. So it is called a non-equivalent problem.
  The fitness function $f(x)$ belongs to a narrow-gap landscape in Hamming distance, defined by: $\exists \delta >$ where $\delta = O(1)$ and $\forall x \in S_i$:
  - $\exists y \in N_l(x, \delta)$: $f(y) > f(x)$.
  - or $\exists y \in N_l(x, \delta)$: $y \in S_f$.
- **T2: non-equivalent wide-gap problem.** $\exists y \in S_i$,

$$f(y) > f_{\max}. \tag{6}$$

  The fitness function $f(x)$ belongs to a wide-gap landscape in Hamming distance, defined by some $\delta = \omega(1)$
  - $\exists x \in S_i$, $\forall y \in N_l(x, \delta) : y \in S_i$ and $f(x) > f(y)$.

A special case is that

- **T2.a:** $\delta = \epsilon n$, where $0 < \epsilon < 1$ is a constant.

– **T3: non-equivalent wide-plateau problem:** $\exists y \in S_i$,

$$f(y) > f_{\max}. \tag{7}$$

The fitness function $f(x)$ belongs to a wide plateau landscape in Hamming distance, defined by: $\exists x \in S_i$, $\forall y \in N_l(x, \delta)$ where $\delta = \omega(1)$, y satisfies that

- $y \in S_i$.
- $f(x) \geq f(y)$.
- $\exists c > 0$ where $c > 0$ is a constant and $\forall y \in N_l(x, \delta)$, $\exists z \in N_l(y, c)$: $f(z) = f(x)$.

A special case is that

- **T3.a:** $\delta = \epsilon n$, where $0 < \epsilon < 1$ is a constant.

– **T4: Equivalent narrow-gap problem.** For any infeasible solution y, whose fitness satisfies

$$f(y) < f_{\max}.$$

The problem of maximizing $f(x)$ is equivalent to the original knapsack problem. So it is called an equivalent problem.
The fitness function $f(x)$ belongs to a narrow-gap landscape in Hamming distance as defined in Type 1. A special case is

- **T4.a:** For any infeasible solution x, flip one of $\mid x \mid -1$ one-valued bits, the new individual y satisfies $f(y) > f(x)$.

– **T5: Wide-gap equivalent problem.** For any infeasible solution y, whose fitness satisfies:

$$f(y) < f_{\max}.$$

The fitness function $f(x)$ belongs to a wide-gap problem in Hamming distance as defined in Type 2. A special case is that

- **T5.a:** there exists some infeasible solution x, and the distance of its neighbour $N_l(y, \delta)$ satisfies $\delta = \epsilon n$, where $0 < \epsilon < 1$ is a constant.

– **T6: Wide-plateau equivalent problem.** For all infeasible solutions y, whose fitness satisfies

$$f(y) < f_{\max}.$$

The fitness function $f(x)$ belongs to a wide-plateau gap problem in Hamming distance as defined in Type 3. A special case is that

- **T6.a:** there exists some infeasible solution x, and the distance of its neighbour $N_l(y, \delta)$ is as large as $\delta = \epsilon n$, where ϵ is a constant.

Proposition 1. *The fitness function derived from Algorithm $A_p[1]$ belongs to Type 2.*

Proof: For any infeasible solution x, its penalty is

$$g_1(x) = \ln(1 + \rho(\sum_{i=1}^{n} w_i x_i - C_1)),$$

and its fitness is

$$f(x) = \sum_{i=1}^{n} p_i x_i - \ln(1 + \rho(\sum_{i=1}^{n} w_i x_i - C_1)).$$

Let $\mathbf{1} = (1 \cdots 1)$ which is infeasible, and its fitness is

$$f(x) = \sum_{i=1}^{n} p_i - \ln(1 + \rho(\sum_{i=1}^{n} w_i - C_1)).$$

Assume $x^* = (x_1^* \cdots x_n^*)$ is the optimal solution. Then denote $I(x^*)$ to be the set of all items in the knapsack, i.e. the subscripts i with $x_i^* = 1$.

Since x^* is feasible, we have

$$\sum_{i \in I(x^*)} x_1^* w_i \leq C_1,$$

Since $w_i \in [1, \cdots, \nu]$, we know that the cardinality of $I(x^*)$ satisfies

$$\mid I(x^*) \mid \leq C_1 = \frac{n}{10}.$$

Notice that

$$f(\mathbf{1}) - f(x^*) = \sum_{i=1}^{n} p_i - \sum_{i \in I(x^*)} p_i - \ln(1 + \rho(\sum_{i=1}^{n} w_i - C_1))$$

$$= \sum_{i \notin I(x^*)} p_i - \ln(1 + \rho(\sum_{i=1}^{n} w_i - C_1))$$

Now we want to prove that

$$f(\mathbf{1}) > f(x^*).$$

Equivalently,

$$\exp(\sum_{i \notin I(x^*)} p_i) \geq 1 + \rho(\sum_{i=1}^{n} w_i - C_1)$$

Since $p_i \in [1, \cdots, \nu]$ and $w_i \in [1, \cdots, \nu]$, we have

$$\rho = \max_{i=1,\cdots,n} \{\frac{p_i}{w_i}\} \leq \nu$$

From $C_1 = n/10$, we get

$$1 + \rho(\sum_{i=1}^{n} w_i - C_1) \leq 1 + \frac{n^3}{400},$$

Since $\mid I(x^*) \mid \leq \nu$, we get

$$\exp(\sum_{i \notin I(x^*)} p_i) \geq \exp(n/10).$$

So it is proved that

$$\exp(\sum_{i \notin I(x^*)} p_i) \geq 1 + \rho(\sum_{i=1}^{n} w_i - C_1).$$

In other words,

$$f(\mathbf{1}) > f(x^*) = f_{\max}.$$

Since for any y in the neighbourhood $N_l(\mathbf{1}, n/2) := \{y;\ H(y, \mathbf{1}) < n/2\}$, it is easy to see that $f(\mathbf{1}) > f(y)$ if $y \neq \mathbf{1}$.

So the proposition is proved. ■

Proposition 2. *(1) The fitness function derived from the algorithm $A_p[2]$ belongs to Type 1, 3, 4 or 6, but doesn't belong to Types 2 or 5.*

(2) The probability for the fitness function belonging to Type 3 or 6 is small.

Proof: (1) It is possible that $f(x) > f_{\max}$ for some infeasible solution under a certain condition. Let's see such condition.

Let $x^* = (x_1 \cdots x_n)$ be the optimal solution, and I^* be the set of all items in the optimal knapsack. If for some $j \notin I^*$, it holds that

$$-p_j + \rho(\sum_{i \in I(x^*)} w_i + w_j - C_1) > 0,$$

then we can prove $f(x) > f_{\max}$ under this condition.

Let x' be the solution with $x_i = 1$ for $i \in I(x^*) \cup \{j\}$ and $x_i = 0$ for the other subscripts. Then we have

$$f(x') - f(x) = -p_j + \rho(\sum_{i \in I(x^*)} w_i + w_j - C_1) > 0.$$

So we have proved that it is possible that $f(x) > f_{\max}$. In other words, the problem is a non-equivalent problem.

Now we prove that the fitness function $f(x)$ is not a wide-gap problem.

Notice that the fitness function is

$$f(x) = \sum_{i=1}^{n} x_i p_i - \rho(\sum_{i=1}^{n} x_i w_i - C_1).$$

Let $x = (x_1 \cdots x_n)$ be an infeasible solution, i.e. $\sum_{i=1}^{n} x_i w_i \geq C$. Without losing generality, assume that $x_1 = 1$ and let $x' = (0x_2 \cdots x_n)$, then

$$f(x') - f(x) = -p_1 + \rho w_1 = w_1(-\frac{p_1}{w_1} + \rho).$$

Since

$$\rho = \max_{i=1,\cdots,n} \left\{ \frac{p_i}{w_i} \right\},$$

then we know that

– if $p_1/w_1 = \rho$, then

$$f(x') - f(x) = 0.$$

– if $p_1/w_1 < \rho$, then

$$f(x') - f(x) > 0.$$

This has proved that $f(x)$ generated by $A_p[2]$ does not belong to a wide-gap fitness landscape.

(2) Let us consider $f(x)$ belonging to a wide-plateau fitness landscape (Type 3 or Type 6). Then in this case, as proved above, at least δ pairs of p_i and w_i satisfy $p_i/w_i = \rho$.

For a given p_i, since w_i are chosen at random from $[1, \cdots, \nu]$, the probability of $p_i/w_i = \rho$ happening is no more than $1/\nu$.

And then the probability for δ pairs of p_i and w_i to satisfy $p_i/w_i = \rho$ is up to

$$\left(\frac{1}{\nu}\right)^{\delta}.$$

Since $\delta = \omega(1)$ and $\nu = n/20$, the above probability is very small. ■

Example 1. In algorithm $A_p[2]$, let $w_i = p_i = c$,

For all infeasible solutions, the penalty function is

$$g_2(x) = \sum_{i=1}^{n} cx_i - C_1,$$

and the fitness function is

$$f(x) = C_1.$$

If c is chosen to make $f_{\max} > C_1$, then the fitness function $f(x)$ is in Type 3.

