VICTORIA UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF COMPUTER AND MATHEMATICAL SCIENCES

Mapping, Order-independent Genes and the Knapsack Problem

Robert Hinterding

(35 COMP 8)

TECHNICAL REPORT

VICTORIA UNIVERSITY OF TECHNOLOGY
(P O BOX 14428) MELBOURNE MAIL CENTRE
MELBOURNE, VICTORIA, 3000
AUSTRALIA

TELEPHONE (03) 688 4249 / 4492
FACSIMILE (03) 688 4050

Footscray Campus

Mapping, Order-independent Genes and the Knapsack Problem

Robert Hinterding
Victoria University of Technology
PO Box 14428 MMC,
Australia 3000
Email: rhh@matilda.vut.edu.au

Abstract - **This paper uses the simple structure of the knapsack problem to study the issues of mapping and representation for genetic algorithms. Two genetic algorithms using different mappings were implemented to solve the problem. In one of these neither the order or position of genes is significant. Both of the genetic algorithms perform well on the problem, and we attribute the divergent parameter settings to the different mappings.**

1. Introduction

The knapsack problem can be described as selecting from among various items those items which are most useful to you, given that the knapsack has limited capacity. Knapsack problems have been intensively studied because of their simple structure and because they can model many classical industrial problems such as capital budgeting, cargo loading and stock cutting.[Martello & Toth 90].

Genetic Algorithms (GAs) are stochastic search methods inspired by the genetics of real populations. A population of individuals (chromosomes which decode to solutions) evolves with selection processes favouring the 'fittest' individuals. Classical genetic algorithms [Goldberg 89] use a binary alphabet and fixed length chromosomes (binary strings). Binary strings are theoretically tractable and form the basis of the theory underlying GAs. However GAs using larger alphabets and variable length chromosomes have been developed.

This paper studies the issues of mapping and representation associated with GAs by using two different GAs to solve the knapsack problem. As more direct methods already exist to solve the knapsack problem in polynomial time it's not the aim of this study to compete with these. Instead we use the knapsack problem as a vehicle to investigate effects of different mappings on the behaviour and the efficiency of GAs. The motivation was to further explore GAs where neither the position or ordering of genes in the chromosome is significant. Such GAs were used to solve the bin packing problem [Falkenauer & Delchambre 92] and stock cutting problems [Hinterding & Juliff 93].

Two GAs using different mappings were implemented to solve the Knapsack Problem. One uses variable length chromosomes, an encoder to generate valid (legal) knapsacks as chromosomes with new crossover and mutation operators. The other uses a fixed length chromosome with Uniform Order Based crossover, swap mutation and a decoder to map from the chromosome to legal knapsacks.

Both GAs performs well on the problems. The parameter settings for the two GAs differ markedly and this can be attributed to the different mappings. The GA where neither the position or order of the genes in the chromosome are significant is shown to be viable, although the current theory of schemata does not cover this type of GA.

2. The Knapsack Problem

The knapsack problem under consideration is the 0-1 Knapsack problem which is classified as NP-hard [Garey & Johnson 79]. The 0-1 knapsack problem has been solved using a number of mathematical techniques including dynamic programming, and branch and bound algorithms.

The 0-1 Knapsack Problem (KP) is described as:

Given a set of n items and a knapsack, with:
p_j = profit of item j,
w_j = weight of item j,
c = capacity of the knapsack,

select a subset of the items so as to

$$\text{maximise } z = \sum_{j=1}^{n} p_j x_j$$

$$\text{subject to } \sum_{j=1}^{n} w_j x_j \leq c,$$

$$x_j = 0 \text{ or } 1,\ j \in N = \{1..n\},$$

$$\text{where } x_j = \begin{cases} 1 \text{ if item } j \text{ is selected;} \\ 0 \text{ otherwise.} \end{cases}$$

The knapsack problem was chosen as a problem with the following features:

- The solution is not numerical.
- The alphabet is larger than the number of genes.
- The number of items in a solution is not fixed.
- The order of the items in a solution is of no significance.

A problem where the structure and the restraints usually associated with GAs were absent was desired. The knapsack problem fits these requirements very well. The problem is to select the most valuable set of items from a larger set, satisfying a simple capacity constraint.

