Effectiveness of Penalty Function in Solving the Subset Sum Problem

Hong WANG, Zhiqiang MA and Kenji NAKAYAMA
Department of Electrical and Computer Engineering
Kanazawa University, Kanazawa 920, JAPAN
{wh,ma,nakayama}@leo.ec.t.kanazawa-u.ac.jp

ABSTRACT

In this paper we investigate the evolutionary heuristics used as approximation algorithm to the subset sum problem. We propose a graded penalty function in a fitness function of genetic algorithms to penalize an infeasible string in solving the subset sum problem. An exponential term of generation variable, t^θ, is added into the penalty function for increasing penalty generation by generation. The experiments show that the proposed penalty function is more efficient than other existing penalty functions. It is suggested that the penalty pressure is increased step by step.

Keywords: penalty function, genetic algorithm, subset sum, combinatorial optimization.

I. INTRODUCTION

The Subset Sum Problem (SSP) is a kind of constrained combinatorial optimization problem which can be used for the optimal memory management in multiple programming. The SSP is a Nondeterministic Polynomial complete (NP-complete) problem whose computational complexity is thought to increase exponentially as n increases. The n means terms number of a sequence in a NP-complete problem.

A Genetic Algorithm (GA) is a form of evolution that occurs on a computer. Genetic Algorithms (GAs) are adaptive methods which may be used to solve search and optimization problems. Genetic algorithms are probably the best known evolutionary algorithms. Their basic principles were first laid down rigorously by Holland [3], and are well described in [2] and [5].

We use a genetic algorithm described by pseudo code as shown below.

```
Algorithm GA is
    t := 0;
    initialize P(t);
    evaluate P(t);
    while not terminate P(t)
    do
        t := t + 1;
        P(t) := select P(t − 1);
        mutate P(t);
        recombine P(t);
        evaluate P(t);
    od
end GA
```

where t is generation, $P(t)$ is a population of individual.

In this paper we present the results of applying the genetic algorithm to the SSP. Unlike many traditional approaches that use domain-specific knowledge and specialized genetic operators, we make use of a graded penalty term incorporated in the fitness function of genetic algorithms. An exponential term of generation variable is added into a penalty function. We discuss influence of the exponential term on GA procedures. An random algorithm is introduced for comparing with genetic algorithms.

II. SUBSET SUM PROBLEM

Given a set $\mathbf{W}$ of positive integers $\{w_1, w_2, \cdots, w_n\}$ and a positive integer C, the subset sum problem is to find subset $\mathbf{S}$ of $\mathbf{W}$ whose sum is closest to, without exceeding, C.

In order to run a genetic algorithm on a practical problem, a suitable coding for the problem must be devised. In the SSP we assume x_i denote w_i in $\mathbf{W}$, where $x_i \in \{0,1\}$. $i = 1, 2, \cdots, n$. We define $x_i = 1$ if w_i exists in $\mathbf{S}$ and $x_i = 0$ otherwise. So the SSP can be expressed as follows:

$$Minimize \qquad C - \sum_{i=1}^{n} w_i x_i \tag{1}$$

$$Subject\ to \qquad \sum_{i=1}^{n} w_i x_i \le C \tag{2}$$

$$x_i \in \{0,1\}, \quad i = 1, 2, \cdots, n$$

III. PENALTY FUNCTION

Because there are constraints in a constrained combinatorial optimization problem, the solution space is a subset of search space. Recently, there are four methods used to handle constraints with GAs: rejection of infeasible offspring, repair algorithms [7], modified genetic operators, and penalty functions [6; 4; 9].

When infeasible offsprings have been created, these offsprings can be rejected from entering next population. This method spends a great deal of time in the evaluation and rejection of these infeasible strings. When an infeasible string has been created by an operator, special repair algorithms for that operator can be employed to restore feasibility. However, repair algorithms are problem specific, the children often do not resemble their parents, and restoring feasibility may be as difficult as the optimization problem.

