Nature Inspired Genetic Algorithms for Hard Packing Problems

Philipp Rohlfshagen & John A. Bullinaria

School of Computer Science
University of Birmingham
Birmingham, B15 2TT
UK

{P.Rohlfshagen, J.A.Bullinaria}@cs.bham.ac.uk

Abstract

This paper presents two novel genetic algorithms (GAs) for hard industrially relevant packing problems. The design of both algorithms is inspired by aspects of molecular genetics, in particular, the modular exon-intron structure of eukaryotic genes. Two representative packing problems are used to test the utility of the proposed approach: the bin packing problem (BPP) and the multiple knapsack problem (MKP). The algorithm for the BPP, the exon shuffling GA (ESGA), is a steady-state GA with a sophisticated crossover operator that makes maximum use of the principle of natural selection to evolve feasible solutions with no explicit verification of constraint violations. The second algorithm, the Exonic GA (ExGA), implements an RNA inspired adaptive repair function necessary for the highly constrained MKP. Three different variants of this algorithm are presented and compared, which evolve a partial ordering of items using a segmented encoding that is utilised in the repair of infeasible solutions. All algorithms are tested on a range of benchmark problems, and the results indicate a very high degree of accuracy and reliability compared to other approaches in the literature.

Keywords: Genetic Algorithms; Bin Packing Problem; Multiple Knapsack Problem; Repair Algorithms; Decoders; Exon Shuffling; RNA Editing.

To appear in: Annals of Operations Research.

1 Introduction

This paper is concerned with developing improved algorithms for packing tasks, which are an important class of problems that occur frequently in many industrial contexts. In general, these tasks involve the placement of items that have some attribute (e.g., weight, shape) into some container (e.g., bin or sheet). The objective is to minimise some notion of cost, while obeying the problem's constraints (e.g., capacity or area). In this paper we present, and analyse the performance of, two novel nature-inspired algorithms for two well defined packing problems: the Bin Packing Problem (BPP) and the Multiple Knapsack Problem (MKP). These two problems were chosen because (a) they are representative of the key issues involved in packing problems more generally, (b) they involve different but widely used natural encodings, (c) instances exist that current algorithms find "hard" or impossible, and (d) they are the most widely studied packing problems in the field of Evolutionary Computation (EC). Although the BPP and MKP are conceptually similar, and some derivations of the BPP may be reduced to the MKP (as, for example, discussed by Bein, Correa, and Han (2008)), there are also important differences. This paper shows how similar nature-inspired techniques may be employed to solve these two problems, and how the differences between them demand subtle changes in the design of the algorithms.

The algorithms presented here are based upon Genetic Algorithms (GAs), which have become a popular choice of algorithm for numerous difficult optimisation problems where conventional methods tend to fail (Holland 1975; Mitchell 1996). GAs maintain a population of potential solutions to the problem of interest and evolve solutions of increasing quality by means of selection, crossover and mutation. GAs have traditionally been modelled on the field of population genetics (Daida et al. 1999), with individuals in the algorithm's population representing "chromosomes with many loci and few alleles per locus" (Holland 1975, p.71). Although GAs were never meant to be accurate models of biological systems, we believe it may be beneficial to consider possible shortcomings from a biological point of view, in order to yield improvements over the algorithm's traditional design. The design of the algorithms presented in this paper are examples of such a nature inspired undertaking. In particular, inspiration is drawn from the modular exon-intron structure of eukaryotic genes and the exploitation of this modularity by means of cellular processes. This modularity (or segmentation) here takes the form of useful coherent groupings of items, such as bins in the BPP, that may facilitate the discovery of better solutions.

The first algorithm presented here, ESGA, exploits the intrinsic structure of the BPP, drawing meaningful parallels from the theory of exon shuffling. This involves developing a sophisticated crossover operator that not only generates offspring of high quality, but also over time eliminates sub-solutions that violate any constraints. The second algorithm, ExGA, employs an RNA inspired adaptive repair function to restrict the search to the feasible regions of the search space at all times. The repair is based upon a segmented encoding that allows for a partial ordering of problem variables to emerge. This order is subsequently utilised in a two-phase repair function. Finally, the cost of repair is taken into consideration and two variants of the ExGA are proposed that are computationally more efficient without a significant loss in performance.

The remainder of this paper is structured as follows: First, section 2 presents a general discussion of nature inspired design and a brief introduction to the relevant aspects of genetics. Then the ESGA is discussed in section 3, followed by the presentation of ExGA in section 4, with performance analysis of both. Finally, the paper ends in section 5 with some conclusions, a comparison of the algorithms and a discussion of future work in this area.

2 Nature Inspired Design

Canonical GAs are highly effective yet suffer from numerous technical problems, such as premature convergence, that may prevent them from locating global optima in difficult search spaces. Numerous extensions have been suggested in the past to address these issues, and a considerable amount of effort has

focussed on the implementation of natural processes. Such inspiration from natural systems has been, and remains, the single most influential factor in the field of EC. Spector (2003, p. 11), for example, points out that "it posits that genetic programming practice (including both applications and technique enhancements) is moving toward biology and that it should continue to do so". Natural systems have evolved for considerable lengths of time, and it makes good sense to copy what has emerged from that process. On the other hand, one should not take nature inspired design as the ultimate answer to all problems. Borrowing from nature is not without difficulties, and Freeland (2003) describes the delicate balance between a healthy level of abstraction and a blind 'biological envy' (reciting from a talk given by Goldberg). The literature reveals numerous examples where the rejection of biological details has been advantageous (e.g., Freeland (2003)).

This leads to the conclusion that finding the right abstraction is immensely difficult. Atlan (2003, p. 2) recognises the possible frustration one might encounter in mimicking natural processes as "nature has not built these machines [cells] according to some blueprint designed on purpose by an engineer. This is where physicists, engineers and computer scientists, tempted by the adventure of taking up this challenge, must learn from what biologists have accumulated as partial, often paradoxical, observations. The biological structure is not necessarily the one that a clever engineer would have chosen in order to perform the same function." Clearly, artificial evolution is in no way restricted by physical or biochemical laws. As Freeland (2003, p 310) notes: "A luxury of EC is to bend fundamental rules of evolution beyond anything biologically plausible, and thus to answer questions where biologists have assumptions". This is similar to the vision of Langton (1992, p. 40) that artificial life will "lead us not only to, but quite often beyond, known biological phenomena: beyond life-as-we-know-it into the realm of life-as-it-could-be". It seems logical then to also consider non-biological (e.g., statistical) components, given the inherent differences between any artificial problem space and the ecological niche of an organism.

The concept of nature inspired design thus requires some caution. It is important to distinguish between computer simulations and heuristics. Computer simulations aim to mimic natural processes as closely as possible to gain a better understanding of their underlying principles. Their core objective is biological plausibility, and all details of the natural system to be modelled need to be captured to some degree. The sole concern of heuristics, on the other hand, is the performance in accomplishing a certain task, and one has to be much more selective when it comes to the implementation of natural processes. In the case of nature inspired approaches, it is of no significance whether the final abstraction bears recognisable similarities with the system it emulates. Matters such as biological plausibility are not of primary interest. In the end, the real efficacy of nature inspired design stems from its ability to provoke a thought process that may lead to an efficient design rather than the relentless imitation of detail. This concept is very much evident in the approaches presented here, which have been inspired by selected aspects of molecular genetics (which are discussed below), yet result in final designs that diverge greatly from their natural counterparts.

2.1 A 'New' Direction

The canonical (binary) GA (cGA) is loosely based upon the principles of population genetics, a field of evolutionary biology concerned primarily with the distribution of alleles within a population of organisms under selection pressure (Holland 1975; Mitchell 1996). Each individual encoding corresponds to a chromosome composed exclusively of artificial genes that may be in one of two states. These simple artificial genes correspond most closely to those of prokaryotes. Prokaryotes (bacteria and archaea) are organisms whose cells lack a proper nucleus and whose DNA is usually relatively condensed and simple. Eukaryotes (protists, fungi, plants and animals), on the other hand, have a more sophisticated cellular structure that allows for numerous regulatory and structural elements to occur in their DNA. Eukaryotic genes contain, for example, non-coding segments, which, amongst other attributes, allow a single gene to produce a variety of different proteins depending on different factors such as cell type.

The emergence of eukaryotes some 2.5 billion years ago is seen by many as one of the most significant events in evolutionary history (Maynard Smith and Szathmãry 1999).

There are numerous examples in the literature, most notably from recent years, that depart from the simplistic '2-allele' model in order to develop more sophisticated artificial genes that more closely resemble the genes of eukaryotes. This undertaking is primarily driven by recent advances in genetics, such as the genome sequencing projects, that expose a wealth of different cellular processes and their attributes. This is encouragement to design evolutionary algorithms (EAs) from a molecular perspective, and to draw attention to those areas of molecular genetics that have so far been largely ignored by the EC community.

Of relevance here is the 'central dogma of genetics' which may be summarised as follows. The carrier of hereditary information in almost all organisms is DNA which contains well defined regions, known as genes, that encode information for the production of functional units, such as proteins. These sections of DNA are expressed as follows: DNA is first transcribed to RNA which is subsequently altered by the cellular machinery: unlike prokaryotic genes, eukaryotic genes are composed of not only coding (exons) but also non-coding (introns) segments. RNA processing removes the introns to yield a continuous strand of protein coding RNA which is finally translated using a mapping known as the genetic code. The genetic code, which minimises the phenotypic variations caused by single-point mutations, maps any three nucleotides to a specific amino acid, the building blocks of proteins. The modular exon-intron structure of eukaryotic genes is 'exploited' by numerous processes, two of which are exon shuffling and RNA editing, that have inspired the design of the two algorithms presented here.

2.2 Exon Shuffling

The phenomenon of interrupted genes has been widely analysed and discussed ever since the discovery of the first introns. Roy (2003) provides an excellent review of the *Exon Theory of Genes*, the most important points of which are summarised here. Blake (1978) proposed that interrupted genes are essentially patched together from exons that code for simple protein structures, and Gilbert (1978) observed that introns could serve as buffers that allow the recombination of exons to create new protein products. Blake also proposed that this mechanism would be most efficient if the exons corresponded to independent units that determine discrete characteristics of a protein. Similarly, the "exon shuffling hypothesis" proposed by Gilbert views the evolution of genes as the recombination of independent units (exons) that code for independent protein structures. Numerous genes have been found where exons do indeed correspond to independent protein domains, and exon shuffling is likely to have played a vital role in the emergence of complex genes and other existing phenomena such as multi-cellularity (Patthy 2003; Kolkman and Stemmer 2001). This concept is evident in our ESGA algorithm that attempts to exploit the intrinsic structure of the BPP by means of an encoding that consists of well-defined sub-solutions that may be recombined in different ways to yield complete optimal solutions.