If c is chosen to make $f_{\max} < C_1$, then the fitness function $f(x)$ is in Type 6.

Proposition 3. *The fitness function derived from Algorithm $A_p[3]$ belongs to Type 1 or Type 3 (narrow-gap problem).*

Proof: It is possible that the function belongs to Type 1. Let x^* be the optimal solution, and $I(x^*)$ be the set of subscripts i such that $x_i = 1$. If for some $j \notin I(x^*)$, it holds that

$$-p_j + \rho^2(\sum_{i \in I(x^*)} w_i + w_j - C_1)^2 > 0$$

then we can prove the fitness function belongs to Type 1.

Let x' be the solution with $x_i = 1$ for $i \in I(x^*) \cup \{j\}$ and $x_i = 0$ for the other subscripts. Then we have

$$f(x') - f(x) = -p_j + \rho^2(\sum_{i \in I^*} w_i + w_j - C_1)^2 > 0.$$

Now we prove the fitness function doesn't belong to Types 2, 3, 5 and 6.

Let $x = (x_1 \cdots x_n)$ be an infeasible solution, i.e. $\sum_{i=1}^{n} x_i w_i \geq C_1$. Without losing generality, assume that $x_1 = 1$ and let $x' = (0 x_2 \cdots x_n)$. Denote $I(x)$ to be the set of $\{i;\ x_i = 1\}$ and $I(x')$ the set of $\{i;\ x'_i = 1\}$, then

$$f(x') - f(x) = -p_1 - \rho^2(\sum_{i \in I(x')} x'_i w_i - C_1)^2 + \rho^2(\sum_{i \in I(x)} x_i w_i - C_1)^2$$

$$= -p_1 + \rho^2(\sum_{i \in I(x)} x_i w_i + \sum_{i \in I(x')} x'_i w_i - 2C_1) w_1.$$

Since $\sum_{i \in I(x)} x_i w_i - C_1 > 0$ and $\sum_{i \in I(x')} x'_i w_i - C_1 > 0$,

$$\rho(\sum_{i \in I(x)} x_i w_i + \sum_{i \in I(x')} x'_i w_i - 2C_1) > \rho(\sum_{i \in I(x)} x_i w_i - \sum_{i \in I(x')} x'_i w_i) = \rho w_1 \geq 1.$$

Then

$$f(x') - f(x) > -p_1 + \rho w_1 \geq 0\ .$$

From the above result it follows that $f(x') > f(x)$ and we have proved that the fitness function derived from algorithm $A_p[3]$ doesn't belong to Types 2, 3, 5, and 6. ■

Example 2. In algorithm $A_p[3]$, let $w_i = 0.5$ and $p_i = 1$, where c is a constant, then $\rho = 2$.

For any infeasible solution, its penalty is

$$g_2(x) = 4(\sum_{i=1}^{n} 0.5 x_i - C_1)^2$$

and its fitness is

$$f(x) = \sum_{i=1}^{n} x_i - 4(\sum_{i=1}^{n} 0.5 x_i - C_1)^2.$$

Let x and y be two infeasible solutions such that $\mid x \mid > \mid y \mid$.

$$f(x) - f(y) = \mid x \mid - \mid y \mid -4 \mid x \mid^2 +4C_1 \mid x \mid +4 \mid y \mid^2 -4C_2 \mid y \mid$$

$$= (\mid x \mid - \mid y \mid)(1 + 4C_1 - 4 \mid x \mid -4 \mid y \mid)$$

Since x is infeasible, then

$$0.5 \mid x \mid > C_1,$$
$$\mid x \mid > 2C_1,$$
$$(1 + 4C_1 - 4 \mid x \mid -4 \mid y \mid) < 0,$$
$$f(x) < f(y).$$

4 Analysis of the Restrictive Capacity Knapsack

4.1 Analysis of GAs using Penalty Methods

Since a problem in Type 1, 2 or 3 is not equivalent to the original knapsack problem, it is enough to discuss the first hitting time to the optimal solution only.

Proposition 4. *When a (1+1) $A_p[i], i = 1, 2, 3$ is used to solve a problem in Type 1, 2 or 3, the expected first hitting time τ in the worst case is $+\infty$.*

Proof: Since the optimal feasible solution may have smaller fitness than some infeasible individual, the optimal individual may not be accepted by the elitist selection used in the (1+1) GA. ■

Note: this trouble is caused by the static penalty. It may not exist for other methods, e.g. the method of dynamic penalty, death penalty, repairing infeasible solutions and superiority of feasible solutions etc [5].

If the problem is an equivalent problem, then the population sequence $\{\xi^{(t)}\}$ is convergent to the optimal solution [25, 26].

Proposition 5. *When a (1+1) $A_p[i], i = 1, 2, 3$ is used to solve a problem in Type 4, 5 or 6, the expected first hitting time τ in the worst case is finite.*

Proof: The conclusion is drawn directly from the elitist selection and bitwise mutation used in the GAs. ■

Proposition 6. *(1) If a $(1 + 1)$ $A_p[i], i = 1, 2, 3$ is used to solve a problem in Type 4, the expected first hitting time $\bar{\tau}_n$ is $O(n^{\delta+1})$ in the worst case and the expected number of fitness evaluations is $O(n^{\delta+1})$.*

(2) If a $(1 + 1)$ $A_p[i], i = 1, 2, 3$ is used to solve any problem in Type 4.a, the expected first hitting time $\bar{\tau}_n$ in the worst case is $\Theta(n \ln n)$ and the expected number of fitness evaluations is $\Theta(n \ln n)$.

Proof: (1) The time bound can be drawn by applying drift analysis [27] to estimate the first hitting time.

Define the distance between an individual x and S_a to be

$$d(x) = \min\{H(x, S_n)\}$$

A positive drift can happen if δ 1-valued bits are flipped. The probability this happens is at least

$$\frac{1}{n^\delta}(1-\frac{1}{n})^{n-\delta}$$

So the drift is at least

$$\Delta d(x) \geq \delta\frac{1}{n^\delta}(1-\frac{1}{n})^{n-\delta}$$

Then the mean first hitting time is

$$\mathrm{E}[\tau_n] \leq \frac{n}{\max \Delta d(x)} = O(n^{\delta+1}).$$

(2) For Type 4.a, an individual makes a positive drift towards the set S_n only if at least a 1-valued bit is flipped to 0. Since there are $\mid x \mid -1$ candidate bits, the first hitting time of the (1+1) GA to reach the set S_n on problem 4.a is not faster than for the OneMax problem.

From the result on the OneMax problem, we know the expected first hitting time is $\Omega(n \ln n)$.

In the following we prove the bound is tight. Define

$$l(x) = \min\{H(x, S_n)\}$$

and define the distance

$$d(x) = n \ln l(x).$$

Then at each generation, a positive drift towards the set S_n can happen if the following event appears: one 1-valued bit is flipped to 0.

From the definition of Type 4.a, we know that the probability of flipping one 1-valued bit and leading to the positive drift is at least

$$\binom{\mid x \mid -1}{1}\frac{1}{n}(1-\frac{1}{n})^{n-1}, \tag{8}$$

and the positive drift is at least

$$\begin{aligned}\Delta d(x) &\geq (n \ln l - n \ln(l-1))(\mid x \mid -1)\frac{1}{n}(1-\frac{1}{n})^{n-1} \\ &\geq c,\end{aligned}$$

where $c > 0$ is a constant.

Then the expected first hitting time is no more than

$$\mathrm{E}[\tau] \leq \frac{n \ln n}{\min \Delta d(x)} = O(n \ln n).$$

We have proved that the expected first hitting time to the set S_n is $\Theta(n \ln n)$. ■

If a $(1+1)$ algorithm $A_p[i], i = 1, 2, 3$ is used to solve problems in Type 6, a trivial lower bound is given as follows.

Proposition 7. *If a* $(1+1)$ *algorithm* $A_p[i], i = 1, 2, 3$ *is used to solve problems in Type 6, then the expected first hitting time* $\bar{\tau}_n$ *in the worst case is* $\Omega(n \ln n)$.

Proof: Since the wide-plateau problem is at least harder than the OneMax problem, the time bound for the (1+1) GA to solve the wide-plateau is at least $\Omega(n \ln n)$. ■

At the end let us makes a short analysis for the population-based $(N + N)$ GA.

Proposition 8. *If an* $(N + N)$ $A_p[i], i = 1, 2, 3$ *is used to solve a problem in Type 1, 2 or 3, then*

1. *the expected first hitting time* $\bar{\tau}$ *in the worst case is* $+\infty$ *if the selection strategy is "winner-take-all".*
2. *the expected first hitting time* $\bar{\tau}$ *in the worst case is finite if the selection pressure is not too high, i.e. to allow an individual with smaller fitness in the population to survive with a positive probability.*

Proof: 1) Since the selection is "winner-take-all", starting from an initial individual $\xi^{(0)} = x$ with $f(x) > f_{\max}$, then the optimal solution is never selected and appears in the next generation through the elitist strategy.