Other researchers have solved the 0-1 knapsack problem using GAs. Watannabe et al [Watannabe 92], looks only

at small problems (16 and 24 items) using a bit string representation. Gordon at al [Gordon & Whitley 93] [Gordon, Bohn & Whitley 93] use a much larger range of problem sizes (20 to 1000 items) comparing the results of different serial and parallel GA models with the results obtained from more direct methods such as branch-and-bound and depth-first search. They also use a bit string representation and report disappointing results compared to the more direct methods.

3. Representing the Problem.

Two different representations were used to solve the knapsack problem using GAs. The Selection based GA which uses a direct representation and dense encoding, and an Order based GA which uses an indirect representation. This GA uses an encoding with genes in the chromosome which are not expressed in the phenotype.

The following example knapsack problem is used in this section. The capacity of the knapsack is 100. The numbers used in the text refer to the item numbers in the example problem.

Item No.	1	2	3	4	5	6	7
Weight	40	50	30	10	10	40	30
Profit	40	60	10	10	3	20	60

Table1: Example knapsack problem.

3.1. Mapping using a Selection based Genetic Algorithm

The most direct mapping is to let the chromosomes represent legal knapsacks directly. The genes in the chromosome will represent the items in the knapsack, and as the number of items in a legal knapsack is not fixed, we need to use a variable length chromosome. The order of the items in a legal knapsack has no meaning, so the order of the genes in our chromosome will also have no meaning. This means that we don't have alleles in our chromosome. Hence (4 5 7 3) and (2 6 5) are both legal knapsacks and both (2 6 5), (6 2 5) represent the same legal knapsack.

```
Repeat
        Search left to right for first item that does
        not violate the capacity constraint.
        If item found
                add to knapsack,
        else
                terminate algorithm
```

Figure 1: First fit algorithm

As we use a direct representation, we do not need a decoder to map from the genotype to the phenotype. To generate the initial population we need an encoder to generate random legal knapsacks. The encoder we use is a first fit algorithm (see fig. 1), which given an ordering of items will select in order those that meet the capacity constraint and reject items resulting in a constraint violation. The initial population is generated from random orderings of the items using the first fit algorithm.

3.1.1. Reproduction

The reproduction operators we use can not be any of the standard ones as these rely on the position or the ordering of the genes in the chromosome being significant.

3.1.1.1. Crossover

The crossover operator used is the Injection crossover, which is a simplification of Falkenauers' Grouping crossover(BPCX) [Falkenauer & Delchambre 92]. This crossover will work with chromosomes of different lengths, and does not depend on any ordering of the genes. It was designed so that the child can inherit meaningful information from both its parents. In this case it is the selection of items the parents have. It also behaves as expected; if the parents are identical they will produce a child that is the same as the parents.

The Injection crossover works in the following way: we randomly choose an insertion point in parent 1 and a segment in parent 2. The child is constructed by first copying into it the genes from parent 1 up to the insertion point. Then we copy the segment from parent 2 into the child, and lastly we copy the genes after the insertion point till the end of the chromosome. Note that when copying genes into the child we do not add genes that already exist in the child. The proto-child chromosome which we have now constructed may contain too many items(in term of the capacity of the knapsack). So we now apply a fix-up step by applying the first fit algorithm to the proto-child chromosome to produce the child chromosome, which is a legal knapsack.

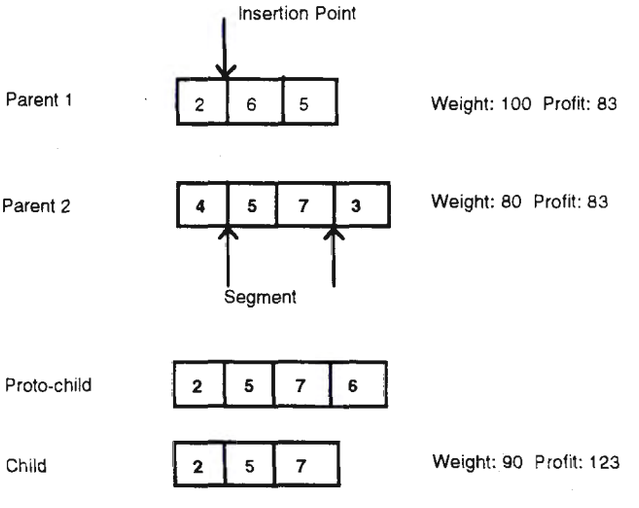

Figure 2: Injection Crossover

3.1.1.2. Mutation

Mutation is based on Faulkenauer's group mutation operator [Falkenauer & Delchambre 92]. A number of genes are chosen at random and these are deleted. We then append the items in random order that were not present in the chromosome, and use the first fit algorithm to generate a chromosome which is a legal knapsack. The

purpose of mutation in this case is to bring in new items into the chromosome.