2.3 RNA Editing

RNA editing is the targeted modification of nucleotides in exons, often guided by an adjacent intron which functions as a template. In one scenario, RNA editing takes place on double stranded RNA segments which are formed by folding a downstream intron back onto the preceding exon (Higuchi et al. 1993). The intron contains the exon-specific editing information and Bass (1997) notes that the 3' and 5' untranslated regions may offer similar advantages. Interestingly, some types of DNA require RNA editing to remove specific sequences of nucleotides that would otherwise prevent the production of active proteins (Herbet and Rich 1999). In other words, RNA editing serves as a repair mechanism that restores protein synthesis which would otherwise be prevented. The repair of unfeasible 'encodings' (RNA in this case) by means of selective modification seems highly relevant to the design of GAs for

constrained optimisation problems. Here, our ExGA algorithm employs such a repair mechanism that is guided by an underlying encoding that resembles the structure of an interrupted gene.

3 The Exon Shuffling Genetic Algorithm (ESGA)

The design of the ESGA originates from the similarities between the one-dimensional BPP and the modular structure of eukaryotic genes in general, and the theory of exon shuffling in particular (Rohlfshagen and Bullinaria 2007).

3.1 The Bin Packing Problem (BPP)

The BPP has many industrial applications and is found frequently in real world scenarios such as vehicle loading. The BPP is very simple to state, yet is NP-hard (Coffman, Garey, and Johnson 1978). The objective is to fit a fixed number of items, of different weights, into the fewest possible number of bins, each of which has limited capacity c. More formally, given a set $\mathbb{U}$ of n items with labels $l = 1, ..., n$ and weights $\{w_l\}$, we need to find a disjunctive partition $B = \{B_1, ..., B_{|B|}\}$ of $\mathbb{U}$, representing the contents of $|B|$ bins, corresponding to

$$\min(|B|) \quad | \quad \sum_{l \in B_i} w_l \leq c \quad \forall B_i \in B \tag{1}$$

In other words, the number of bins required to contain all items in $\mathbb{U}$ has to be minimised while none of the bins may be overfilled. We studied three sets of benchmark problems that may be found online at `www.wiwi.uni-jena.de/Entscheidung/binpp/`, with attributes as follows:

- **Set 1** contains 20 randomly generated instances for each of 36 instance classes
  - n : 50, 100, 200, 500
  - c : 100, 120, 150
  - w_l : from [1,100], [20,100], [30,100] for $l = 1, 2, \ldots, n$
- **Set 2** contains 10 randomly generated instances for each of 48 instance classes
  - n : 50, 100, 200, 500
  - c : 1000
  - w_{avg} : c/3, c/5, c/7, c/9
  - d : 20%, 50%, 90%
- **Set 3** contains 10 particularly hard instances
  - n : 200
  - c : 100000
  - w_l : from [20000,35000] for $l = 1, 2, \ldots, n$

where n is the number of items, c is the capacity of the bins, w_l are the weights of each item, w_{avg} is the desired average weight of the items, and d sets the maximal deviation of individual values w_l from w_{avg}. These values specify the generation of the random benchmark instances. For example, if an instance of Set 2 has $c = 1000$, $w_{avg} = c/5$ and $d = 20\%$, the n weights w_l are chosen randomly from the uniform distribution $[(1000/5) * (1 - 0.2), (1000/5) * (1 + 0.2)] = [160, 240]$.

3.2 Related Work

There are numerous evolutionary approaches for the BPP in the literature. Interestingly, the majority of these approaches are hybrid, combining GAs with problem specific heuristics. It is now well established that evolutionary approaches work very well in combination with problem specific heuristics and the review of relevant literature confirms this. This section will therefore briefly overview a range of relevant heuristics, followed by a discussion of GAs for the BPP. It should be noted that due to the vast number of publications addressing the BPP, this review is necessarily restricted to the relevant key ideas.

The classic heuristics are the first fit decreasing (FFD) and the best fit decreasing (BFD) algorithms. Both of these approaches consider all items in decreasing order of their weights, and then place them systematically into the bins. The worst case result for FFD and BFD is $(11/9)opt + 4$, where opt represents the optimal number of bins (Coffman, Garey, and Johnson 1978). As pointed out by Gupta and Ho (1999), the performance of both heuristics deteriorates when the optimal solution requires the majority of bins to be filled to (near) the maximum degree possible. (These are the kind of "hard" problems this paper aims to address.) They therefore suggested a new heuristic, called minimum bin slack (MBS), to overcome this deficiency. MBS is naturally bin-focused, with any given bin filled to the maximum degree before the next bin is considered. Backtracking may be used to escape from local optima. Fleszar and Hindi (2002) have suggested several variations of MBS, one of which is MBS'. This technique only differs from MBS in the use of an initialisation procedure that speeds up the algorithm. In their study, they found that amongst several suggested approaches, a combination of heuristics gave the overall best performance when tested on a large set of benchmark problems. The application of MBS' followed by a variable neighbourhood search proved an effective combination. Alvim et al. (2004) also suggested the use of a hybrid heuristic that incorporates many different techniques. Their approach, HI_BP, is probably the most complex one, but it also produces the best results of all reviewed techniques (see Table 2). However, HI_BP is rather difficult to implement and relies crucially upon numerous parameters that need to be decided upon prior to execution.

GAs require the problem to be encoded in such a way that allows the variation operators to create useful and efficient neighbourhood structures. The most intuitive encoding for the BPP is a simple permutation of integers. The solution may subsequently be constructed from the encoding by scanning the permutation from one end to the other, fitting items into each bin while possible, before starting a new bin. This ensures that all the solutions are legal, and also allows the use of well established crossover and mutation operators that work for any permutation based encoding (such as for the travelling salesman problem). However, as was pointed out by Falkenauer (1996), this approach has a very high level of redundancy. In fact, it is clear that any permutation of items in any one bin will encode an identical solution. For example, if a combination of 5 items fit into one bin, there are $5! = 120$ different ways to encode the same solution, and multiplying such numbers across all the bins quickly leads to enormous redundancies. Falkenauer (1996) consequently suggested the idea of employing a group encoding, whereby only the group membership of an item matters and not its position within the group. However, a complicated crossover operation is then needed to ensure that the offspring are complete and do not contain any duplicate items. Duplicates need to be deleted, and missing items have to be re-inserted using a problem specific heuristic. The algorithm suggested by Falkenauer (1996) implemented a concept based on the dominance criterion due to Silvano and Paolo (1990).

The group based encoding has also been used by Iima and Yakawa (2003), who suggested a crossover operator that makes use of MBS' and a first-fit heuristic. Their crossover operator transmits some of the parental segments to the offspring in their entirety, and the remaining items are added using the aforementioned heuristics. Explicit care needs to be taken to ensure that no grouping exceeds any of the bins' capacities. They compared their GA against two non-evolutionary methods and concluded that their GA produced results of greater accuracy. Another hybrid GA, due to Kao and Lin (1992), utilised simulated annealing to improve its performance. The design of that algorithm was driven by the stochasticity of both GAs and simulated annealing, which was believed to overcome some of the

shortcomings of deterministic approaches such as FFD or BFD. There are numerous further evolutionary approaches that have not been considered here as they are either tailored to specific instances of the BPP, or designed for variants of the one-dimensional BPP (e.g., the two-dimensional BPP).

3.3 The Algorithm

The discussion of the BPP and the review of related work has shown that this problem has an inherently modular composition. This is particularly evident in the success of methods that either adopt a bin-focused view or a group-based encoding. This allows the exploitation of biologically inspired processes that are directly linked to the modularity of eukaryotic genes. A highly abstract interpretation of the theory of exon shuffling views this process as a modular bottom-up approach that incrementally leads to proteins of increasing complexity. Exons, often synonymous with independent protein domains, are combined in any perceivable way to create novel proteins of advanced functionality. The key idea is to have modules that are known to work, and a process that allows the combination of these modules in a meaningful way. This is the key principle underlying the design of ESGA.

The general evolutionary framework employed by ESGA is that of a steady-state GA: two parents are selected from the population and crossed over to produce a single offspring. A binary tournament determines the placement of the offspring into the current population. Each individual in the population P represents a partition of the set $\mathbb{U}$ of n items, with labels $l = 1, ..., n$ and weights $\{w_l\}$, into $|B|$ disjoint sets $B = \{B_1, \ldots, B_{|B|}\}$, where each bin $B_i = \{b_{i1}, \ldots, b_{im_i}\}$ contains m_i items with labels $b_{ij} \in \{1, ..., n\}$. Each such B must satisfy: $\bigcup B_i = \mathbb{U}$ and $B_i \cap B_j = \emptyset, \forall i, j;\ i \neq j$. In other words, each individual represents a partition of $\mathbb{U}$ into non-overlapping sets, the union of which contains all items exactly once. If the total bin capacity is c, the spare capacity s_i of each bin B_i is then

$$s_i = c - \sum_{j=1}^{m_i} w_{b_{ij}} \tag{2}$$

Negative values correspond to overfilled bins, and these bins are penalised by a user-defined penalty factor $p \leq -1$ that negatively impacts on the cost of the bin, giving bin costs:

$$\delta_i = \begin{cases} s_i & \text{if } s_i \geq 0 \\ ps_i & \text{if } s_i < 0 \end{cases} \tag{3}$$

The fitness of the individual encoding B is the sum of bin costs $\sum_{i=1}^{|B|} \delta_i$ and the quality of the solution is simply the number of bins $|B|$. The GA works best if some over-filled bins remain in the early populations (i.e. $|p|$ is not too high), but not too many (i.e. $|p|$ is not too low). A penalty value of $p = -10$ was found to be suitable for the test cases considered here, but other values may work better for different sets of benchmark problems.