2) Since the bitwise mutation is global, a feasible solution can be generated through mutation; furthermore, since the selection has low pressure, the optimal solution can be kept in the next generation with a positive probability. Therefore, $E[\tau] < +\infty$. ■

Proposition 9. *If an* $(N + N)$ $A_p[i], i = 1, 2, 3$ *is used to solve a problem in Type 4, then the expected first hitting time in the worst case is* $\Omega(n \ln n/N)$ *and the expected number of fitness evaluations is* $\Omega(n \ln n/N)$.

Proof: Similar to the above analysis of the $(1 + 1)$ GA, the behavior of the $(N + N)$ GA in reaching the set S_a on Type 2 is similar to the behavior of the $(N + N)$ GA on the OneMax problem. Since the discussion is only involved in finding a lower bound on the first hitting time τ_f, without losing generality, it is enough to consider the case of the OneMax problem.

For a population $\mathbf{x}$, define its distance from the set S_n as

$$d(\mathbf{x}) = \min\{d(x);\ x \in \mathbf{x}\},$$

where $d(x) = H(x, S_n)$ is the Hamming distance between x and the set S_n.

Let $\mathbf{x} = (x_1, \cdots, x_N)$ with $d(x) = k$ (where $k > 0$) be the parent population and $\mathbf{y} = (y_1, \cdots, y_N)$ be the offspring population. Since here only the lower bound needs to be estimated, it is enough to consider the case of $d(x_1) = \cdots = d(x_N) = k$ and estimate the relevant drift. It is easy to see that in the other cases, the drift is less.

Notice that

$$d(\mathbf{x}) = \min\{d(x_i);\ i = 1, \cdots, N\}.$$

Since each individual is mutated independently, the events $d(x_i) - d(y_i) \geq j$ are independent. For any $j > 0$, we know that

$$\mathrm{P}[d(\mathbf{x}) - d(\mathbf{y}) \geq j] = 1 - \prod_{i=1}^{N}(1 - \mathrm{P}[d(x_i) - d(y_i) \geq j]),$$

and get,

$$\mathrm{P}[d(\mathbf{x}) - d(\mathbf{y}) = j] \leq N\mathrm{P}[d(x_1) - d(y_1) = j].$$

Consider the (1+1) GA, i.e. the case $N = 1$, and denote by σ_k the first hitting times of the (1+1) GA to the set S_n if starting from $d(x) = k$. Then $\sigma_k = \Omega(n \ln n)$ if $k = \Omega(n)$.

Let $d(\mathbf{x}) = \sigma_k/N$ if $D(\mathbf{x}) = k$, then we have

$$\Delta_k = \sum_j (d_k - d_{k-j})\mathbf{P}[d(\mathbf{x}) - d(\mathbf{y}) = j)] \leq \sum_j \left(\frac{\sigma_k}{N} - \frac{\sigma_{k-j}}{N}\right) N\mathbf{P}[d(x_1) - d(y_1) = j] = 1.$$

So by applying drift analysis [19], we obtain that if the initial population satisfies $d(\xi^{(0)}) = \Omega(n)$, the first hitting time of the $(N+N)$ GA satisfies

$$\mathbf{E}[\tau_a \mid \xi^{(1)}] \geq \frac{\sigma_{\Omega(n)}}{N} = \Omega\left(\frac{n \ln n}{N}\right). \tag{9}$$

Since at each generation the fitness is evaluated N times, the expected number of fitness evaluations is $\Omega(n \ln n)$. ■

Similar to the above theorem, we have

Proposition 10. *If an $(N + N)$ $A_p[i], i = 1, 2, 3$ is used to solve a problem in Type 6, then the expected first hitting time in the worst case is $\Omega(n \ln n/N)$ and the expected number of fitness evaluations is $\Omega(n \ln n)$.*

4.2 Algorithms Repairing Infeasible Solutions

Proposition 11. *When a (1+1) $A_p[i], i = 1, 2, 3$ is used to solve any knapsack problem, the expected first hitting time τ in the worst case is finite.*

Proof: The statement is directly drawn from the elitist selection and the bitwise mutation. ■

Proposition 12. *If a $(1 + 1)$ $A_r[i], i = 1, 2$ is used to solve any knapsack problem, then the expected first hitting time $\bar{\tau}_f$ is 1 and the expected number of fitness evaluations is $\Theta(n)$.*

Proof: At each generation, if x is infeasible, then one item will be removed from the knapsack. This procedure will be repeated until a feasible solution is found. Since at most n items are in the initial knapsack, then at most n steps are needed to find a feasible solution. This means the first hitting time τ_f is 1.

With a probability not less than $1 - e^{-n/16}$, the initial individual has more than $n/4$ items and it is infeasible. So at least $n/4 - n/10$ fitness evaluations are needed to reach the set S_f and the number of fitness evaluations is $\Theta(n)$. ■

Proposition 13. *When an (N+N) $A_p[i], i = 1, 2, 3$ is used to solve any knapsack problem, the expected first hitting time τ in the worst case is finite.*

Proposition 14. *If an $(N + N)$ $A_r[i], i = 1, 2$ is used to solve any knapsack problem, then the expected first hitting time $\bar{\tau}_n$ is 1 and the expected number of fitness evaluations is $\Theta(Nn)$.*

Proof: Since all individuals in the initial population will be infeasible with probability $1 - (1 - \exp(-n/16))^N$ and each individual has more than $n/4$ items, for each individual at least $\Theta(n)$ items have to be removed before a feasible solution is reached. Since the population size is N, the number of fitness evaluations is $\Theta(Nn)$. ■

This proposition means that the introduction of a population doesn't bring any benefit towards reducing the number of fitness evaluations.

4.3 Comparison Between Repairing Feasible Solutions and Penalizing Feasible Solutions

The results of the above two subsections are summarized in Tables 1 and 2.

From the two tables, we can see that

- The approach of repairing infeasible solutions is more efficient in finding a feasible solution. The expected first hitting time $\bar{\tau}_n$ of $A_p[i], i = 1, 2, 3$ is larger than that of $A_r[i], i = 1, 2$. The expected number of fitness evaluations of $A_p[i]$ is larger than that of $A_r[i]$.
- Since the problems produced by Algorithms $A_p[2]$ belong to a narrow-gap problem with a high probability and the problems from $A_p[3]$ are definitely a kind of narrow-gap problem, both algorithms can find a near-feasible solution in polynomial time. This result is not observed from the experiments.

4.4 A Note on the role of Crossover

In the above discussion the crossover is omitted. The analysis of crossover usually is very difficulty. Currently only theoretical analyses of GAs on some artificial functions [29, 30] are available. The paper will not make further investigations on this issue. Here we only give a note.

For the repair algorithms, since the problem is always an equivalent problem, the larger is the fitness, the better is the individual. In this case, a crossover operator may play a positive role, i.e. help GAs to find a better solution.

Table 1. Results of (1+1) GAs using a penalty function. $\bar{\tau}$, $\bar{\tau}_f$ and $\bar{\tau}_n$ are the expected first hitting times to an optimal, feasible and near-feasible solution respectively. κ is the number of fitness evaluations. "$-$" means that no result is given, "$*$" means that the algorithm doesn't produce that type.

		algorithm		
type		$A_p[1]$	$A_p[2]$	$A_p[3]$
1	$\bar{\tau}$	$*$	$+\infty$	$+\infty$
	$\bar{\tau}_f$	$*$	$-$	$-$
	$\bar{\tau}_n$	$*$	$-$	$-$
	κ	$*$	$-$	$-$
2	$\bar{\tau}$	$+\infty$	$*$	$*$
	$\bar{\tau}_f$	$-$	$*$	$*$
	$\bar{\tau}_n$	$-$	$*$	$*$
	κ	$-$	$*$	$*$
3	$\bar{\tau}$	$*$	$+\infty$	$*$
	$\bar{\tau}_n$	$*$	$-$	$*$
	$\bar{\tau}_f$	$*$	$-$	$*$
	κ	$*$	$-$	$*$
4(a)	$\bar{\tau}$	$*$	finite	finite
	$\bar{\tau}_f$	$*$	$-$	$-$
	$\bar{\tau}_n$	$*$	$\Omega(n \ln n)$	$\Omega(n \ln n)$
	κ	$*$	$\Omega(n \ln n)$	$\Omega(n \ln n)$
6(a)	$\bar{\tau}$	$*$	finite	$*$
	$\bar{\tau}_f$	$*$	$-$	$*$
	$\bar{\tau}_n$	$*$	$\Omega(n \ln n)$	$*$
	κ	$*$	$\Omega(n \ln n)$	$*$

However, for the penalty algorithms, if the problem is not equivalent to the original knapsack problem, the larger the fitness is, the worse is the individual. In this case crossover may play a negative role.