3.2. Mapping using an Order based Genetic Algorithm

Here we let the chromosomes represent orderings of all the items. We use a decoder which utilises the first fit algorithm to generate a legal knapsack from the ordering of items. Hence a chromosome of (1 6 4 7 3 2 5) would decode to the knapsack (1 6 4 5) which has a profit of 73; items 7, 3 and 2 are rejected because their individual inclusion would result in constraint violation (see table 2). Here the genotype represents an ordering of the items, and the phenotype is a legal knapsack. We can use an order based chromosome and choose suitable crossover and mutation operators. Using this method we convert what is a selection problem into an ordering problem.

Knapsack	Items left	Weight	Profit
empty	1 6 4 7 3 2 5	0	0
1	6 4 7 3 2 5	40	40
1 6	4 7 3 2 5	80	60
1 6 4	7 3 2 5	90	70
1 6 4 5	7 3 2	100	73

Table 2: First fit decoding

One of the features of this representation is that not all the genes in the chromosome are represented in the solution. That is, there are genes in the genotype that are unexpressed in the phenotype, and the number of such genes will depend on the capacity of the knapsack and the items used in the knapsack.

The crossover operator used is Uniform Order Based crossover[Davis 91], and mutation is the swapping of two randomly chosen genes in the chromosome.

3.3. The Fitness Function

The fitness function that we maximise is the profit of the items in the knapsack divided by the profit of all the items. This gives us a value in the range 0 to 1, but it will always be less than 1 for this example, and depend on the capacity of the knapsack. Hence the smaller the capacity of the knapsack the smaller the maximum fitness will be.

4. The GA

The basic Genetic Algorithm used in both cases is a steady-state GA based on the description of OOGA in Davis [Davis 91]. It was developed using Smalltalk/V for Windows and then translated into C++. The following parameters can be set:

- Population Size - set the size of the population.
- Replacement Rate - set the percentage of the population that will be replaced by reproduction in one generation.
- Number of Evaluations - set the number of evaluations for the run. We use evaluations rather than generations so that we can compare between runs where the population size and replacement rate are different.
- Mutation Rate - set the percentage of the replacement population that will be produced by mutation. The remainder of the replacement population will be produced by crossover.
- Allow Duplicates - set a flag to allow or disallow duplicates to exist in the population. If duplicates are not allowed, any duplicates produced by reproduction are discarded while they still count as an evaluation. We determine whether two chromosome are the same by comparing their genotypes.

In the Genetic Algorithm used, a new chromosome is produced either by crossover or mutation but not both. This was done so that the separate effects of these reproduction operators could be determined.

The mutation rate for the order based GA is the mutation rate of the chromosomes, and mutation will swap two genes in the chromosome. The mutation rate for the selection GA is the mutation rate of the genes. It was felt that this reflected better what happens in nature, in that we would expect more genes to mutate in a long chromosome than in a short one, rather than the same number of chromosomes mutating in a chromosome regardless of its length. This was done by using a random generator with a Poisson distribution. We set λ to be the gene mutation rate multiplied by the length of the chromosome. The values returned are the number of genes in the chromosome which are to mutate. A value of zero indicates that mutation does not occur and crossover is to be used to generate the new chromosome. Further research needs to be done on this, but the results indicate that using this method produces superior performance.

5. Results

For this study we generated a 50 item and a 100 item problem. We used three different knapsack capacities expressed as a percentage of the total weight of the items for each of the problems. The global maximum fitness for each of the problems is summarised in the following table:

Capacity/ Problem	25%	50%	75%
50	0.64781	0.861296	0.960133
100	0.57001	0.81889	0.959883

Table 3: Problem configurations and global maximums

For the 50 item problem the GA was run for 10,000 evaluations, and for the 100 item problem it was run for 20,000 evaluations. All the results were produced by averaging the data of 20 runs.