The evolution begins with a randomly initialized population. There are many ways this initialization could be done, with various levels of complexity. For this study, the simplest possible process that results in no over-filled bins was employed: First, a random permutation of all items is generated, giving an ordered list of labels $\Delta_{\mathbb{U}}$, and items are added sequentially to bins while possible. As soon as a bin is full, or the inclusion of an item would exceed the current bin's capacity, that bin is closed and added to the individual, and a new bin is opened. The following shows a typical simple BPP initialization:

- Problem size: $n = 5$ items of capacity $c = 7$
- Problem weights: $\{w_1 = 3, w_2 = 5, w_3 = 4, w_4 = 6, w_5 = 2\}$
- Random permutation of (labels of) items: $\Delta_{\mathbb{U}} = (5, 2, 3, 4, 1)$
- Generate B by adding items one at a time:
  - $B = \{\}, B_1 = \{5\}, s_1 = 5$ (spare capacity, continue adding)

  - $B = \{\}, B_1 = \{5, 2\}, s_1 = 0$ (bin full, start a new bin)
  - $B = \{B_1\}, B_2 = \{3\}, s_2 = 3$ (next item won't fit, start a new bin)
  - $B = \{B_1, B_2\}, B_3 = \{4\}, s_3 = 1$ (next item won't fit, start a new bin)
  - $B = \{B_1, B_2, B_3\}, B_4 = \{1\}, s_4 = 4$ (all items placed, finish)
- Initialized individual: $B = \{\{5, 2\}, \{3\}, \{4\}, \{1\}\}, |B| = 4$

Once initialized in this way, the population evolves via a series of crossover and mutation operations.

The mutation operator considers every item contained within any newly generated offspring, and mutates it with probability p_m: the item to be mutated is, with probability 0.5, either moved to another randomly chosen bin or exchanged with a randomly chosen item from another bin. It is possible that the same item could be mutated multiple times, although the probability of that happening is very small.

The central feature of ESGA is its crossover operator which creates a single offspring B^γ from two parents, B^α and B^β, selected from P. First, the crossover operator merges B^α and B^β to yield a set of bins that contains every item exactly twice, and this needs to be reduced by half to yield a valid offspring. This reduction not only aims to preserve as much as possible of the structural information (bin memberships) from the parents, but also attempts to construct the offspring from the most promising sub-solutions (bins B_i with the least cost). First, the cost δ_i of each bin in $B^\alpha \cup B^\beta$ is calculated and used to establish an ordered list $\Lambda = [\Lambda_i \in B^\alpha \cup B^\beta : \delta_{\Lambda_i} \leq \delta_{\Lambda_{i+1}}]$ of bins Λ_i sorted in non-decreasing order of their cost (with bins of equal cost in random order). Starting from an empty offspring B^γ, the first phase of crossover considers all bins in the order of increasing cost, and those that are disjoint to all the bins already contained in B^γ are added to B^γ. In the second phase, all bins that contain at least one item not yet contained in $\bigcup B^\gamma$ are added to B^γ, with any duplicate items replaced by the most similar items not already present. The similarity of items is judged by their weight differences. More specifically, item b_i is replaced by item $b_j \notin \bigcup B^\gamma$ if $|w_{b_i} - w_{b_j}| \leq |w_{b_i} - w_{b_k}| \; \forall b_k \in \mathbb{U} - \bigcup B^\gamma$. When there are no remaining items to be added, the duplicates are simply deleted. The pseudocode for this crossover operator is shown in Algorithm 1.

The following simple example of crossover should clarify this: Consider a simple BPP with $n = 5$ items of weights $\{w_1 = 1, w_2 = 2, w_3 = 3, w_4 = 4, w_5 = 5\}$ and bin capacity $c = 5$. It is clear that the optimum solution consists of 3 bins $\{\{5\}, \{4, 1\}, \{3, 2\}\}$. Two parents B^α and B^β are crossed over to yield offspring B^γ, and an overfill penalty value of $p = -5$ is used. For clarity, each bin is labelled by its originating parent and its cost:

- $B^\alpha = \{\{5\}_0^\alpha, \{1, 3, 2\}_5^\alpha, \{4\}_1^\alpha\}$
- $B^\beta = \{\{5, 1\}_5^\beta, \{2, 3\}_0^\beta, \{4\}_1^\beta\}$
- $\Lambda = [\{5\}_0^\alpha, \{2, 3\}_0^\beta, \{4\}_1^\beta, \{4\}_1^\alpha, \{1, 3, 2\}_5^\alpha, \{5, 1\}_5^\beta]$
- Phase 1: $B^\gamma = \{\{5\}_0^\alpha, \{2, 3\}_0^\beta, \{4\}_1^\beta\}$
- Phase 2: $B^\gamma = \{\{5\}_0^\alpha, \{2, 3\}_0^\beta, \{4\}_1^\beta, \{1, \not{3}, \not{2}\}_4^\alpha\}$

The crossover operator is clearly greedy in the sense that it always considers the seemingly best (most tightly packed) bins first. This approach will consequently be susceptible to entrapment in local optima, particularly in cases where the optimal solutions contain bins with moderate amounts of free space. Adding noise is a standard procedure for escaping local optima and although mutations supply this to a certain extent, often they are insufficient by themselves. Here noise can be added directly to the cost of each bin, rendering the ordering of the bins only approximate and thus affecting the formation of the new offspring directly; unlike mutations, the noise only affects the ordering of bins, not their content. The maximum amount of noise, measured as a percentage of a bin's capacity, can be determined by the evolutionary encoding itself: a binary sequence encodes an integer percentage of maximum potential noise that can be randomly added to the bin's true cost during each crossover event. A four bit control sequence proved effective, encoding a maximum noise range of 5-20%. Attempts to evolve the noise over the full 0-100% range failed to produce any further improvements.

Algorithm 1 Pseudo-code for the crossover operator used in ESGA.

```
Select parents B^α and B^β from P
Merge parents to give bin set B^α ∪ B^β
Sort bin set by non-decreasing cost δ_i to yield Λ = [Λ_i ∈ B^α ∪ B^β : δ_{Λ_i} ≤ δ_{Λ_{i+1}}]
Create an empty offspring B^γ ← {}
// Phase 1: find and add mutually exclusive bins
for i = 1, ..., |B^α| + |B^β| do
   if Λ_i ∩ (⋃ B^γ) = ∅ then
      B^γ ← B^γ ∪ Λ_i
   end if
end for
// Phase 2: allocate or delete items from remaining bins
for i = 1, ..., |B^α| + |B^β| do
   if Λ_i ⊄ (⋃ B^γ) then
      for b_ij ∈ Λ_i ∩ (⋃ B^γ) do
         if ∃b_x ∈ 𝕌 − ⋃ B^γ then
            b_ij ← b_x with min |w_{b_ij} − w_{b_x}|
         else
            delete b_ij
         end if
      end for
      B^γ ← B^γ ∪ Λ_i
   end if
end for
```

The ESGA crossover process ensures that not only does the offspring inherit the most tightly packed bins from both parents, but also that the bins with the least amount of free space are preferentially preserved. Although the bins are considered as independent units, or modules, there exists an interrelationship between them, because any individual must contain all items exactly once. This is guaranteed by the replacement procedure, inspired by properties of the genetic code, with unwanted items replaced by the most similar available ones. This means that even modified bins retain their original composition to the maximum degree possible. Furthermore, the penalty term applied to infeasible bin arrangements ensures that those bins are less likely to be considered first during crossover, and, assuming the penalty term is chosen appropriately, will tend to purge all infeasible bins over time. The need for an explicit repair procedure is thus avoided.

3.4 Experimental Setup

All experiments designed to test the ESGA were based on a canonical steady-state GA with a population of size 150. Parents were selected at random and the offspring placed into the population depending on a binary tournament with a randomly chosen individual. The previously discussed crossover operator was applied with a probability of 0.8. In the event that crossover was not applied, a randomly chosen parent was reproduced asexually (i.e. cloned). The mutation probability was set to $1/n$, and the noise control sequence was mutated by flipping each bit with probability 0.25. These parameters were established systematically, although by no means exhaustively, in a preliminary series of experiments.

We are primarily interested in finding solutions for "hard" packing problems, that other algorithms are known to perform poorly on, so our testing focussed mainly on the hardest group of benchmark problems, set 3, which contains 10 problem instances. The algorithm was executed 50 times on each instance, with an imposed limit of 50,000 function evaluations (FEs) for each run. The performance was

Instance	Opt	%	Avg FE	Min FE	Max FE	Accuracy	Avg over
hard0	56	100	7177.38	2488	17186	56	0
hard1	57	100	5826.58	2384	15996	57	0
hard2	56	0	-	-	-	57	0
hard3	55	0	-	-	-	56	0
hard4	57	98	13474.10	6362	35809	57.02	0
hard5	56	92	15361.15	6316	44610	56.06	0.04
hard6	57	100	5501.98	1924	10143	57	0
hard7	55	100	7899.92	2275	22461	55	0
hard8	57	100	5265.58	1830	21475	57	0
hard9	56	98	13931.55	5079	30045	56.02	0

Table 1: Summary of the ESGA results: Each instance is listed by its name and optimal number of bins (Opt). The third column (%) is the percentage of solved trials, followed by the average number of function evaluations required (Avg FE). Next, the minimum and maximum number of function evaluations required are shown (Min FE, Max FE). The last two columns show the overall accuracy of the algorithm (Accuracy), and the average number of overfilled bins in the final solution (Avg over).

evaluated by measuring the number of times the optimum for each instance was found, and how many FEs, on average, were required. Accuracy was measured by the average number of bins used and how many bins, on average, were overfilled.

3.5 Results and Analysis

The results from all our ESGA experiments are summarized in Table 1. It shows that the algorithm successfully found the global optimum in 8 out of the 10 problem instances. In 5 cases, the optimum was found in all trials, while in 3 other cases the optimum was found in over 90% of trials. There are two cases, hard2 and hard3, where the global optimum was not found at all. However, all the final solutions for these two cases were exactly a single bin away from the optimal solution. The same is true for the other instances which were not solved with 100% reliability – the unsuccessful trials were all at most one bin away from the global optimum. Interestingly, there is only a single trial in which an invalid solution was returned (for hard5), highlighting the algorithm's ability to purge infeasible sub-solutions.

The overall behaviour of the algorithm was further examined by looking at its convergence over time. This was done by focussing on the best individual in the population at each generation. There are three factors to consider: fitness (bin slack), number of bins, and number of constraint violations. Three representative instances were selected for this analysis, one of which had been solved consistently (hard0), one which had never been solved (hard2), and one that had been solved but not all the time (hard5). Averages over 50 trials are presented in Figure 1. These graphs indicate that the algorithm quickly converges towards a (near-)optimal number of bins, although several of them are initially overfilled. In fact, the number of overfilled bins roughly peaks when the optimal number of bins is first found. The fitness of the best individual also seems to stagnate around that point. The subsequent change is very gradual, reducing the number of overfilled bins without affecting the actual number of bins. This change is hardly visible in the graphs: although a penalty factor of $p = -10$ is applied to infeasible bin arrangements, the penalty only applies at the level of the bin and hence may not make a significant contribution to the overall fitness of an individual (especially if only few bins are overfilled).