5 Average Capacity Knapsack

Lemma 2. *For any average capacity knapsack, if the initial individual is chosen at random, then the probability of an initial individual of being a feasible solution is* 0.5.

Proof: The result is directly derived from the binomial distribution. Let $x = (x_1 \cdots x_n)$ be infeasible. Hence it satisfies

$$\sum_{i=1}^{n} w_i x_i > C_2.$$

Then let $\bar{x} = (\bar{x}_1 \cdots \bar{x}_n)$ where $\bar{x}_i = 1 - x_i$. If $x \neq y$, then $\bar{x} \neq \bar{y}$.

Table 2. Results of (1+1) GAs using the repair method. $\bar{\tau}$, $\bar{\tau}_f$ and $\bar{\tau}_n$ are the expected first hitting times to an optimal, feasible and near-feasible solution respectively. κ is the number of fitness evaluations.

	$A_r[1]$	$A_r[2]$
$\bar{\tau}$	finite	finite
$\bar{\tau}_f$	1	1
$\bar{\tau}_n$	1	1
κ	$\Theta(n)$	$\Theta(n)$

Since

$$\sum_{i=1}^{n} w_i 1 = 2C_2,$$

then $\bar{x} = (\bar{x}_1 \cdots \bar{x}_n)$ satisfies

$$\sum_{i=1}^{n} w_i \bar{x}_i < C_2.$$

This means $\bar{x}$ is feasible. From the above analysis, we see

$$\mid S_f \mid \geq \mid S_i \mid .$$

If we do the calculations vice versa, we get

$$\mid S_i \mid \geq \mid S_f \mid .$$

Combining the above inequalities together, we get

$$\mid S_i \mid = \mid S_f \mid .$$

Since the initial individual is chosen at random, the conclusion is proved. ■

Corollary 2. *For any average capacity knapsack, the initial population of the (N+N) GA contains a feasible solution with a probability not less than* $1 - 0.5^N$.

Based on Lemma 2, it is reasonable to assume that in the initial population $\xi^{(0)}$, at least one individual is feasible.

Definition 1. *A local optimal set* S_l *with size* δ *is*

$$S_l := \{x;\ \forall y \in N(x, \delta) : f(x) > f(y)\}.$$

where $\delta \geq 1$ *is a constant.*

Define the global optimal set to be

$$S_g := \{x \in S_f;\ \forall y \in S : f(x) \geq f(y)\}.$$

Definition 2. *Assume $\xi^{(0)} = (x, \cdots x)$, where $x \in S_l$. Let τ_b be the first hitting time to reach a solution with a better fitness than $\xi^{(0)}$:*

$$\tau_b := \min\{t;\ f(\xi^{(t)}) > f(\xi^{(0)})\}.$$

Similar to the restrictive capacity knapsack, the average capacity knapsack can be classified into the six types too:

- **T1′: narrow-gap non-equivalent problem.** $\exists x \in S_i$, it holds

$$f(x) > f_{\max}. \tag{10}$$

  And $\forall x \in S$, $\exists y \in N(x, \delta)$ $(\delta = O(1))$ it holds:

$$f(y) > f(x).$$

- **T2′: wide-gap non-equivalent problem.** $\exists x \in S_i$, it holds

$$f(x) > f_{\max}. \tag{11}$$

  And $\exists x \in S$, $\forall y \in N(x, \delta)$ where $\delta = \omega(1)$, it holds:

$$f(x) > f(y).$$

- **T3′: wide-plateau non-equivalent problem.** $\exists x \in S_i$, it holds

$$f(x) > f_{\max}. \tag{12}$$

  And
  1. $\exists x \in S$, $\forall y \in N(x, \delta)$ where $\delta = \omega(1)$, it holds:

$$f(y) \leq f(x).$$

  2. $\forall y \in N(x, \delta)$, $\exists z \in N(y, \delta')$ where $\delta' = O(1)$, it holds:

$$f(z) = f(x).$$

- **T4′: narrow-gap equivalent problem.** $\forall y \in S_i$, it holds

$$f(y) \leq f_{\max}. \tag{13}$$

  And $\forall x \in S$, $\exists y \in N(x, \delta)$ $(\delta = O(1))$ it holds:

$$f(y) > f(x).$$

- **T5′: wide-gap equivalent problem:** $\forall y \in S_i$, it holds

$$f(y) \leq f_{\max}. \tag{14}$$

  And $\exists x \in S$, $\forall y \in N(x, \delta)$ where $\delta = \omega(1)$, it holds:

$$f(y) < f(x).$$

– **T6′: wide-plateau equivalent problem:**$\forall y \in S_i$, it holds

$$f(y) \leq f_{\max}. \tag{15}$$

And

1. $\exists x \in S$, $\forall y \in N(x, \delta)$ where $\delta = \omega(1)$, it holds:

$$f(y) \leq f(x).$$

2. $\forall y \in N(x, \delta)$, $\exists z \in N(y, \delta')$ where $\delta' = O(1)$, it holds:

$$f(z) = f(x).$$

Proposition 15. *For Algorithm $A_p[1]$, denote $x^* = (x_1^* \cdots x^*)$ to be a global optimal solution , $I(x^*)$ the set of items in the optimal knapsack and $J(x^*)$ the items outside the optimal knapsack.*

1. *if the following condition holds:*

$$\exp(\sum_{i \notin I(x^*)} p_i) > 1 + 0.5\rho(\sum_{i=1}^{n} w_i)$$

then the problem derived from the algorithm is a non-equivalent problem (one of Types $1'$, $2'$, or $3'$).
2. *if the above condition doesn't hold, then the landscape derived from it is an equivalent problem (one of Types $4'$, $5'$, or $6'$).*

Proof: (1) For any infeasible solution x, its penalty term is

$$g(x) = \ln(1 + \rho(\sum_{i=1}^{n} w_i x_i - C_1)),$$

and its fitness is

$$f(x) = \sum_{i=1}^{n} p_i x_i - \ln(1 + \rho(\sum_{i=1}^{n} w_i x_i - C_1)).$$

Let $\mathbf{1} = (1 \cdots 1)$, whose fitness is

$$f(x) = \sum_{i=1}^{n} p_i - \ln(1 + \rho(\sum_{i=1}^{n} w_i - C_1)).$$

Since x^* is feasible, we have

$$\sum_{i \in I(x^*)} x_1^* w_i \leq C_2,$$

Notice that

$$f(\mathbf{1}) - f(x^*) = \sum_{i=1}^{n} p_i - \sum_{i \in I(x^*)} p_i - \ln(1 + \rho(\sum_{i=1}^{n} w_i - C_1))$$

$$= \sum_{i \notin I(x^*)} p_i - \ln(1 + \rho(\sum_{i=1}^{n} w_i - C_1))$$

Because of the condition

$$\exp(\sum_{i \notin I(x^*)} p_i) \geq 1 + \rho(\sum_{i=1}^{n} w_i - C_1)$$

Now we come to

$$f(\mathbf{1}) > f(x^*).$$

(2) Notice $(1 \cdots 1)$ is the infeasible solution with the maximum fitness value among all infeasible solutions. If the condition doesn't hold, then we will have

$$f(1) \leq f_{\max}.$$

Then the problem belongs to an equivalent problem. ■

Corollary 3. *For Algorithm $A_p[1]$, if the cardinality $\mid J(x^*) \mid \geq 3 \ln n$, then the problem is a non-equivalent problem.*

Proof: Since $p_i \geq 1$, then

$$\sum_{j \in J(x^*)} p_i \geq 3 \ln n$$

and then

$$\exp(\sum_{j \in J(x^*)} p_i) \geq n^3.$$

Since $w \leq \nu$ and $\rho \leq n$, we get

$$1 + 0.5\rho(\sum_{i=1}^{n} w_i) \leq n^2 \nu \leq 0.1 n^3.$$

Then we get

$$\exp(\sum_{j \in J(x^*)} p_i) > 1 + 0.5\rho(\sum_{i=1}^{n} w_i)$$

From the above proposition, it is drawn that the fitness landscape derived from the algorithm is that of a non-equivalent problem. ■

The corollary shows that only when the optimal knapsack includes many items, the problem derived from Algorithm $A_p[1]$ can belong to an equivalent problem. This event may happen with a small probability. The estimation of such probability needs further investigations.

Proposition 16. *The problem derived from Algorithm $A_p[2]$ can belong to Type $1'$, $2'$, $3'$, $4'$, $5'$ and $6'$.*

Denote $x^* = (x_1 \cdots x_n)$ to be the optimal solution, $I(x^*)$ the set of items in x^* and $J(x^*)$ outside x^*. If for some $j \notin I(x^*)$, it holds that

$$-p_j + \rho(\sum_{i \in I(x^*)} w_i + w_j - C_2) > 0,$$

then the problem derived from the algorithm is a non-equivalent problem.