Replacement Rate: this was varied from 15% to 95% and a replacement rate of 25% was found to give best results. All the results shown were run with a replacement rate of 25%.

Duplicates Allowed: It was found that not allowing duplicates in the population significantly improved the results. All results shown are from runs where duplicates were not allowed.

A range of values for the other parameters was tried in order to find the values which maximised the efficiency of the GA. The results are summarised in tables 4 and 5. Comparative performance of the GAs on the 100 items 75% capacity problem is shown in Figure 3. The Selection based GA did not find the global maximum for the 100 item problem, however the values found are within 99.8% of the global maximum.

Prob/Cap	Pop size	Gene mutation	Best fitness	Evals needed
50/25	50	2	0.64781	6,000
50/50	50	2	0.861296	5,000
50/75	50	2	0.96013	2,000
100/25	50	1.8	0.569823	20,000
100/50	50	1.2	0.817624	20,000
100/75	50	1	0.958861	20,000

Table 3: Selection Based GA results

Prob/Cap	Pop size	Chrom. mutation	Best fitness	Evals needed
50/25	200	0	0.64781	4,000
50/50	150	0	0.861296	2,000
50/75	150	0	0.96013	2,000
100/25	700	0	0.57001	19,000
100/50	700	0	0.81889	17,000
100/75	600	0	0.959883	15,000

Table 4: Order Based GA results

The mutation rates for the selection based GA are observed to be very high. Table 5 relates the gene mutation rates used to chromosome mutation rates. For the 50 item problem, only integer values for the gene mutation rates were used.

Prob/Cap	Gene mutation	Av chrom size	Poisson means	Chrom. mutation
50/25	2	16	0.31	27%
50/50	2	27	0.55	42%
50/75	2	38	0.77	54%
100/25	1.8	29	0.51	40%
100/50	1.2	52	0.55	46%
100/75	1	77	0.77	53%

Table 5: Selection Based GA mutation

Mutation rates for the order based GA showed some unexpected trends. For population sizes found to be less than optimum, the mutation rate had to be very high to get the best results (up to 50%). While with population sizes greater or equal to the optimum, mutation rates less than 10% did not degrade the efficiency of the GA, but did not improve it either. Table 4 shows the maximum fitness reached after 20,000 evaluations when no mutation is used for a range of population sizes for the order based GA on the 100 item 50% capacity problem.