It is clearly important that any new algorithm be tested against existing algorithms. A comparison of different approaches for this particular set of benchmark problems is presented in Table 2, and shows that ESGA ranks highly in terms of success rate. In fact, only the HL_BP algorithm of Alvim et al. (2004) performs better. Moreover, HI_BP requires a number of pre-processing steps, and undergoes multiple

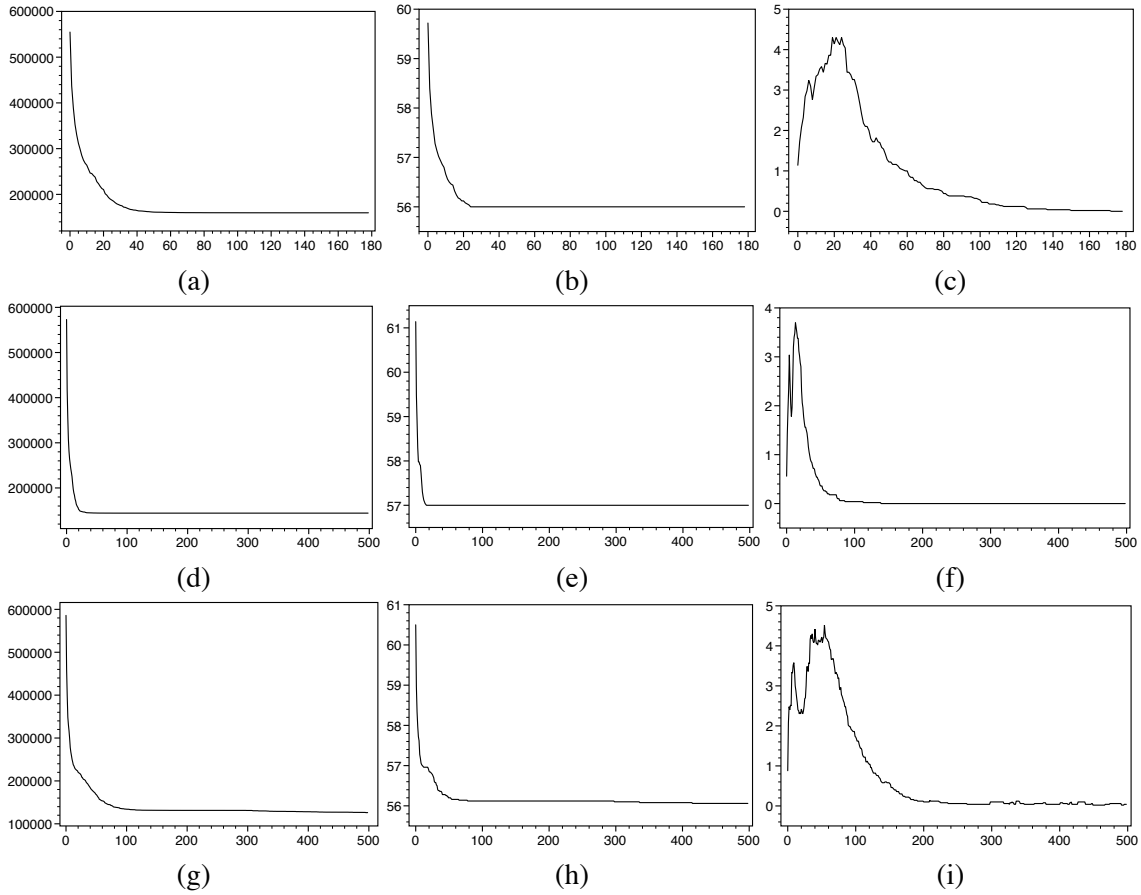


Figure 1: Graphs showing the algorithm's convergence for three selected problems: hard0 (a,b,c), hard2 (d,e,f) and hard5 (g,h,i). The first graph in each row depicts the best fitness over time as measured by the space available across all bins. The second graph shows the number of bins of the population's best solution over time. The third graph shows the number of overfilled bins in the best solution over time.

phases exploiting several mathematical properties of the BPP. ESGA, on the other hand, is very simple to implement and could be easily applicable to other problems that exhibit similar structural properties.

3.6 Further Experimentation

The greedy nature of the crossover operator and its ability to purge invalid sub-solutions makes ESGA highly suitable for those instances of the BPP that are usually considered difficult or "hard" (e.g., those where the global optimum consists of very tightly packed bins). Although we are primarily concerned with solving those problems that existing algorithms find difficult, for completeness it is worth checking how well ESGA performs on other instances, that are normally considered "less difficult". The algorithm was therefore tested on the other two sets of benchmark problems described above (i.e. Sets 1 and 2), for problem sizes up to 200 items. That means a total of $540 + 360 = 900$ instances. For each instance, the algorithm was executed 25 times with a limit of 25,000 FEs. Tables 3 and 4 show the success rates obtained, averaged over the various problem classes. The success rates are seen to be very similar for both sets. Table 5 summarises the performance of ESGA across all three sets of benchmark problems, and shows that it is relatively robust across different types of instances. In fact, there seems to be no single attribute that indicates why the algorithm fails to solve some instances and not others, and more analysis of this issue will clearly be required to develop the algorithm further.

Approach	Reference	Solved
MBS	(Gupta and Ho 1999) in (Fleszar and Hindi 2002)	0
MBS'	(Fleszar and Hindi 2002)	0
Perturbation MBS'	(Fleszar and Hindi 2002)	0
Sampling MBS'	(Fleszar and Hindi 2002)	0
FFD	(Scholl et al. 1997) in (Fleszar and Hindi 2002)	0
BFD	(Scholl et al. 1997) in (Fleszar and Hindi 2002)	0
WFD	(Scholl et al. 1997) in (Fleszar and Hindi 2002)	0
B2F	(Scholl et al. 1997) in (Fleszar and Hindi 2002)	0
FFD-B2F	(Scholl et al. 1997) in (Fleszar and Hindi 2002)	0
Relaxed MBS'	(Fleszar and Hindi 2002)	2
VNS	(Fleszar and Hindi 2002)	2
Perturbation MBS' & VNS	(Fleszar and Hindi 2002)	2
Genetic Algorithm	(Iima and Yakawa 2003)	3
BISON	(Scholl et al. 1997) in (Iima and Yakawa 2003)	3
Dual Tabu	(Scholl et al. 1997) in (Fleszar and Hindi 2002)	3
Exon Shuffling GA	This paper	8
HI_BP	(Alvim, Ribeiro, Glover, and Aloise 2004)	10

Table 2: A comparison of ESGA with previously published algorithms in terms of cases solved among the same set of 10 benchmark instances.

Class	%	Class	%	Class	%
N1C1W1	95.4	N2C1W1	96.4	N3C1W1	81.2
N1C1W2	100	N2C1W2	96.2	N3C1W2	84.0
N1C1W4	99.6	N2C1W4	96.8	N3C1W4	93.2
N1C2W1	99.2	N2C2W1	77.0	N3C2W1	63.2
N1C2W2	97.6	N2C2W2	90.2	N3C2W2	79.2
N1C2W4	93.8	N2C2W4	93.0	N3C2W4	83.6
N1C3W1	95.6	N2C3W1	82.8	N3C3W1	51.6
N1C3W2	100	N2C3W2	87.8	N3C3W2	70.6
N1C3W4	97.0	N2C3W4	89.8	N3C3W4	73.0

Table 3: Summary of the success rates of ESGA on Set 1 of benchmark problems.

We do not compare ESGA to other approaches on these instances as we did in the previous section. ESGA was aimed at solving problem instances that other algorithms find difficult or impossible, and Table 2 demonstrates its success. Further comparison is less relevant to this objective, and may indeed be misleading if we only compare the success rates without regard to running times. It is actually very difficult to make meaningful comparisons to these highly problem specific methods, as they usually run very fast, but may not be able to locate the best-known solutions. ESGA, on the other hand, is designed to find the optimal solution reliably and within 'reasonable' time, and this ability is reflected in the data.

4 The Exonic Genetic Algorithm (ExGA)

The design of the ExGA was motivated by the fact that many hard packing problems in the real world involve more complex constraints than the BPP considered previously. Constraints essentially fragment the search space into feasible and unfeasible regions, and considerable effort may be wasted exploring the

Class	%	Class	%	Class	%
N1W1B1	99.6	N2W1B1	98.8	N3W1B1	96.0
N1W1B2	100	N2W1B2	73.6	N3W1B2	23.2
N1W1B3	72.0	N2W1B3	59.6	N3W1B3	14.4
N1W2B1	100	N2W2B1	100	N3W2B1	90.0
N1W2B2	100	N2W2B2	99.2	N3W2B2	81.2
N1W2B3	100	N2W2B3	85.6	N3W2B3	42.4
N1W3B1	99.6	N2W3B1	91.2	N3W3B1	80.0
N1W3B2	100	N2W3B2	100	N3W3B2	99.6
N1W3B3	98.8	N2W3B3	100	N3W3B3	65.2
N1W4B1	100	N2W4B1	100	N3W4B1	100
N1W4B2	100	N2W4B2	100	N3W4B2	100
N1W4B3	100	N2W4B3	94.4	N3W4B3	86.4

Table 4: Summary of the success rates of ESGA on Set 2 of benchmark problems.

	%	%	%		
n→	50	100	200	100%	0%
Set 1	97.6	90.0	75.5	62.8	4.8
Set 2	97.5	91.9	73.2	81.9	8.3
Set 3			78.8	50.0	20.0

Table 5: Summary of the average success rates of ESGA on all three sets of benchmark problems. The second row shows the number of items n for each class of problems and the three rows below show the average success rate across all instances of that class. The two right-most columns indicate the percentage of instances that have been solved across all trials (100%) and those that have never been solved (0%).

unfeasible regions. We have shown elsewhere (Rohlfshagen and Bullinaria 2006) that repair functions perform significantly better than penalty based approaches (such as that used with the ESGA) for the instances of the MKP considered here, and a similar conclusion was arrived at by Gréwal et al. (2006) for a set of digital signal processor benchmark problems. Here we show how the principles of RNA editing can be abstracted to create effective repair functions that allow constraint satisfaction and modular encoding within our evolutionary framework. We present several variants of our algorithm, and test them on the well known MKP problem.