Proof: It is enough to prove $f(x) > f_{\max}$ for some infeasible solution x.

Let $x^* = (x_1 \cdots x_n)$ be the optimal solution. Then

$$f(x^*) = \sum_{i \in I(x^*)} p_i.$$

Let x' be the solution with $x_i = 1$ for $i \in I^* \cup \{j\}$ and $x_i = 0$ for other subscripts.

$$f(x') = \sum_{i \in I^* \cup \{j\}} p_i - \rho(\sum_{i=1}^{n} w_i x_i - C_2).$$

Then

$$f(x') - f(x^*) = p_j - \rho(\sum_{i \in I} w_i + w_j - C_2) > 0.$$

Since for some $j \in J(x^*)$, it holds that

$$p_j - \rho(\sum_{i \in I(x^*)} w_i + w_j - C_2) > 0,$$

then

$$f(x') > f_{\max}$$

for the infeasible solution x'. ■

Example 3. A non-equivalent problem. In $A_p[2]$, let $w_i = p_i = 1$ and n an odd number.

For all infeasible solutions, the penalty function is

$$g_2(x) = \sum_{i=1}^{n} x_i - C_1,$$

where $C_1 = 0.5n$ and the fitness function is

$$f(x) = \sum_{i=1}^{n} x_i - g_2(x) = C_1.$$

Since n is an odd number, then a feasible solution satisfies:

$$\sum_{i=1}^{n} x_i \leq 0.5n$$

and we have

$$f_{\max} = \frac{n-1}{2}$$

But

$$f(\mathbf{1}) = 0.5n$$

and then we have

$$f(\mathbf{1}) > f_{\max}.$$

Example 4. A wide-gap equivalent problem.

$$w_1 = n-1, w_2 = \cdots w_n = 1,$$
$$p_1 = 10^n, p_2 = \cdots = p_n = 1.$$

Then $(10\cdots0)$ is the only optimal point.

Let $x = (01\cdots1)$ which is a feasible solution. And for any feasible solution y in its neighbourhood $N(x, n-2)$, it holds:

$$f(y) < f(x).$$

Proposition 17. *For $A_p[3]$, denote x^* to be the optimal solution, and $I(x^*)$ and $J(x^*)$ as in the previous propositions. If for some $j \in J(x^*)$, it holds that*

$$p_j - \rho^2(\sum_{i\in I(x^*)} w_i + w_j - C_2)^2 > 0$$

then the problem derived from Algorithm $A_p[3]$ belongs to the non-equivalent problem.

Proof: The proof is similar to that of the above proposition. Let x' be the solution with $x_i = 1$ for $i \in I(x^*) \cup \{j\}$ and $x_i = 0$ for other subscripts. Then

$$f(x') - f(x^*) = p_j - \rho^2(\sum_{i\in I^*} w_i + w_j - C_1)^2 > 0.$$

In other words, $f(x') > f(x^*)$ where x' is infeasible. ■

6 Analysis of GAs for Average Capacity Knapsack

6.1 GAs of Penalizing Infeasible Solutions

Proposition 18. *Given an average capacity knapsack,*

1. *if the fitness function derived from Algorithms* $A_p[i], i = 1, 2, 3$ *is an equivalent problem, then Algorithms* $A_p[i], i = 1, 2, 3$ *may need an exponential time to find the optimal solution in the worst case.*
2. *If the fitness function derived from Algorithms* $A_p[i], i = 1, 2, 3$ *is an equivalent problem, then the (1+1) Algorithms* $A_p[i], i = 1, 2, 3$ *cannot find the optimal feasible solution.*

Proof: (1) Notice that the example of wide-gap problem 4 is a kind of deceptive problem. Such deceptive problem is hard for genetic algorithms. So Algorithms $A_p[i], i = 1, 2, 3$ need exponential time to find the optimal solution to the problem.

(2) If the GA starts with an infeasible solution whose fitness is greater than $f_{\max}$, then it cannot find the optimal solution due to the elitism of the selection operator. ■

Proposition 19. *Given an equivalent problem, if the fitness of all feasible solutions is better than that of all infeasible solutions, then Algorithms* $A_p[i], i = 2, 3$ *start from a feasible solution and the first hitting time to a local optima* S_l *with neighbour size 1 is*

$$E[\tau_l] = O(n^2).$$

Proof: Notice that if an individual x is not on a local optima with neighbourhood size 1, then there is some y such that $f(x) < f(y)$ and $H(x, y) = 1$. The probability for x to mutate into y is $\Omega(1/n)$, and the procedure can last up to $O(n)$ steps, so only $O(n^2)$ generations are needed for a feasible solution x to reach a local optima with neighbourhood size 1. ■

Proposition 20. *For the narrow-gap equivalent problem, a* $(1 + 1)$ $A_p[i], i = 1, 2, 3$ *can find a better solution in* $O(n^\delta)$ *generations.*

Proof: Given an individual x, the Hamming distance from x to a better solution y is only $H(x, y) = \delta$. The probability for x to generate a better offspring y with $f(y) > f(x)$ and $H(x, y) = \delta$ is not less than

$$\frac{1}{n^\delta}(1 - \frac{1}{n})^{n-\delta} = \Omega(\frac{1}{n^\delta}).$$

And the drift is not less than

$$\Delta(x) = \Omega(\frac{1}{n^\delta})$$

So the expected first hitting time τ_b is

$$E[\tau_b] = \frac{n}{\max \Delta(x)} = O(n^{\delta+1}).$$

This means the GA can find a better solution quickly if $\delta = O(1)$. ■

Proposition 21. *For the wide-gap or wide-plateau equivalent problem, starting from the feasible solution x, a $(1+1)$ $A_p[i], i = 1, 2, 3$ can find a better solution in $O(n^{\delta})$ generations.*

Proof: The proof is similar to that of the above proposition. The only difference is that: $\delta = \omega(1)$. ■

The above propositions give some general but rough upper bounds. More accurate upper bounds are dependent on the instance.

6.2 GAs Repairing Infeasible Solutions

Similar to the analysis of the GAs penalizing feasible solutions, we have the following results about the GAs using the repair method. The only difference is that all the problems derived from the repair method algorithms are a kind of equivalent problem.

Proposition 22. *Given an average capacity knapsack, algorithms $A_r[i], i = 1, 2$ may need exponential time to find the optimal solution in the worst case.*

Proposition 23. *Given any average capacity knapsack, algorithms $A_r[i], i = 1, 2$ can find a local optima with neighbourhood size 1 in $O(n^2)$ generations.*

Proposition 24. *For a narrow-gap problem, the $(1+1)$ $A_r[i], i = 1, 2, 3$ can find a better solution in $O(n^{\delta+1})$ generations, where $\delta = O(1)$.*

Proposition 25. *For the wide-gap or wide-plateau problem, starting from a feasible solution x, a $(1+1)$ $A_r[i], i = 1, 2$ can find a better solution in $O(n^{\delta+1})$ generations, where $\delta = \omega(1)$.*

6.3 A Comparison Between Repairing Infeasible Solutions and Penalizing Infeasible Solutions

Given an average capacity knapsack, the genetic operators and the fitness function of algorithms A_p and A_r are the same in the feasible solution area. So if the search is restricted to the feasible solution area, then the behavior of these two algorithms are equivalent.

The behavior of algorithms A_p and A_r are different only in the infeasible solution area. In the following it is assumed that an infeasible solution is generated.

Let's investigate the event of algorithms A_p and A_r generating an infeasible solution. Given an individual x, denote the items in the knapsack by

$$I(x) = \{i \mid x_i = 1\}$$

and the items outside the knapsack by

$$J(x) = \{j \mid x_j = 0\}.$$

The individual x is mutated into an infeasible solution only when at least one item from $J(x)$ is added, (where the added items are denoted by $A(x)$), several (or null) items are removed from $I(x)$ (where the removed items are denoted by $R(x)$), and the constraint is violated:

$$\sum_{i \in I(x)} w_i - \sum_{i' \in R(x)} w_{i'} + \sum_{j \in A(x)} w_j > C_2.$$

In practice, the maximum generation T is always fixed, for example, $T = 500$ in Michalewicz's experiment when $n = 100, 250, 500$ [14]. Therefore it is reasonable to assume that $T \leq 5n$ in the paper.

Let k be the number of items in $J(x)$ added into x simultaneously. The probability of adding k items is no more than

$$\binom{n}{k} \left(\frac{1}{n}\right)^k$$

If k is beyond a constant, for example, $k = \ln n$, then the above probability is very small, so it may seldom happen if the maximum generation is $T \leq 5n$.

The analysis of a population-based GA is usually very complex. Here a simple idea is followed: if a better offspring x'' appears after a few generations, then the individual should come from some parent x (due to no crossover). It is necessary to trace how the individual x generates x''. The individual x is mutated to an infeasible solution x' first.

- With the penalty method, x' survives in the selection phase and enters the next generation with some probability. In the next generation, it is possibly mutated into the individual x'' with a better fitness $f(x'') > f(x)$, or in more longer generations, x' is mutated to x''.