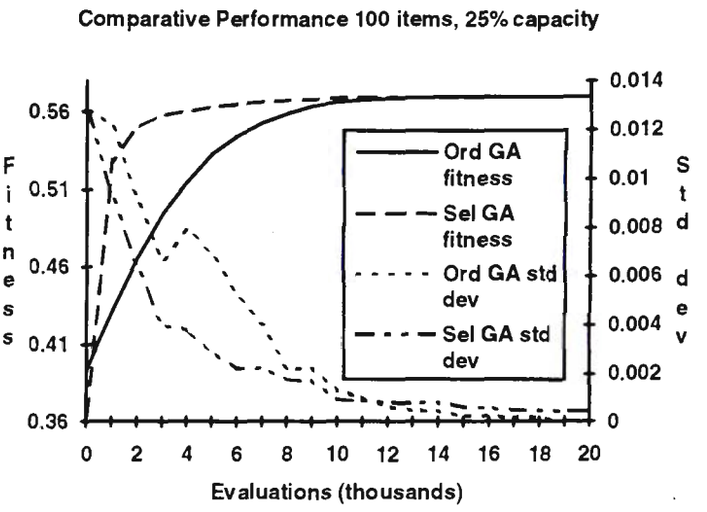

Figure 3: Comparative results of the GAs

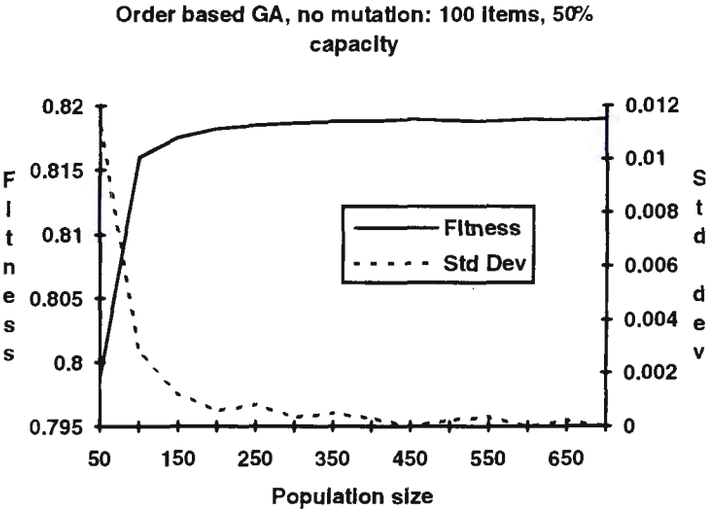

Figure 4: Population size for Order based GA

The time taken for the order based GA running on a Sparc 10 workstation was 4 seconds for the 50 item problem to 2,000 evaluations and 110 seconds for the 100 item problem to 17,000 evaluations.

6. Discussion

Both the GAs performed well, while the order based GA was better on the 100 item problems. The difference in the population size and mutation rate for the two GAs is very marked.

The selection based GA uses a high mutation rate and small population. A small population with a set number of evaluations will result in more generations being run, than for a larger population. As the chromosomes only contain items that are in the knapsack, mutation is essential to bring in new items and a larger number of generations is needed to explore selections using these items. As the mutation rate is so high, further research is needed to see the effect of allowing both mutation and crossover to produce new chromosomes. This may increase the GAs efficiency.

The order based GA uses a large population size and does not need mutation for populations sufficiently large. When the population is not large enough it converges very rapidly on to a phenotype and all members of the population will decode to the same phenotype. When this is the case, mutation rates high enough to stop convergence to the phenotype are also too disruptive for the GA to find the global maximum. What appears to happen here is that the many to one mapping of genotype to phenotype effectively reduces the diversity of the population. Hence a large population size must be used to increase the population diversity. When a population size larger than the optimum is used the efficiency of the GA is reduced. Also if sufficient diversity is present no mutation is needed as Uniform Order Based crossover can generate the missing orderings.

Conclusions

We have shown that the mapping chosen for a GA can dramatically effect the parameter settings of the GA for best efficiency. We have also shown that a GA (the selection based GA) which uses a large alphabet, has variable length chromosomes and does not have alleles compares favourably with a more traditional GA (the order based GA). In the selection based GA, the size of the chromosomes decreases as the capacity decreases for the same sized problem, and we consequently need a higher mutation rate to explore the solution space. Also as neither the order or position of genes is significant the current theory of schemata does not hold in this case.

Acknowledgments

The author wishes to thank Emmanuel Falkenaur for interesting and useful discussions, and Clement Leung for his help and suggestions with writing the paper.

References

[Davis 91] Davis, L., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold

[Garey & Johnson 79] Garey, Micheal R. and Johnson David S., *Computers and Intractability - A Guide to the Theory of NP-completeness*, W.H. Freman Co., San Francisco.

[Goldberg 89] Goldberg, David E., *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley

[Gordon & Whitley 93] Gordon, V. Scott and Whitley, Darrell, *Serial and Parallel Genetic Algorithms as Functions Optimizers*, Proceedings of ICGA-93, pp 177-183, Urbana-Champaing 1993

[Gordon, Bohn & Whitley 93] Gordon, V. S., Bohm, A. P. W. and Whitely, D., *A note on the performance of Genetic Algorithms on Zero-One Knapsack Problems*, Technical Report CS-93-108, Department of Computer Science, Colorado State University.

[Hinterding & Juliff 93] Hinterding, Robert and Juliff, Kate, *A Genetic Algorithm for Stock Cutting: An exploration of Mapping Schemes*, Technical Report 24COMP3, Department of Computer and Mathematical Sciences, Victoria University of Technology, Victoria Australia.

[Falkenauer & Delchambre 92] Falkenauer, E.A, *A Genetic Algorithm for Bin Packing and Line Balancing*, Proceedings of 1992 IEEE International Conference on Robotics and Automation(RA92), pp. 1186-1193, Nice 1992.

[Martello & Toth 90] Martello, S. and Toth, P., *Knapsack Problems*, John Wiley & Sons 1990, p 2,13.

[Watannabe et al 92] Watannabe, K., Ikeda, Y., Matsuo, S. and Tsuji, T. *Improvement of the Genetic Algorithm and its Application*, Memoirs - Faculty of Engineering Fuki University 1992, Vol 40 Issue 1.