4.1 The Multiple Knapsack Problem (MKP)

The MKP is a much studied NP-hard combinatorial optimisation problem that is a generalisation of the single knapsack problem. It is highly constrained, and has received particular attention in the GA community because it can be conveniently encoded by binary strings. There are actually many variants of the MKP. We shall use the version for which the objective is to fill a series of m knapsacks, each of capacity c_j, with any subset of n items, each of value v_i and weight w_{ij} when in knapsack j, in such a way that the combined value of all chosen items is maximized without exceeding any of the knapsack's capacities. More formally, the aim is to find the maximum fitness

$$\max(\sum_{i=1}^{n} v_i X_i) \quad | \quad \sum_{i=1}^{n} w_{ij} X_i \leq c_j \quad \forall j = 1, \ldots, m \tag{4}$$

in which $X_i \in \{0, 1\}$ indicates inclusion of item i. We studied the widely used SAC'94 suite of benchmark MKP problems, which may be found online at `http://elib.zib.de/pub/Packages/mp-testdata/ip/sac94-suite/`. It contains 55 problem instances which range in size from 15-105 objects and 2-30 knapsacks.

4.2 Related Work

The MKP problem is one of the most widely studied constrained optimisation problems in the field of EC. Here we concentrate on reviewing four approaches that have attempted to solve instances of the SAC'94 library of benchmark problems. The earliest was by Khuri, Bäck, and Heitkötter (1994) who suggested the use of a graded penalty term (violation product) that penalises invalid solutions depending on the number of constraint violations. They used a cGA with a modified fitness function to focus the search on the feasible regions of the search space. This concept was taken further by Kimbrough, Lu, Wood, and Wu (2002) who proposed a 2-Market GA (2-MGA) that restores feasibility using a two phase procedure. The first phase performs optimality improvement and the second phase feasibility improvement. They tested two different penalty terms, violation product and sum of violations, the former of which is identical to the graded penalty term suggested by Khuri et al. (1994).

Further improvement was achieved by using instance specific knowledge: Cotta and Troya (1997) suggested a Hybrid GA (HGA) with a greedy construction heuristic that builds solutions by selectively choosing items of high value-weight ratios. The encodings represent perturbations of the problem instance, generated by manipulation of the profits across all items (noise). Jun, Xiande, and Lu (2003) suggested an Evolutionary Game Algorithm (EGA) that also utilised the value-weight ratio of each item. That domain-specific information was used to construct a repair function that removes items in order of increasing ratios until feasibility is obtained. This is followed by a second phase that adds items, if possible, in order of their decreasing value-weight ratios. Diversity was maintained by a perturbation phase that inflicts random changes with small probability prior to repair (mutations). The results of these four studies are summarised in Table 6 and are used later for comparison with the performance of our new approach. The superiority of EGA over the previous approaches demonstrates the importance of repair functions that produce tightly packed solutions and maximize the value-weight ratios of the items considered for inclusion.

4.3 Problem Specific Knowledge

A general review of numerous different optimisation techniques shows that problem specific approaches are generally likely to outperform their 'blind' counterparts (Grefenstette 1987). Most algorithms use some understanding of the structure of the problem in their design, but only a few use instance specific attributes. As the previous section showed, approaches for the MKP that make explicit use of the items' value-weight ratios clearly outperform approaches that do not exploit such information. It may be difficult, however, to utilise domain specific knowledge properly. It is, for example, not clear whether to use the average value-weight ratio of an item across all knapsacks or the minimum/maximum value. The new algorithm we are proposing here does not depend on such instance specific knowledge. Instead, it relies upon a general assumption made about the class of MKP problems in general: namely that optimal solutions have, on average, minimal waste of capacity and, because fitness is the sum of values, are more likely to contain items with higher value-weight ratios. This intuitive concept has already proved useful in the previously mentioned EGA approach of Jun, Xiande, and Lu (2003).

4.4 ExGA Variant I

During evolution, crossover and mutation may generate individuals that violate one or more constraints, and these must be discarded, unless they can be used to generate valid individuals. This is analagous to

			Approach							
			cGA		HGA		EGA		2-MGA	
Instance	n/m	Opt	5,000-200,000		20,000		20,000		2,500,000	
hp2	35/4	3186	-		-		100%	3186	-	3186
pb6	40/30	776	-		-		100%	776	-	730.2
pb7	37/30	1035	-		-		-	1034.6	-	1033
pet3	15/10	4015	83%	4012.7	100%	4015	100%	4015	-	
pet4	20/10	6120	33%	6102.3	94%	6119.4	100%	6120	-	
pet5	28/10	12400	33%	12374.7	100%	12400	100%	12400	-	
pet6	39/5	10618	4%	10536.9	60%	10609.8	99%	10617.9	-	
pet7	50/5	16537	1%	16378	46%	16512	100%	16537	-	16486.6
sent01	60/30	7772	5%	7626	75%	7767.9	100%	7772	-	7769.8
sent02	60/30	8722	2%	8685	39%	8716.5	69%	8721.7	-	8720.4
weing7	105/2	1095445	0%	1093897	40%	1095386	100%	1095445	-	1094727
weing8	105/2	624319	6%	613383	29%	622048.1	73%	623844.8	-	623627.8
weish12	50/5	6339	-		-		100%	6339	-	6339
weish17	60/5	8633	-		-		100%	8633	-	8633
weish21	70/5	9074	-		-		100%	9074	-	9074
weish22	80/5	8947	-		-		100%	8947	-	8947
weish25	80/5	9939	-		-		100%	9939	-	9939
weish29	90/5	9410	-		-		100%	9410	-	9203.2

Table 6: Summary of previous results from the literature: Each problem instance is listed alongside its size (number of items/number of knapsacks) and optimal solution (Opt). For each approach is shown the percentage of times the global optimum has been found and the final average value, for all available instances. The number of function evaluations are shown below the name of each approach: cGA refers to the canonical GA with graded penalty term, HGA is the Hybrid GA with greedy construction heuristic, EGA is the Evolutionary Game Algorithm, and 2-MGA is the 2-Market GA.

RNA sequences that are discarded by means of "nonsense-mediated decay" because they fail to produce functional proteins as a result of premature stop codons or shifted reading frames. RNA editing, however, may selectively target individual nucleotides to restore protein synthesis, and can thus be viewed as a repair algorithm that prevents degradation. The idea of ExGA is to create a repair function, loosely based upon the principles of RNA editing, that keeps all individuals within the feasible region of the search space.

For the MKP, the solutions are sub-sets of the available items, that are conveniently encoded as binary inclusion vectors $\vec{X}$ over all items, rather different to the sets of bins B that partition the set of all items in the BPP. We need repair procedures to adjust $\vec{X}$ so that the constraints are satisfied. Jun, Xiande, and Lu (2003) used a repair function which adjusts knapsack contents in order of value/weight ratios until feasibility is achieved. ExGA follows a similar approach, but rather than using value/weight ratios, it *evolves* the relative order of items to be considered for removal or inclusion. That ordering of items is partial, rather than strict: groupings are used to establish order, but items within the same grouping are chosen randomly. The encoding of each individual solution $(\vec{X}, \vec{G})$ thus consists of two vectors: an inclusion vector $\vec{X} \in \{0,1\}^n$ that is identical to the traditional encoding of the MKP as a binary-encoded optimisation problem, with item i included in the solution if and only if $X_i = 1$. A second vector, $\vec{G} \in \{1,\ldots,k\}^n$, assigns to each item i a grouping number G_i, with a user-defined number of groupings k. Throughout the execution of the algorithm, a mutation operator is used that iterates over all items and randomly re-assigns the value of G_i with probability p_{m_G}. Also, a standard binary mutation operator works in an identical fashion on the X_i with bit-probability p_{m_X}.

The grouping numbers G_i define a structure that is subsequently exploited during the repair process.

Prior to the application of the repair procedure, the overall fitness, and the spare capacity

$$s_j = c_j - \sum_{i=1}^{n} w_{ij} X_i \tag{5}$$

of each knapsack j is computed. Any negative values for s_j indicate that there is a constraint violation that needs repair. The repair consists of two phases: an exclusion (feasibility) phase that removes a sufficient number of items to render all spare capacities positive, and then an inclusion (greedy) phase that adds as many additional items as possible without violating any constraints. The second phase takes place even if there are no constraint violations needing repair.

The grouping numbers establish a partial ordering of items. All items are sorted according to increasing grouping numbers to give a hierarchy of k sets $\vec{H} = [H_1, H_2, \ldots, H_k]$ with $H_j = \{i \mid G_i = j\}$. Here, the vector $\vec{H}$ contains, in order, k sets, each of which contain indices to the items under consideration. The feasibility phase considers all groupings H_j in increasing order ($j = 1, 2, \ldots, k$) and chooses items H_{ji} within the same grouping j in random order for removal from the knapsacks (i.e. for setting $X_{H_{ji}} = 0$). After each removal, the spare capacities are updated, and the process continues until no constraint violations remain. The repair moves on to the next grouping only once all items within the current grouping have been considered.

The second phase considers the groupings in reverse order ($j = k, k-1, \ldots, 1$), and attempts to fill any spare capacity by adding items (i.e. setting $X_{H_{ji}} = 1$) as long as the inclusion does not result in the violation any of the constraints. This process continues until all items in all groupings have been considered. The pseudocode for the full repair mechanism is shown in Algorithm 2.

As with ESGA, ExGA employs a standard evolutionary framework that consists of the common selection, crossover, mutation and replacement cycle. However, because of the greedy nature of the repair process, a few changes were made in an attempt to preserve the population's diversity. Now, the first parent is chosen at random, and the second parent is chosen to be the individual in the population with maximum Hamming distance to the first. Also, each offspring now replaces one of its parents, chosen randomly (rather than, as in the ESGA, a completely randomly chosen individual), and duplicate encodings are not allowed to enter the population. Although it is debatable whether these changes can be considered 'nature inspired', with our limited population sizes they are certainly important for improving the algorithm's overall performance.