0-7803-2902-3/96/$4.00

- In GAs, the method of penalty function is better than the method of "rejection of infeasible offspring".
- When penalty function is not relative to generation number ($\theta = 0$), the curve is not best.
- When $1 < \theta < +\infty$, the larger θ is, the worse curve is.
- When $\frac{1}{2} \leq \theta \leq 1$, the curve is better than others.

These results have been gotten because infeasible strings may give good genes to their offsprings.

VI. Conclusions

Genetic algorithms are effective, robust search procedure for combinatorial optimization problems. Obviously, they are more efficient than the random algorithm.

Since we only use domain-specific knowledge in the fitness function and penalty function, and do not use domain-specific knowledge in GAs, e.g. specialized genetic operators, GAs can be easily applied to a broad range of combinatorial optimization problems.

The method of penalty function is better than the method of "rejection of infeasible offspring".

It is suggested that infeasible strings will be progressively reduced activity by penalizing them because infeasible strings may give good genes to their offsprings. If the exponent of the generation in penalty function belongs to closed interval $[\frac{1}{2}, 1]$, the effect will be all the better.

Acknowledgements

We gratefully acknowledge Dr. Y. Wang and others at our laboratory for all the valuable discussions.

References

[1] T. Bäck, *A User's Guide to GENEsYs 1.0.* University of Dortmund, July 1st 1992.

[2] D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning.* Reading, MA: Addison-Wesley, 1989.

[3] J. H. Holland, *Adaptation in natural and artificial systems.* Ann Arbor: The University of Michigan Press, 1975.

[4] J. A. Joines and C. R. Houck, "On the use of non-stationary penalty funtions to solve nonlinear constrained optimization problems with GA's," in *Proceedings of the First IEEE Conference on Evolutionary Computation*, pp. 579–584, IEEE World Congress on Computational Intelligence, 1994.

[5] Z. Michalewicz, *Genetic algorithms + data structures = evolution programs. Artificial Intelligence*, New York: Springer-Verlag, 1992.

[6] A. L. Olsen, "Penalty funtions and the knapsack problem," in *Proceedings of the First IEEE Conference on Evolutionary Computation*, pp. 554–558, IEEE World Congress on Computational Intelligence, 1994.

[7] D. Orvosh and L. Davis, "Using a genetic algorithm to optimize problems with feasibility constraints," in *Proceedings of the First IEEE Conference on Evolutionary Computation*, pp. 548–552, IEEE World Congress on Computational Intelligence, 1994.

[8] H. Wang, *A Genetic Algorithm with Penalty Functions for Combinatorial Optimization Problems.* Master's thesis, Kanazawa University, Kanazawa 920, Japan, January 1996.

[9] H. Wang, Z. Ma, and K. Nakayama , "Using genetic algorithms to solve subset sum problem," in *Proceedings of the Joint Conference of Hokuriku Chapters of Institutes of Electrical Engineers*, p. 355, Sept. 1995.

When the domain-specific knowledge is added into standard genetic operators, we can get modified genetic operators for special problems. However, modified genetic operators are also problem specific.

The methods of penalty function transform a constrained problem into an unconstrained one by penalizing those strings which are infeasible. We use a penalty function to adjust the fitness of illegal individuals. The penalty function is inserted into a fitness function.

In general, evaluation function and fitness function are defined as follows:

$$f(\vec{x}) = O(\vec{x}) + \sum_{i=1}^{k} s_i P_i(\vec{x}) \tag{3}$$

$$fitness(\vec{x}) = f^{-1}(\vec{x}) \tag{4}$$

where $\vec{x} = \{x_1, x_2, \cdots, x_n\}$ is an individual. $O(\vec{x})$, $P_i(\vec{x})$ and k denote an objective function, a penalty function and the number of constraints, respectively. s_i is 0 when $\vec{x}$ is a feasible solution and s_i is 1 otherwise.

Because the SSP has a constraint, k equals 1 in the SSP.

In designing fitness functions, we make use of the following two principles.