  From the above analysis, it is seen that at least two generations are needed with the penalty method to generate a better offspring: first an infeasible individual is generated, and then in the next generation or longer, the infeasible individual is mutated into a feasible solution.
- With the repair method, x' is repaired to generate a feasible solution x'' within the same generation.

This idea shows that it is possible to analyze the behavior of populations by analyzing that of individuals.

Let's start the discussion from the event of adding one item into the individual x.

Event 1 *Given an individual x, one item j from $J(x)$ is added into x (x' is an infeasible solution); then an item i is removed from x', and a better offspring x'' is generated.*

Analysis of algorithm $A_r[2]$. Let's see when the above event will happen: let the item $i_{\min} \in I(x)$ be the item such that

$$\arg \min_{i \in I(x)} \left\{ \frac{p_i}{w_i} \right\}.$$

There exists some item $j \in J(x)$ which satisfies,

$$p_j > p_{i_{\min}}, \tag{16}$$

$$\frac{p_j}{w_j} \geq \frac{p_{i_{\min}}}{w_{i_{\min}}}. \tag{17}$$

Denote $J_a(x)$ to be the set of all items $j \in J(x)$ satisfying the above conditions.

Starting from the individual x, the repair method $A_r[2]$ chooses an item j from $J_a(x)$ with a probability not less than

$$c\binom{\mid J_a(x) \mid}{1}\frac{1}{n} = \Omega\left(\frac{\mid J_a(x) \mid}{n}\right).$$

where c is a constant. Then the repair method removes the minimum item $i_{\min}$ and finds a better solution with probability 1.

So if $J_a(x)$ is not empty the probability for $A_r[2]$ of producing a better solution is at least

$$\Omega\left(\frac{\mid J_a(x) \mid}{n}\right). \tag{18}$$

Note: if the set $J_a(x)$ is empty, the above probability is 0, and the repair procedure plays no role in handling infeasible solutions.

However since the discussion here is restricted to the early search phase, i.e. $T \leq 5n$, the event of $J_a(x)$ being empty may seldom happen, especially in the initial population or very early phase populations.

From the above analysis, Algorithm $A_r[1]$ can find a better solution x quickly until the condition holds: for all $j \in J(x)$,

$$\frac{p_j}{w_j} < \frac{p_{i_{\min}}}{w_{i_{\min}}}.$$

Analysis of algorithm $A_r[1]$. Let's see when Event 1 will happen: there exists some item $j \in J(x)$, and j satisfies for some $i \in I(x)$,

$$p_j > p_i \tag{19}$$

Denote $J_b(x)$ to be the set of all item $j \in J(x)$ satisfying the above condition. It is obvious that $J_a(x) \subset J_b(x)$.

Given an item $j \in J_b(x)$, denote the items i with $p_i \leq p_j$ by $I_b(x, j)$.

Starting from individual x, Algorithm $A_r[1]$ first adds one item j from $J_b(x)$ with some probability and then an infeasible solution may be generated. The probability of such event happening is not less than

$$c\binom{\mid J_b(x) \mid}{1}\frac{1}{n} = \Omega\left(\frac{\mid J_b(x) \mid}{n}\right),$$

where c is a constant.

Assume one item $j \in J_b(x)$ is added. Then the algorithm repairs the infeasible solution by removing one item from $I(x)$ at random. It will remove an item $i \in I_b(x, j)$ with a probability

$$\frac{\mid I_b(x,j) \mid}{\mid I(x) \mid}$$

So the probability of generating a better solution is

$$\Omega\left(\sum_{j \in J_b(x)} \frac{1}{n} \frac{\mid I_0(x,j) \mid}{\mid I(x) \mid}\right) \tag{20}$$

Analysis of algorithm A_p. It is similar to the analysis of Algorithm A_r[1]. Let's see when Event 1 will happen: there exists some $j \in J(x)$, and j satisfies for some $i \in I(x)$,

$$p_j > p_i \tag{21}$$

Still denote $J_b(x)$ to be the same set of all items $j \in J(x)$ satisfying the above condition. Given an item $j \in J_b(x)$, denote the items i with $p_i \leq p_j$ by $I_b(x, j)$.

Starting from x, the probability of x generating an infeasible solution by adding one item $j \in J_b(x)$ is no more than

$$c\binom{\mid J_b(x) \mid}{1}\frac{1}{n} = O\left(\frac{\mid J_b(x) \mid}{n}\right).$$

Then the individual survives the selection phase with some probability, denoted by $q(x')(q(x') < 1)$. This probability is dependent on the selection pressure and the fitness of the other individuals in the population.

In the next generation, one item $i \in I_b(x, j)$ is removed and a better individual is generated with a probability not higher than

$$c\binom{\mid I_b(x,j) \mid}{1}\frac{1}{n}.$$

If the above three events are considered together, the probability of generating a better individual in two generations is

$$O\left(q \sum_{j \in J_b(x)} \frac{1}{n} \frac{\mid I_b(x,j) \mid}{n}\right). \tag{22}$$

However sometimes no better individual is generated in the second generation, so we should investigate the probability of generating a better individual in the third generation.

Similar to the above analysis, the probability is no more than

$$O\left(q^2 \sum_{j \in J_b(x)} \frac{1}{n} \frac{\mid I_b(x,j) \mid}{n}\right). \tag{23}$$

By deduction, the total probability of x generating a better individual in up to T generations is

$$O\left((q+q+\cdots+q^T)\sum_{j\in J_b(x)}\frac{1}{n}\frac{\mid I_b(x,j)\mid}{n}\right), \tag{24}$$

$$=O\left(\frac{q(1-q^{T+1})}{1-q}\sum_{j\in J_b(x)}\frac{1}{n}\frac{\mid I_b(x,j)\mid}{n}\right), \tag{25}$$

This means the probability of algorithm $A_p[i], i=1,2,3$ producing a better offspring is smaller than that of the algorithm $A_r[1]$.

From the above upper bound, it is seen that if the selection pressure is smaller, i.e. q is bigger, then the upper bound on the probability of generating a better individual may be larger under Event 1. This means that among the three algorithms $A_p[1], A_p[2], A_p[3]$, $A_p[1]$ may have the biggest probability to generate a better offspring.

However, this needs a precondition, i.e. the fitness of all feasible solutions is better than that of infeasible solutions. As proved before, for Algorithm $A_p[1]$, if the cardinality $\mid J(x^*)\mid \geq 3\ln n$, then the problem to maximize the fitness function is not equivalent to the knapsack problem. Since for many instances, it holds that $\mid J(x^*)\mid \geq 3\ln n$, Algorithm $A_p[1]$ may find some infeasible solution with a large fitness rather than a feasible solution. So it may be the worst among the three penalty algorithms.

By using the same analysis, we consider the event of adding one item and removing two items.

Event 2 *Given an individual x, one item j from $J(x)$ is added into x, and the offspring x' is an infeasible solution. Then two items $i_1, i_2 \in I(x)$ are removed from x', and a better offspring x'' is generated.*

This event can happen when there exists one item $j \in J(x)$ and for this item j, there are two items $i_1, i_2 \in I(x)$,

$$p_j > p_{i_2} + p_{i_1} \tag{26}$$

$$p_j < p_{i_1}, \quad p_j < p_{i_2}, \tag{27}$$

$$\sum_{i\in I(x)} w_i - w_{i_1} - w_{i_2} + w_j \leq C_2. \tag{28}$$

Denote the set of all items j satisfying the above condition by $J_c(x)$. And for the given item $j \in J_c(x)$, denote the items i_1, i_2 that satisfy the above conditions to be $I_c(x,j)$.

Analysis of algorithm $A_r[2]$. Let's see when the event will happen: let $i_{\min 1} \in I(x)$ be the item such that

$$\arg\min_{i\in I(x)}\frac{p_i}{w_i}$$

and $i_{\min 2} \in I(x)$ the item such that

$$\arg\min_{i \in I(x) \setminus \{i_{\min 1}\}} \frac{p_i}{w_i}.$$

There is one item $j \in J_c(x)$ and for the above two items $i_{\min 1}, i_{\min 2} \in I(x)$,

$$\frac{p_j}{w_j} \geq \frac{p_{i_{\min 1}}}{w_{i_{\min 1}}}, \quad \frac{p_j}{w_j} \geq \frac{p_{i_{\min 2}}}{w_{i_{\min 2}}} \tag{29}$$

Denote $J_d(x)$ to be the set of all the above $j \in J_c(x)$ satisfying the above condition.

Starting from individual x, Algorithm $A_r[2]$ first adds one item $j \in J_d(x)$ in x and an infeasible solution is generated. The probability of this event is not less than

$$c\binom{\mid J_d(x) \mid}{1}\frac{1}{n} = \Omega\left(\frac{\mid J_d(x) \mid}{n}\right). \tag{30}$$

where c is a constant.

Then the repair method removes two items $i_{\min 1}$ and $i_{\min 2}$ with probability 1.