4.5 ExGA Variant II

A major disadvantage of any repair technique is the additional computational cost inherent in the inspection and modification of infeasible encodings. This cost should be accounted for, and, if significant, minimized. The fitness of an encoding can be calculated in $\mathcal{O}(n)$, and the constraints require $\mathcal{O}(mn)$, giving a total evaluation cost of $\mathcal{O}(n + mn)$. In the worst case, the cost of repair is $\mathcal{O}(2mn)$. The idea behind ExGA II is to reduce the cost of repair by carrying it out as a solution is constructed from an evolved construction template, rather than beginning from a potential solution that has been evolved. In the process, we shall see if such an alternative encoding can result in better solutions too.

The encoding $(\vec{X}, \vec{G})$ for the first variant, ExGA IIa, is essentially identical to that of ExGA I, except that the binary inclusion vector $\vec{X}$ is now constructed on-the-fly using the grouping numbers $\vec{G}$ alone. In effect, we start with $\vec{X} = \vec{0}$, so there are automatically no constraint violations to begin with, and the 'repair' process is reduced to the second phase, which is now more accurately described as a decoder-like construction process. The constraints now only need checking during the construction of $\vec{X}$ from $\vec{G}$ as described above, and the search is effectively restricted to the feasible regions of the search space by mini-repair events at each stage. The pseudocode for the construction process for $\vec{X}$ is shown in Algorithm 3.

Algorithm 2 Pseudocode for the repair mechanism of ExGA I.

```
Construct H⃗ from G⃗ such that H_j = {i | G_i = j}
//Phase 1: Exclusion (feasibility) phase
if ∃s_r < 0 then
  for j = 1, 2, ..., k do
    for i ∈ H_j do
      if X_i = 1 then
        X_i ← 0
        update all s_r
        if all s_r ≥ 0 then
          terminate phase 1
        end if
      end if
    end for
  end for
end if
//Phase 2: Inclusion (greedy) phase
for j = k, k − 1, ..., 1 do
  for i ∈ H_j do
    if X_i = 0 then
      X_i ← 1
      update all s_r
      if ∃s_r < 0 then
        X_i ← 0
        revert all s_r
      end if
    end if
  end for
end for
```

Although $\vec{X}$ is generated dynamically, it is still used to compare individuals in the population during selection and replacement. Furthermore, it is used to guide the mutation operator that alters the grouping numbers in $\vec{G}$. The mutation operator ensures an item is moved from a group where it has a certain binary value (0 or 1) to a group that contains at least one item with the opposite binary value. The reason for this is as follows: the first group (from left to right) that contains an item that can not be included (i.e. its binary value is 0) defines a critical threshold. All items situated in groups to the left of this threshold will be included in the solution independent of their order. The mutation operator moves items across this threshold to ensure that the mutation is likely to affect the construction of the next solution.

The encoding for the second variant aims to achieve an even better performance by allowing the number of groupings to adapt during the evolutionary process. It now consists of four vectors $(\vec{X}, \vec{G}, \vec{K}, \vec{R})$, with $\vec{X}$ and $\vec{G}$ as before, $\vec{K} \in \{0,1\}^q$ and $\vec{R} \in [0,1]^n$. The binary vector $\vec{K}$, which undergoes standard binary mutation, encodes the number of groupings k used to group the items, and $\vec{R}$ encodes the ordering. The k and $\vec{R}$ are used together to construct the group membership vector $\vec{G}$ on-the-fly using:

$$G_i = \lceil R_i k \rceil \tag{6}$$

and this is subsequently used to construct the binary solution $\vec{X}$ as described above. The vector $\vec{R}$ is mutated in a similar fashion to $\vec{G}$ in ExGA IIa, with each floating point number replaced by a randomly chosen one (in $[0,1]$) such that the item would cross the threshold as discussed above. Any mutations

Algorithm 3 Pseudocode for the decoder-like construction mechanism of ExGA IIa.

```
Construct H⃗ from G⃗ such that H_j = {i | G_i = j}
//Construct X⃗ from H⃗ starting from X⃗ = 0⃗
for j = k, k − 1, ..., 1 do
  for i ∈ H_j do
    X_i ← 1
    update all s_r
    //Mini-repair
    if ∃s_r < 0 then
      X_i ← 0
      revert all s_r
    end if
  end for
end for
```

to $\vec{R}$ will change the number of groups k used and this will in turn affect the distribution of items. The length q of $\vec{K}$ is chosen to allow any power of 2 smaller than n to be expressed. The use of floating point numbers to evolve permutations is not new (e.g., see the random keys of Bean (1994)), but has not yet, to the best knowledge of the authors, been used before in combination with a segmented encoding (i.e. to evolve a partial ordering).

4.6 Illustrative Example of Encodings and Algorithms

The following simple example illustrates all three ExGA variants. Consider the $n = 5$, $m = 2$ MKP shown in Table 7 with capacities $c_1 = c_2 = 25$.

In the case of ExGA I, assume an individual has been generated with $k = 4$,$\vec{X} = (1, 1, 1, 0, 1)$ and $\vec{G} = (1, 3, 3, 2, 4)$. It follows that $s_1 = -2$ and $s_2 = -11$ (as $\sum \vec{W} = 27$ for c_1 and $\sum \vec{W} = 36$ for c_2), and the 'repair' proceeds as follows:

- Phase 1: Process the components of $\vec{H} = (\{1\}_1, \{4\}_2, \{2, 3\}_3, \{5\}_4)$ in order $i = 1, 2, 3, 4$:
  - Group H_1: $X_1 = 1$, invert to give $X_1 = 0$, $s_1 = 3$ and $s_2 = -3$
  - Group H_2: $X_4 = 0$, so ignore
  - Group H_3: randomly choose item $X_3 = 1$, invert to give $X_3 = 0$, $s_1 = 13$ and $s_2 = 9$
  - No constraint violations remaining, so proceed to next phase
- Phase 2: Process the components of $\vec{H} = (\{1\}_1, \{4\}_2, \{2, 3\}_3, \{5\}_4)$ in order $i = 4, 3, 2, 1$:
  - Group H_4: $X_5 = 1$, so ignore
  - Group H_3: randomly choose item, X_3: if X_3 is inverted, $s_1 = 3$ and $s_2 = -3$ so ignore
  - Group H_3: remaining item, $X_2 = 1$, so ignore
  - Group H_2: if X_4 is inverted $s_1 = 10$ and $s_2 = 6$, so include: $X_4 = 1$
  - Group H_1: if X_1 is inverted $s_1 = 5$ and $s_2 = -2$, so ignore

The final solution after repair is thus $\vec{X} = (0, 1, 0, 1, 1)$ which is feasible.

In the case of ExGA IIa, assume an individual has been generated with $k = 4$ and$\vec{G} = (1, 3, 3, 2, 4)$. We start with $\vec{X} = \vec{0}$, $s_1 = 25$ and $s_2 = 25$. The algorithm then 'constructs' $\vec{X}$ as follows:

- Process the components of $\vec{H} = (\{1\}_1, \{4\}_2, \{2, 3\}_3, \{5\}_4)$ in order $i = 4, 3, 2, 1$:
  - Group H_4: if $X_5 \leftarrow 1$, $s_1 = 23$ and $s_2 = 20$, so include $X_5 = 1$
  - Group H_3: randomly choose item, X_2: if $X_2 \leftarrow 1$, $s_1 = 13$ and $s_2 = 9$, so include $X_2 = 1$

Items	1	2	3	4	5
Values	10	13	10	7	5
Weights	5	10	10	3	2
	8	11	12	3	5

Table 7: A simple $n = 5$, $m = 2$ MKP.

Parameter	ExGA I	ExGA IIa	ExGA IIb
population size	100	100	100
selection	random	tournament	tournament
k	n/2	n/4	adaptive
p_c	0.9	0.9	0.9
p_{m_X}	0.5/n	-	-
p_{m_G}	0.5/n	1/n	1/n

Table 8: The various parameter settings for ExGA I, IIa and IIb.

- Group H_3: remaining item X_3: if $X_3 \leftarrow 1$, $s_1 = 3$ and $s_2 = -3$ so keep $X_3 = 0$
- Group H_2: if $X_4 \leftarrow 1$, $s_1 = 10$ and $s_2 = 6$, so include $X_4 = 1$
- Group H_1: if $X_1 \leftarrow 1$, $s_1 = 5$ and $s_2 = -2$, so keep $X_1 = 0$

This yields the feasible solution $\vec{X} = (0, 1, 0, 1, 1)$.

Finally, in the case of ExGA IIb, assume an individual has been generated with $\vec{K} = (0, 1, 0, 0)$ and $\vec{R} = (0.17, 0.55, 0.71, 0.30, 0.89)$. The vector $\vec{K}$ translates to $k = 4$. Then, the actual groups, $\vec{G}$, are established as follows:

- G_1: $\lceil 0.17 * 4 \rceil = 1$
- G_2: $\lceil 0.55 * 4 \rceil = 3$
- G_3: $\lceil 0.71 * 4 \rceil = 3$
- G_4: $\lceil 0.30 * 4 \rceil = 2$
- G_5: $\lceil 0.89 * 4 \rceil = 4$

This yields the same $\vec{G}$ as in the ExGA IIa example, and so the remainder of this example is identical.

4.7 Experimental Setup

Experiments were conducted to perform a comparison of the ExGA variants, and with the earlier approaches reviewed in Section 4.2. Each ExGA algorithm was executed 100 times on each instance of the SAC'94 library of benchmark problems, with a maximum of 20,000 function evaluations. The most successful ExGA variant was subsequently compared against the earlier approaches on a subset of 18 instances. The parameter settings for each variant are shown in Table 8. For each algorithm and instance was recorded the percentage of times the global optimum was found and the average number of FEs required to find it, and the statistical significance of the differences in FEs required between algorithms was determined using t-tests requiring $p < 0.005$.