- First, a graded penalty function used in the fitness function is an increasing function of vector $\vec{x}$ and generation number t.
  A graded penalty function makes that different vectors $(\vec{x})$ correspond to different penalty values. Moreover, when an infeasible string becomes bad and generation number is added, the value of penalty function is increased.
- Second, the best infeasible string can never be better than even the worst feasible string.
  The fitness of feasible strings is bigger than the fitness of adjusted infeasible strings. Therefore, feasible strings are selected more easily than infeasible strings in GAs.

According to these principles, we design equations for the SSP as shown below.

$$f(\vec{x}) = O(\vec{x}) + s_1 P_1(\vec{x}, t, \theta) \tag{5}$$

$$O(\vec{x}) = C - M(\vec{x}) \tag{6}$$

$$P_1(\vec{x}, t, \theta) = M(\vec{x})(1 + t^{\theta}) - C \tag{7}$$

where $M(\vec{x}) = \sum_{i=1}^{n} w_i x_i$, $s_1 \in \{0, 1\}$, t denotes the generation, θ is a positive real variable.

Therefore, $f(\vec{x}) = C - M(\vec{x})$ if $\vec{x}$ is a feasible string and $f(\vec{x}) = M(\vec{x})t^{\theta}$ otherwise.

Equation (7) is a graded penalty function which associates with generation variable t. According to equation (7), the proposed penalty function has the following features:

- When $\theta = 0$, there are no relationship between the penalty function and the generation t. The penalty function is a function of only $\vec{x}$.
- When $\theta \to +\infty$, the penalty function $P_1(\vec{x}, t, \theta) \to +\infty$. If any infeasible string in population is found, this string will not be selected to produce offsprings because $\lim_{\theta \to +\infty} P_1(\vec{x}, t, \theta) = +\infty$. Therefore, this case is equivalent to the method of "rejection of infeasible offspring".
- When $0 < \theta < +\infty$, the larger θ changes, the larger penalty function changes, that is, the smaller fitness of $\vec{x}$ (fitness($\vec{x}$)) changes.

IV. Random Algorithm

This algorithm (RA) is purely a random search which is good at exploration, but does no exploitation. It can also be used to solve the SSP. We adopt it only to compare with GA methods. The random algorithm is described by pseudo code as follows:

Algorithm RA **is**
  while total number of trial is less than given number
  do
    select $\vec{x} = (x_1, x_2, \cdots, x_n)$ randomly;
    calculate $O(\vec{x}) = C - \sum_{i=1}^{n} w_i x_i$;
    collect the $\vec{x}$ so that $O(\vec{x})$ is not negative;
    find $\vec{x}$ with which $O(\vec{x})$ is minimizes;
  od
end RA

V. Experimental Results

Effectiveness of the proposed penalty function for solving the SSP is demonstrated by the following experiments.

We use the genetic algorithm software package GENEsYs 1.0 [1] in our experiments. The parameters of the SSP are $n = 100$, $w_i = random[0, 999]$, $C = (r + \frac{1}{2}) \sum_{i=1}^{n} w_i$, the difficult degree coefficient $r = \frac{3}{16}$ [8].

In GAs, the population size is 50, the total number of generation equals 600. The last result is gotten by averaging over the 10 runs. We use the traditional genetic algorithm (TGA) which includes proportional selection scheme, 2-point crossover, standard mutation, mutation rate $p_m = 0.001$ and crossover rate $p_c = 0.6$.

In the RA, there is no concept of "generation" in the random algorithm, but in order to comparing with GAs, we assume 50 trials in the RA equal one generation in GAs. So 30000 trials in the RA correspond to 600 generations in the TGA.

In order to compare penalty functions in the TGA, we assign θ in equation (7) equals $0, \frac{1}{2}, 1, 2, 3, 5, +\infty$, respectively.

The experimental results are shown in figure 1. These curves which are shown in figure 1 from the best to the worst are "$\theta = \frac{1}{2}$", "$\theta = 1$", "$\theta = 0$", "$\theta = 2$", "$\theta = 3$", "$\theta = 5$", "$\theta = +\infty$", and "RA".

From figure 1, we have the following results:

- The traditional genetic algorithms with any θ are always better than the random algorithm.