So the probability of Algorithm $A_r[2]$ of finding a better solution is

$$\Omega\left(\frac{\mid J_d(x) \mid}{n}\right). \tag{31}$$

Note: if the set $J_d(x)$ is empty, the above probability is 0, and the repair procedure plays no role in handling infeasible solutions.

Analysis of algorithm $A_r[1]$. Starting from x, Algorithm $A_r[1]$ first adds one item j from $J_c(x)$, then an infeasible solution is generated. The probability of this event happening is no less than

$$c\binom{\mid J_c(x) \mid}{1}\frac{1}{n} = \Omega\left(\frac{\mid J_c(x) \mid}{n}\right),$$

where c is a constant.

The algorithm then repairs the infeasible solution by removing two items from $I_c(x, j)$ at random. The probability is not less than

$$\left(\frac{\mid I_c(x,j) \mid}{\mid I(x) \mid}\right)^2$$

So the probability for the algorithm to generate a better solution is

$$\Omega\left(\sum_{j \in J_c(x)} \frac{1}{n} \frac{\mid I_0(x,j) \mid^2}{\mid I(x) \mid^2}\right) \tag{32}$$

Analysis of algorithm A_p. Starting from x, one item is added to individual x. Then an infeasible x' is generated. This event happens with a probability no more than

$$c\binom{\mid J_c(x) \mid}{1}\frac{1}{n} = O\left(\frac{\mid J_c(x) \mid}{n}\right).$$

And then the individual x' survives in the selection phase with a probability $q(x')(q(x') < 1)$ and the total probability is no more than

$$O\left(q\frac{\mid J_c(x) \mid}{n}\right).$$

In the next generation, two items $j_1, j_2 \in I_c(x, j)$ are removed with a probability no more than

$$c\binom{\mid I_c(x,j) \mid}{2}\frac{1}{n^2},$$

where c is a constant.

If the above three events are considered together, then the event will happen with a probability no more than

$$O\left(q\sum_{j\in J_c(x)}\frac{1}{n}\frac{\mid I_c(x,j) \mid^2}{n^2}\right). \tag{33}$$

If no better individual is generated in the second generation, then in the next or more generations, two items $i_1, i_2 \in I_c(x, j)$ are removed. This event happens with a probability no more than

$$\Omega\left(q^2\sum_{j\in J_(x)}\frac{1}{n}\frac{\mid I_c(x,j) \mid^2}{n^2}\right). \tag{34}$$

By deduction, the probability for Algorithm A_p to generate a better solution in up to T generations is

$$O\left(q\frac{1-q^T}{1-q}\sum_{j\in J_c(x)}\frac{1}{n}\frac{\mid I_c(x,j) \mid^2}{n^2}\right). \tag{35}$$

This means that the probability of algorithm $A_p[i], i = 1, 2, 3$ producing a better offspring is smaller than that of the algorithm $A_r[1]$ for Event 2.

A similar analysis can be applied to other more complex events, for example,

Event 3 *add two items $j_1, j_2 \in J(x)$ to generate a new infeasible individual x' first; then remove one item i from $I(x)$ to generate a better individual.*

The following condition holds

$$p_{j_1} + p_{j_2} > \min_{i\in I(x)}\{p_i\}$$

but

$$p_{j_1} < \min_{i \in I(x)} \{p_i\}$$

$$p_{j_2} < \min_{i \in I(x)} \{p_i\}$$

$$\sum_{i \in I(x)} w_i + w_{j_1} + w_{j_2} - \sum_{k \in R(x)} w_k < C_2$$

The analysis is the same as before, but we will not do it again.

The following example shows the repairing method A_r may be much better than the penalty method A_p and the repairing algorithm $A_r[1]$.

Example 5. Example of a wide-gap equivalent problem 4.

Let $x = (01 \cdots 1)$ be the local optimum. Algorithm $A_r[1]$ or $A_r[2]$ will add the first item into the knapsack with a probability $\Omega(1/n)$, and through the greedy repairing or random repairing, the global optima $(10 \cdots 0)$ then is generated.

However, for any penalty method, if the first item is added, then the probability of removing all other items and keeping the first item at the same time is very small.

This shows that the repair algorithm A_r is much better than the penalty algorithm A_p on this example.

7 Experiments

The above theoretical explanation has shown that GAs using the repair method are more efficient than GAs using the penalty method, at least in the early search phase. Due to the following two reasons, it is necessary to implement a comparative experiment among the five GAs $A_p[i], i = 1, 2, 3$ and $A_r[i] = 1, 2$.

1. The result about the average capacity knapsack is different from the experimental observations reported in [14]. It will be useful to find out whether Algorithm $A_p[1]$ has the best performance as claimed in [14] or Algorithm $A_p[1]$ has a worse performance as predicted in this paper.
2. The GAs considered in this paper don't employ a crossover operator. It is necessary to implement computer experiments for both GAs using crossover and not, although it is believed the introduction of crossover will not change the theoretical claim.

The experimental setting is given as follows: the profits and weights in the knapsack are initialized at random; the initial population is generated at random. $\nu = r = n/20$. Bitwise mutation with rate $p_m = 0.05$ is the same as in [14]. The crossover rate is $p_c = 0.65$. The repairing algorithms $A_r[1]$ and $A_r[2]$ use repairing rate 1. The maximum number of generation is set to be 500. For both the tests on GAs with crossover and without crossover, the experiment includes 45 instances of restrictive capacity knapsacks and 45 average capacity knapsacks, where 15

instances are assigned for each item size $100, 250, 500$. The result is averaged over 10 independent runs. The program is coded in Java and implemented on a SUN AMD 64 Opteron desktop computer. The total time for GAs without crossover is 291 minutes 19 seconds and the total time for GAs with uniform crossover is 4,756 minutes 10 seconds.

Table 3 shows the number of instances in which a GA has produced the best solution among five GAs. From the table, we see that for GAs without crossover, $A_r[2]$ is always the best one on 90 instances. However, if crossover is introduced in the GAs, for 6 instances the performance of algorithm $A_r[2]$ is worse than that of $A_p[2]$ or $A_p[3]$. So further theoretical explanations of the role of crossover are needed as indicated in the previous section.

Table 3. The number of instances which a GA has produced the best solution among five GAs.

	$A_p[1]$	$A_p[2]$	$A_p[3]$	$A_r[1]$	$A_r[2]$
no crossover	0	0	0	0	90
with crossover	0	3	3	0	84

Table 4 gives typical experimental results of the five GAs $A_p[i], i = 1, 2, 3$ and $A_r[i], i = 1, 2$ without using a crossover operator.

Table 5 lists typical experimental results of the five GAs using a uniform crossover operator.

From both tables, it is seen that the experimental results have supported the theoretical claims of the previous sections:

- Algorithm $A_r[2]$ is the best among five algorithms for solving both restrictive capacity and average capacity knapsack problems, no matter whether crossover is used or not.
- Algorithm $A_p[1]$ sometimes cannot find a feasible solution to average capacity knapsack problems and is worse than others.
- Among three GAs using the penalty method, $A_p[2]$ is best.

8 Conclusions and Discussions

There exists a wide gap between the theory and practice of GAs. It is necessary to bridge the gap between them. The paper has provided a good example to demonstrate how a theoretical analysis can play an important role in explaining experimental results of GAs.

Different constraint handling techniques have been incorporated with GAs, however most of current studies are based on computer experiments. An example is Michalewicz's comparison of GAs using different constraint handling techniques on the 0-1 knapsack problem [14]. The following phenomena are observed in the experiments:

Table 4. The experiments results of five GAs without crossover. C_1 restrictive capacity knapsack, C_2 average capacity knapsack. "$*$" means no feasible solution is found during 500 generations.

Correlation	No. of items	Capacity	$A_p[1]$	$A_p[2]$	$A_p[3]$	$A_r[1]$	$A_r[2]$
none	100	C_1	$*$	9.99	13.7	15.9	28.5
		C_2	76.1	138.7	135.5	152.9	184.9
	250	C_1	$*$	50.7	54.2	73.8	132.6
		C_2	476.9	962.8	930.1	995.4	1289.9
	500	C_1	$*$	130.9	156.7	223.3	491.0
		C_2	1491.3	3602.3	3533.2	3703.9	5081.8
weak	100	C_1	$*$	$*$	$*$	12.3	27.0
		C_2	$*$	240.8	230.8	244.8	306.1
	250	C_1	$*$	$*$	$*$	34.1	122.4
		C_2	1093.8	1308.3	1282.9	1370.6	1809.9
	500	C_1	$*$	$*$	$*$	66.4	414.6
		C_2	2439.2	5219.2	5172.5	5473.9	7462.3
strong	100	C_1	$*$	$*$	$*$	23.4	46.4
		C_2	403.4	411.0	406.7	409.6	441.9
	250	C_1	$*$	$*$	$*$	65.8	222.2
		C_2	$*$	2316.9	2292.3	2355.9	2830.1
	500	C_1	$*$	$*$	$*$	131.2	706.5
		C_2	6968.5	9658.7	9568.6	9836.9	11704.7

1. On one hand, the penalty method needs more generations to find a feasible solution to the restrictive capacity knapsack than the repair method.
2. On the other hand, the penalty method can find better solutions to the average capacity knapsack than the repair method.