4.8 Results and Analysis

The results of all experiments are presented in Tables 9 and 10. For each variant is shown the success rate and the average number of FEs required, and the significant differences in performance between variants. The overall performance differences are summarised in Table 11. It is evident that all three algorithms

Instance	n	m	ExGA I		ExGA II a		ExGA II b		I-IIa	I-IIb	IIa-IIb
FLEI	20	10	100%	1773	100%	2175	100%	2118			
HP1	28	4	100%	3247	100%	2397	100%	2417	*	*	
HP2	35	4	100%	4048	100%	3546	100%	3537			
PB1	27	4	100%	2333	100%	2130	100%	2508			
PB2	34	4	100%	4048	100%	3466	100%	3441			
PB4	29	2	100%	2148	100%	2253	100%	2123			
PB5	20	10	100%	1916	100%	2023	100%	2190			
PB6	40	30	100%	2019	100%	1893	100%	1913			
PB7	37	30	100%	4130	100%	4075	100%	4014			
PET2	10	10	100%	111	100%	105	100%	114			
PET3	15	10	100%	579	100%	512	100%	628			
PET4	20	10	100%	993	100%	813	100%	888	*		
PET5	28	10	100%	1467	100%	1503	100%	1617			
PET6	39	5	100%	6208	100%	5102	100%	5223	*		
PET7	50	5	100%	7534	100%	6359	99%	6316	*	*	
SENT01	60	30	100%	5910	100%	6328	100%	6210			
SENT02	60	30	100%	9548	100%	8965	100%	8409		*	
WEING1	28	2	100%	2513	100%	2450	100%	2658			
WEING2	28	2	100%	2394	100%	2223	100%	2403			
WEING3	28	2	100%	2118	100%	1620	100%	1695	*	*	
WEING4	28	2	100%	1726	100%	1628	100%	1937			
WEING5	28	2	100%	1563	100%	1416	100%	1500			
WEING6	28	2	100%	2843	100%	2394	100%	2693	*		*
WEING7	105	2	18%	14499	64%	12875	37%	17402	*	*	*
WEING8	105	2	86%	8349	100%	9198	94%	8828	*		*

Table 9: Comparison of the performance of the three ExGA variants, showing percentages correct and average number of FEs, and the differences that are statistically significant (as indicated by *). (Continued in Table 10.)

have a very high success rate across all instances in SAC'94. ExGA I solves all but two instances in all trials. ExGA IIb solves all but three instances in all trials, but has better success rates overall than ExGA I . ExGA IIa only fails to solve weing7 in all trials, and has a much higher success rate than the other two algorithms in that case. In summary, ExGA IIa has the highest success rate across all instances, but ExGA I and ExGA IIb require fewer FEs on average. ExGA I is significantly better than ExGA IIb on the smaller instances, but significantly worse on the larger instances. A graphical comparison of the three ExGA variants is shown in Figure 2.

Table 12 compares the performance of ExGA IIa to the other algorithms in the literature reviewed earlier. ExGA IIa was chosen because it shows the highest success rate across all instances and is the computationally least expensive of the three ExGA variants. ExGA IIa performs better on almost all instances compared to the other approaches, including the most promising approach, EGA. The only instance ExGA IIa performs worse on is weing7, which proved difficult for all algorithms except EGA. The cost of repair for EGA is similar to that of ExGA I, however, and it follows that ExGA IIa is computationally more efficient. Finally, it should be noted that ExGA I and IIb are also better across most instances compared to the other approaches as they have a similar success rate as ExGA IIa.

The only instance not solved consistently by ExGA IIa within the allowed limit of FEs was weing7. It is instructive to establish whether this poor performance is simply due to the limit imposed on the FEs,

Instance	n	m	ExGA I		ExGA II a		ExGA II b		I-IIa	I-IIb	IIa-IIb
WEISH01	30	5	100%	2684	100%	2789	100%	2942			
WEISH02	30	5	100%	2528	100%	2655	100%	2620			
WEISH03	30	5	100%	1641	100%	2029	100%	2248	*	*	
WEISH04	30	5	100%	1014	100%	1451	100%	1562	*	*	
WEISH05	30	5	100%	810	100%	940	100%	1086		*	
WEISH06	40	5	100%	3241	100%	3787	100%	3625	*	*	
WEISH07	40	5	100%	2666	100%	3373	100%	3395	*	*	
WEISH08	40	5	100%	2987	100%	3304	100%	3178			
WEISH09	40	5	100%	2348	100%	2707	100%	2600	*	*	
WEISH10	50	5	100%	4314	100%	5728	100%	5468	*	*	
WEISH11	50	5	100%	2672	100%	3896	100%	3692	*	*	
WEISH12	50	5	100%	3666	100%	4152	100%	4130	*	*	
WEISH13	50	5	100%	3281	100%	4149	100%	4303	*	*	
WEISH14	60	5	100%	4131	100%	5409	100%	5076	*	*	
WEISH15	60	5	100%	4245	100%	5405	100%	5055	*	*	
WEISH16	60	5	100%	4732	100%	5323	100%	5195	*	*	
WEISH17	60	5	100%	3624	100%	4276	100%	4078	*	*	
WEISH18	70	5	100%	6640	100%	6798	100%	6289			
WEISH19	70	5	100%	4928	100%	5208	100%	4764			*
WEISH20	70	5	100%	6016	100%	7219	100%	6336	*		*
WEISH21	70	5	100%	5232	100%	5584	100%	5120	*		*
WEISH22	80	5	100%	6619	100%	6492	100%	5773		*	*
WEISH23	80	5	100%	5796	100%	6266	100%	5593			*
WEISH24	80	5	100%	7837	100%	7890	100%	7215			*
WEISH25	80	5	100%	7771	100%	8398	100%	7320			*
WEISH26	90	5	100%	7369	100%	7962	100%	6986			*
WEISH27	90	5	100%	6102	100%	6521	100%	5935	*		*
WEISH28	90	5	100%	7032	100%	7671	100%	6707	*		*
WEISH29	90	5	100%	7130	100%	7075	100%	6314		*	*
WEISH30	90	5	100%	7182	100%	7914	100%	7168	*		*

Table 10: Comparison of the performance of the three ExGA variants, showing percentages correct and average number of FEs, and the differences that are statistically significant (as indicated by *). (Continued from Table 9.)

Variant	ExGA I		ExGA IIa		ExGA IIb	
	FEs	%	FEs	%	FEs	%
ExGA I			$+7$ -18	$+2$ -0	$+6$ -14	$+2$ -0
ExGA IIa	$+18$ -7	$+0$ -2			$+12$ -1	$+0$ -2
ExGA IIb	$+14$ -6	$+0$ -2	$+1$ -12	$+2$ -0		

Table 11: Quantitative comparison of ExGA I, IIa and IIb showing how many instances each algorithm is significantly better ($+$) or worse ($-$) in terms of success rate and number of FEs required. To be read top-down.

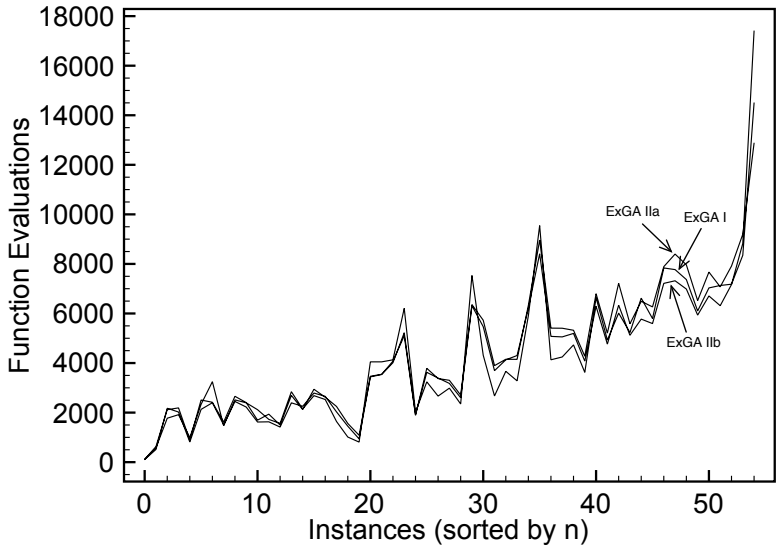


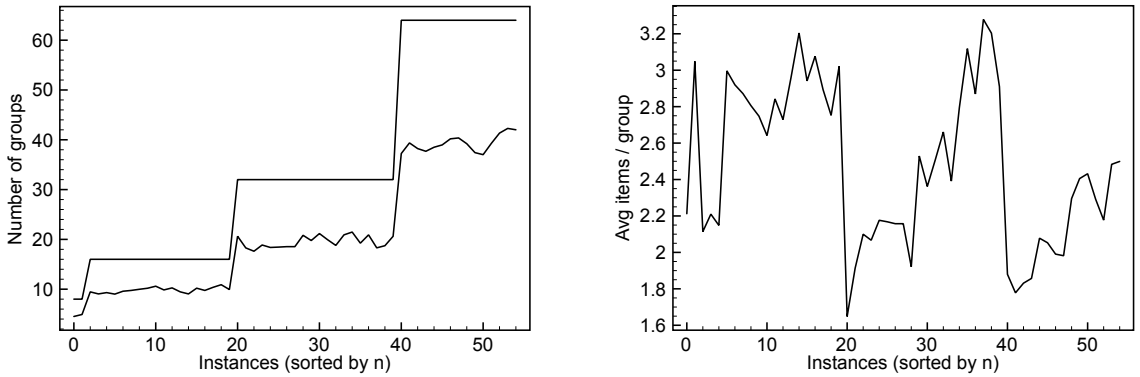
Figure 2: A comparison of the average number of function evaluations required by ExGA I, IIa and IIb to reach the global optimum.

	ExGA IIa			Differences			
Instance	Solved	FunEvals	Average	SGA	HGA	EGA	2-MGA
hp2	100%	3546	3186	-	-	±0%	±0
pb6	100%	1893	776	-	-	±0%	+45.8
pb7	100%	4075	1035	-	-	+0.4	+2
pet3	100%	512	4015	+17%	±0%	±0%	-
pet4	100%	813	6120	+67%	+6%	±0%	-
pet5	100%	1503	12400	+67%	±0%	±0%	-
pet6	100%	5102	10618	+96%	+40%	+1%	-
pet7	100%	6359	16537	+99%	+54%	±0%	+50.4
sent01	100%	6328	7772	+95%	+25%	±0%	+2.2
sent02	100%	8965	8722	+98%	+61%	+31%	+1.6
weing7	64%	12875	1095422	+64%	+24%	−36%	+694.9
weing8	100%	9198	624319	+94%	+71%	+27%	+691.2
weish12	100%	4152	6339	-	-	±0%	±0
weish17	100%	4276	8633	-	-	±0%	±0
weish21	100%	5584	9074	-	-	±0%	±0
weish22	100%	6492	8947	-	-	±0%	±0
weish25	100%	8398	9939	-	-	±0%	±0
weish29	100%	6521	9410	-	-	±0%	+206.8
Instances worse				0	0	1	0
Instances equal				0	2	13	6
Instances better				9	7	4	8

Table 12: Comparison of ExGA IIa with the earlier approaches in the literature, in terms of how frequently each instance was solved across all trials. The only exception is the comparison with 2-MGA where the average fitness of the best individual in the final generation is used. A positive/negative difference indicates by how much ExGA IIa performed better/worse than the approach named.