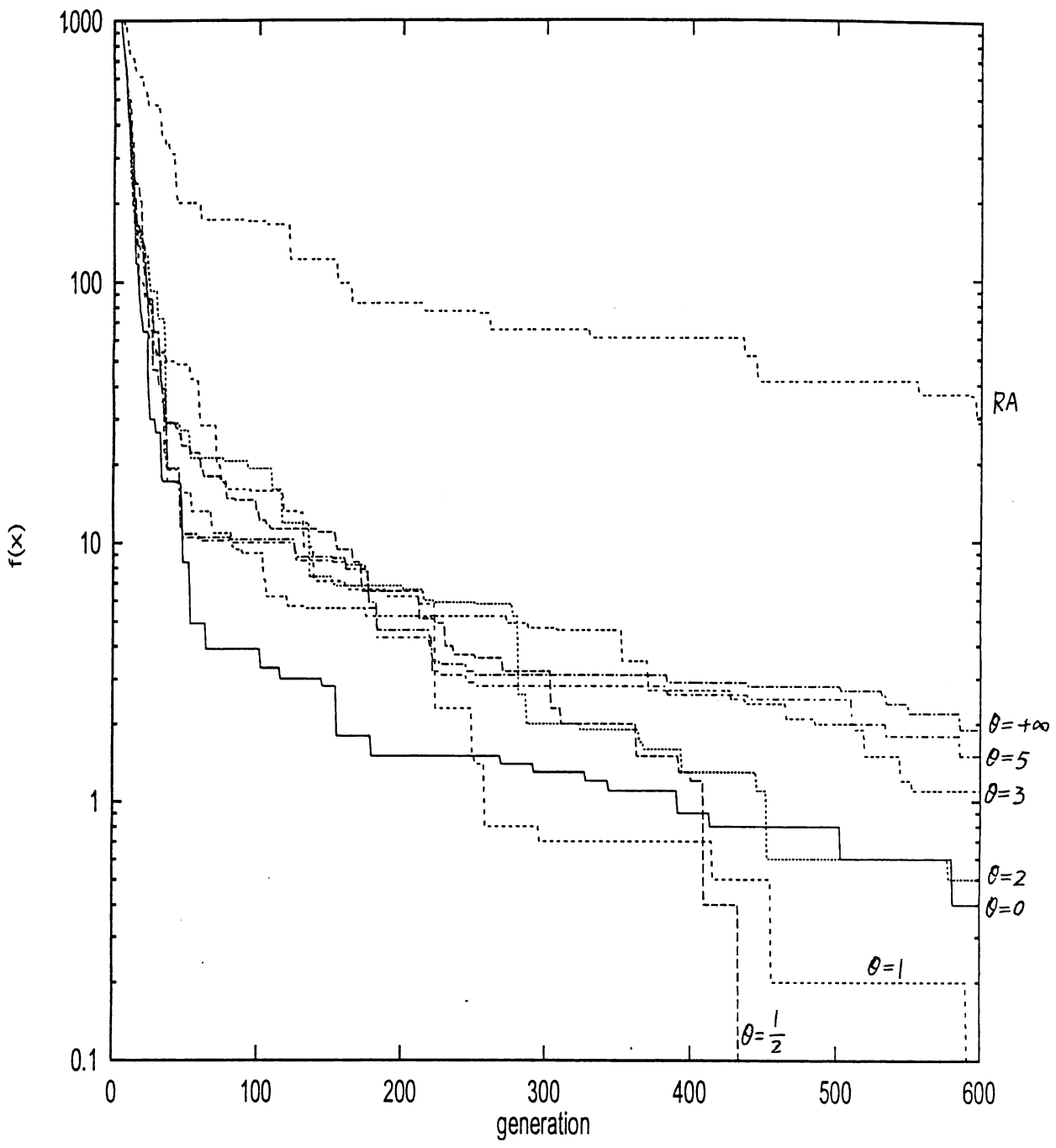

Figure 1: Comparison of θ in Penalty Functions and the RA