Such observations need a theoretical explanation. This paper has provided such a theoretical analysis for Michalewicz's experiments [14]. The main result of the paper is that GAs using the repair method are more efficient than GAs using the penalty method on both restrictive capacity and average capacity knapsack problems. The theoretical result about the average capacity knapsack is a little bit different from Michalewicz's experimental results. So supplemental experiments have been implemented to support the theoretical claim in the paper.

Since the repair method has used some kind of knowledge about the knapsack problem, this theoretical study has confirmed a general principle used in practice: a problem-specific constraint-handling technique (here the repair method) performs better than general-purpose techniques (here the penalty method). A good and efficient constraint-handling technique must exploit domain specific knowledge. In combinatorial optimization, the repair method may be regarded as the best choice in handling constraints [5].

There are many constraint-handling techniques incorporated in GAs. This paper only makes an initial analysis of two of them: the penalty method and the repair method. In the future, it is possible to discuss other constraint handling

Table 5. The experiments results of five GAs with crossover. C_1 restrictive capacity knapsack, C_2 average capacity knapsack. "$*$" means no feasible solution is found during 500 generations.

Correlation	No. of items	Capacity	$A_p[1]$	$A_p[2]$	$A_p[3]$	$A_r[1]$	$A_r[2]$
none	100	C_1	$*$	$*$	$*$	10.1	23.4
		C_2	129.3	157.8	159.5	165.5	194.5
	250	C_1	$*$	$*$	$*$	26.0	108.6
		C_2	221.9	929.0	922.9	978.9	1279.9
	500	C_1	$*$	$*$	$*$	78.0	486.4
		C_2	3161.5	3661.3	3620.1	3755.2	5015.1
weak	100	C_1	$*$	$*$	$*$	12.6	29.9
		C_2	103.4	207.3	194.0	212.7	262.6
	250	C_1	$*$	$*$	$*$	38.9	138.5
		C_2	200.6	1245.8	1161.0	1335.5	1771.6
	500	C_1	$*$	$*$	$*$	85.1	461.8
		C_2	3244.5	5185.1	5120.1	5391.1	7276.6
strong	100	C_1	$*$	$*$	$*$	24.0	41.9
		C_2	155.6	359.4	335.8	389.2	444.0
	250	C_1	$*$	$*$	$*$	62.7	204.0
		C_2	1506.7	2370.0	2368.8	2479.2	2857.6
	500	C_1	$*$	$*$	$*$	128.1	710.6
		C_2	4881.0	9709.2	9588.6	9977.5	11709.4

methods. But it must be kept in mind that none of the constraint-handling techniques can be superior to the others over all problems [5].

Acknowledgment

The authors would like to thank Prof. Z. Michalewicz and P.S. Oliveto for their valuable comments on the draft of the paper.

References

1. Z. Michalewicz and M. Schoenauer, "Evolutionary computation for constrainted parameter optimization problem," *Evolutionary Computation*, vol. 4, no. 1, pp. 1–32, 1996.
2. Z. Michalewicz and C. Z. Janikow, "Genocop: A genetic algorithm for numerical optimization problems with linear constraints." *Commun. ACM*, vol. 39, no. 12es, pp. 175–201, 1996.
3. S. Koziel and Z. Michalewicz, "Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization." *Evolutionary Computation*, vol. 7, no. 1, pp. 19–44, 1999.
4. K. Deb, "An efficient constraint handling method for genetic algorithms," *Computer Methods in Applied Mechanics and Engineering*, vol. 186, no. 2-4, pp. 311–338, 2000.

5. C. A. C. Coello, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art," *Computer Methods in Applied Mechanics and Engineering*, vol. 191, no. 11, pp. 1245–1287, 2002.
6. B. G. W. Craenen, A. E. Eiben, and J. I. van Hemert, "Comparing evolutionary algorithms on binary constraint satisfaction problems." *IEEE Trans. Evolutionary Computation*, vol. 7, no. 5, pp. 424–444, 2003.
7. T. P. Runarsson and X. Yao, "Stochastic ranking for constrained evolutionary optimization." *IEEE Trans. Evolutionary Computation*, vol. 4, no. 3, pp. 284–294, 2000.
8. M.-J. Tahk and B.-C. Sun, "Coevolutionary augmented lagrangian methods for constrained optimization." *IEEE Trans. Evolutionary Computation*, vol. 4, no. 2, pp. 114–124, 2000.
9. H. Myung and J.-H. Kim, "Multiple lagrange multiplier method for constrained evolutionary optimization." *Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol. 4, no. 2, pp. 158–163, 2000.
10. S. Venkatraman and G. G. Yen, "A generic framework for constrained optimization using genetic algorithms." *IEEE Trans. Evolutionary Computation*, vol. 9, no. 4, pp. 424–435, 2005.
11. T. Takahama and S. Sakai, "Constrained optimization by applying the *alpha* constrained method to the nonlinear simplex method with mutations." *IEEE Trans. Evolutionary Computation*, vol. 9, no. 5, pp. 437–451, 2005.
12. T. P. Runarsson and X. Yao, "Search biases in constrained evolutionary optimization." *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, vol. 35, no. 2, pp. 233–243, 2005.
13. E. Mezura-Montes and C. A. C. Coello, "A simple multimembered evolution strategy to solve constrained optimization problems." *IEEE Trans. Evolutionary Computation*, vol. 9, no. 1, pp. 1–17, 2005.
14. Z. Michalewicz, *Genetic Algorithms + Data Structure = Evolution Program*, 3rd ed. New York: Springer Verlag, 1996.
15. S. Droste, T. Jansen, and I. Wegener, "On the analysis of the $(1+1)$ evolutionary algorithms," *Theoretical Computer Science*, vol. 276, no. 1-2, pp. 51–81, 2002.
16. I. Wegener, "Methods for the analysis of evolutionary algorithms on pseudo-Boolean functions," in *Evolutionary Optimisation*, R. Sarker, M. Mohammadian, and X. Yao, Eds. Boston: Kluwer Academic Publisher, 2002, chapter 14, pp. 349–370.
17. H.-G. Beyer, H.-P. Schwefel, and I. Wegener, "How to analyse evolutionary algorithms," *Theoretical Computer Science*, vol. 287, no. 1, pp. 101–130, 2002.
18. J. He and X. Yao, "Towards an analytic framework for analysing the computation time of evolutionary algorithms," *Artificial Intelligence*, vol. 145, no. 1-2, pp. 59–97, 2003.
19. ——, "A study of drift analysis for estimating computation time of evolutionary algorithms," *Natural Computing*, vol. 3, no. 1, pp. 21–35, 2004.
20. M. Laumanns, L. Thiele, and E. Zitzler, "Running time analysis of evolutionary algorithms on a simplified multiobjective knapsack problem." *Natural Computing*, vol. 3, no. 1, pp. 37–51, 2004.
21. R. Kumar and N. Banerjee, "Analysis of a multiobjective evolutionary algorithm on the 0-1 knapsack problem." *Theor. Comput. Sci.*, vol. 358, no. 1, pp. 104–120, 2006.
22. Y. Zhou and J. He, "A runtime analysis of evolutionary algorithms for constrained optimization problems," *IEEE Transactions on Evolutionary Computation*, (accepted).

23. S. Martello and P. Toth, *Knapsack Problems*. Chichester: John Wiley & Sons, 1990.
24. P. C. Chu and J. E. Beasley, "A genetic algorithm for the multidimensional knapsack problem." *J. Heuristics*, vol. 4, no. 1, pp. 63–86, 1998.
25. J. He and L. S. Kang, "On the convergence rate of genetic algorithms," *Theoretical Computer Science*, vol. 229, no. 1-2, pp. 23–39, 1999.
26. J. He and X. H. Yu, "Conditions for the convergence of evolutionary algorithms," *Journal of Systems Architecture*, vol. 47, no. 7, pp. 601–612, 2001.
27. J. He and X. Yao, "Drift analysis and average time complexity of evolutionary algorithms," *Artificial Intelligence*, vol. 127, no. 1, pp. 57–85, 2001.
28. ——, "Erratum to: Drift analysis and average time complexity of evolutionary algorithms - [artificial intelligence 127 (2001) 57-85]," *Artificial Intelligence*, vol. 140, no. 1, pp. 245–248, 2002.
29. T. Jansen and I. Wegener, "The analysis of evolutionary algorithms - a proof that crossover really can help." *Algorithmica*, vol. 34, no. 1, pp. 47–66, 2002.
30. ——, "Real royal road functions–where crossover provably is essential." *Discrete Applied Mathematics*, vol. 149, no. 1-3, pp. 111–125, 2005.