Limit (FEs)	% solved
20000	64
50000	87
100000	97
124812	100

Table 13: The performance of ExGA IIa on weing7 depends on the number of FEs allowed, and increases to 100% as the limit on the FEs is increased.



(a) (b)

Figure 3: The number of groups evolved in ExGA IIb compared to the maximum (a), and the average number of items per group (b).

or whether the algorithm is truly incapable of producing a consistent performance in this case. If the limit on the FEs is increased to infinity, the algorithm does manage to solve all instances with an average of only 24,778 FEs. The worst case requires 124,812 FEs. It is thus evident that the algorithm does not get stuck at local optima for very long, but merely requires additional resources to solve this particular instance. The success rates for different FE limits are shown in Table 13.

Finally, the adaptive attributes of ExGA IIb may be used to investigate the impact of the segmented encoding and the distribution of items across the groups. The maximum number of groups possible in ExGA IIb is restricted to the power of 2 closest to, but less than, n. If, for example, an instance contains 80 items, a binary sequence of 6 bits is chosen for a maximum of $2^6 + 1 = 65$ groups (because there has to be at least one group). Figure 3(a) shows the maximum number of groups possible (upper line) and the actual number of groups in the final solution (lower line). The result is fairly consistent with respect to problem size, and it can be seen in Figure 3(b) that the number of items per group averages approximately 2.5 items. There are some clear variations, however, indicating that the optimal average number of items per group is indeed an instance specific property.

5 Conclusions

This paper has presented two novel nature inspired genetic algorithms (GAs) that address two representative classes of hard packing problems: the bin packing problem (BPP) and the highly constrained multiple knapsack problem (MKP). Empirical results indicate that these algorithms are likely to provide improved performance across a range of real world problems.

The Exon Shuffling GA (ESGA) has been applied successfully to a benchmark suite of hard BPPs. This algorithm uses a biologically inspired group based encoding to achieve an appropriate modular-

isation of the search space which allows the assignment of cost values to individual sub-solutions. The crossover operator is based loosely upon the theory of exon shuffling, and combines parental sub-solutions (bins) in a greedy fashion to produce a single offspring. A control sequence is used to introduce noise during the crossover event to prevent stagnation at local optima. The resulting algorithm has a very low number of parameters, but produces a very high success rate, solving 8 out of the 10 most difficult problem instances. The performance of ESGA on the other benchmark instances is similar, resulting in a very good robust performance overall compared with existing approaches in the literature.

The Exonic GA (ExGA) was proposed for more highly constrained problems, such as the MKP, and three different variants of the algorithm were implemented and compared on a large set of benchmark problems. This algorithm utilises a biological RNA editing inspired repair function that relies upon the relative order of items (left to right, right to left). The actual order of items is adaptive and evolves during the evolutionary process. This is a partial ordering only, with items of equal priority being selected at random during the repair process. This adaptive approach offers at least two advantages over the other approaches in the literature: first, instance specific knowledge is not required. Secondly, items of low value-weight ratios may be part of the globally optimum solution; a rigid approach will always attempt to exclude such items if the solution is unfeasible, but the ExGA may place such items in a position such that it will never be considered for removal (or always be considered for inclusion). It was shown that ExGA has excellent performance compared to other algorithms in the literature.

In summary, we have demonstrated how nature inspired techniques can lead to improved GAs for two broad classes of hard packing problems of importance for many real world applications.

5.1 Future Work

There remain numerous variations of our algorithms still to explore, such as improved population initialization procedures, the incorporation of repair procedures in the ESGA, and further variations of the ExGA encodings. There is also much scope for future work that addresses whether these algorithms may be applied successfully to other constrained optimisation problems, such as the maximum clique or the degree-constrained minimum spanning tree problem. Both of our algorithms exploit domain specific information, but do not exploit instance specific attributes. This should, in general, allow their application to other problems that exhibit similar structural properties. Finally, although both algorithms exhibit a very high success rate across all instances tested, there are some cases (hard2 and hard3 for the ESGA, and weing7 for the ExGA) where the performances of the algorithms deteriorates significantly. This issue will be addressed in the near future, and it is expected that the abstraction of additional phenomena from molecular genetics may help to improve performance further.

References

Alvim, A. C. F., C. C. Ribeiro, F. Glover, and D. J. Aloise (2004). A hybrid improvment heuristic for the one-dimensional bin packing problem. *Journal of Heuristics 10*, 205–229.

Atlan, H. (2003). The living cell as a paradigm for complex natural systems. *ComPlexUs 1*, 1–3.

Bass, B. L. (1997). RNA editing and hypermutation by adenosine deamination. *Trends in Biochemical Sciences 22*(5), 157–162.

Bean, J. C. (1994). Genetic algorithms with random keys for sequencing and optimization. *ORSA Journal on Computing 6*, 154–160.

Bein, W., J. R. Correa, and X. Han (2008). A fast asymptotic approximation scheme for bin packing with rejection. *Theoretical Computer Science 393*(1–3), 14–22.

Blake, C. C. F. (1978). Do genes-in-pieces imply proteins-in-pieces? *Nature 273*, 267.

Coffman, E. G., M. R. Garey, and D. S. Johnson (1978). An application of bin-packing to multimachine scheduling. *Journal of Computing 7*, 1–17.

Cotta, C. and J. M. Troya (1997). A hybrid genetic algorithm for the 0-1 multiple knapsack problem. In G. D. Smith, N. C. Steele, and R. F. Albrecht (Eds.), *Proceedings of the 1997 International Conference on Artificial Neural Networks and Genetic Algorithms*, pp. 250–254. Springer.

Daida, J. M., S. P. Yalcin, P. M. Litvak, G. A. Eickhoff, and J. A. Polit (1999). Of metaphors and darwinism: Deconstructing genetic programming's chimera. In P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala (Eds.), *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, Volume 1, pp. 453–462.

Falkenauer, E. (1996). A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics 2*, 5–30.

Fleszar, K. and K. S. Hindi (2002). New heuristics for one-dimensional bin-packing. *Computers & Operations Research 29*, 821–839.

Freeland, S. (2003). Three fundamentals of the biological genetic algorithm. In R. Riolo and B. Worzel (Eds.), *Genetic Programming, Theory and Practice*, pp. 303–311. Kluwer Academic Publishers.

Gilbert, W. (1978). Why genes in pieces? *Nature 271*(5645), 501.

Grefenstette, J. J. (1987). Incorporating problem specific knowledge into genetic algorithms. In L. Davis (Ed.), *Genetic algorithms and simulated annealing*, pp. 42–60. Morgan Kaufmann Publishers.

Gréwal, G., S. Coros, D. Banerji, and A. Morton (2006). Comparing a genetic algorithm penalty function and repair heuristic in the DSP application domain. In *Artificial Intelligence and Applications*, pp. 31–39.

Gupta, J. N. D. and J. C. Ho (1999). A new heuristic algorithms of the one-dimensional bin-packing problem. *Production Planning & Control 10*(6), 598–603.

Herbet, A. and A. Rich (1999). RNA processing and the evolution of eukaryotes. *Nature Genetics 21*, 265–269.

Higuchi, M., F. N. Single, M. Kohler, B. Sommer, R. Sprengel, and P. H. Seeburg (1993). RNA editing of AMPA receptor subunit GluR-B: A base-pared intron exon structure determines position and efficieny. *Cell 75*, 1361–1370.

Holland, J. H. (1975). *Adaptation in Natural and Artifical Systems*. Ann Arbor, MI: University of Michigan Press.

Iima, H. and T. Yakawa (2003). A new design of genetic algorithm for bin packing. In *The 2003 Congress on Evolutionary Computation*, Volume 2, pp. 1044–1049.

Jun, Y., L. Xiande, and H. Lu (2003). Evolutionary game algorithm for multiple knapsack problem. In *Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology*, pp. 424–427. IEEE.

Kao, C.-Y. and F.-T. Lin (1992). A stochastic approach for the one-dimensional bin-packing problems. In *Systems, Man and Cybernetics 1992*, Volume 2, pp. 1545–1551.

Khuri, S., T. Bäck, and J. Heitkötter (1994). The zero/one multiple knapsack problem and genetic algorithms. In E. Deaton, D. Oppenheim, J. Urban, and H. Berghel (Eds.), *Proceedings of the 1994 ACM Symposium of Applied Computation proceedings*, pp. 188–193. ACM Press.

Kimbrough, S., M. Lu, D. Wood, and D. J. Wu (2002). Exploring a two-market genetic algorithm. In *Proceedings of the 2002 Genetic and Evolutionary Computation Conference*, California, pp. 415–422. Morgan Kaufmann.

Kolkman, J. A. and W. P. C. Stemmer (2001). Directed evolution of proteins by exon shuffling. *Nature Biotechnology 19*, 423–428.

Langton, C. G. (1992). Artificial life. In L. Nadel and D. Stein (Eds.), *Lectures in Complex Systems*, pp. 189–241. Addison-Wesley Publishing Company.

Maynard Smith, J. and E. Szathmãry (1999). *The Origins of Life: From the Birth of Life to the Origins of Language*. Oxford University Press.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. MIT Press.

Patthy, L. (2003). Modular assembly of genes and the evolution of new functions. *Genetica 118*, 217–231.

Rohlfshagen, P. and J. A. Bullinaria (2006). An exonic genetic algorithm with RNA editing inspired repair function for the multiple knapsack problem. In *Proceedings of the 2006 UK Workshop on Computational Intelligence*, Leeds, UK, pp. 17–24.

Rohlfshagen, P. and J. A. Bullinaria (2007). A genetic algorithm with exon shuffling crossover for hard bin packing problems. In *Proceedings of the 2007 Genetic and Evolutionary Computation Conference*, pp. 1365–1371. ACM Press.

Roy, S. W. (2003). Recent evidence for the exon theory of genes. *Genetica 118*(2–3), 251–266.

Scholl, A., R. Klein, and C. Juergens (1997). Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research 24*(7), 627–645.

Silvano, M. and T. Paolo (1990). *Knapsack Problems, Algorithms and Computer Implementations*, Chapter Bin-packing problem, pp. 221–245. England: John Wiley and Sons Ltd.

Spector, L. (2003). An essay concerning human understaning of genetic programming. In R. Riolo and B. Worzel (Eds.), *Genetic Programming Theory and Practice*, pp. 11–22.