ExGA II: An Improved Exonic Genetic Algorithm for the Multiple Knapsack Problem

Philipp Rohlfshagen
School of Computer Science
University of Birmingham
Birmingham B15 2TT, United Kingdom
P.Rohlfshagen@cs.bham.ac.uk

John A. Bullinaria
School of Computer Science
University of Birmingham
Birmingham B15 2TT, United Kingdom
J.A.Bullinaria@cs.bham.ac.uk

ABSTRACT

ExGA I, a previously presented genetic algorithm, successfully solved numerous instances of the multiple knapsack problem (MKS) by employing an adaptive repair function that made use of the algorithm's modular encoding. Here we present ExGA II, an extension of ExGA I that implements additional features which allow the algorithm to perform more reliably across a larger set of benchmark problems. In addition to some basic modifications of the algorithm's framework, more specific extensions include the use of a biased mutation operator and adaptive control sequences which are used to guide the repair procedure. The success rate of ExGA II is superior to its predecessor, and other algorithms in the literature, without an overall increase in the number of function evaluations required to reach the global optimum. In fact, the new algorithm exhibits a significant reduction in the number of function evaluations required for the largest problems investigated. We also address the computational cost of using a repair function and show that the algorithm remains highly competitive when this cost is accounted for.

Categories and Subject Descriptors

G.3 [**Mathematics of Computing**]: Probability and Statistics – Probabilistic Algorithms

General Terms

Algorithms, Experimentation, Performance

Keywords

Genetic algorithms, multiple knapsack problem, repair functions, molecular genetics

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

1. INTRODUCTION

GAs (see [5], [9]) are abstract implementations of evolutionary systems designed to search difficult and usually highly rugged search spaces. A population of potential solutions is maintained which evolves by means of selection, crossover and mutation to explore promising regions of the search space. Traditionally, GAs are inspired by the field of population genetics, although over time many additions have been suggested that follow more closely the principles of molecular genetics. In [10], we presented an adaptive GA, labelled ExGA I, that was inspired by molecular genetics utilising a modular encoding and adaptive repair approach that is loosely analogous to RNA editing. Here we continue the trend of using molecular genetics as inspiration to improve upon the accuracy and reliability of ExGA I. The algorithm presented in this paper, labelled ExGA II, exhibits a very high degree of success on a large set of benchmark problems. The results improve on the success rate of ExGA I in several cases without an overall increase in the number of function evaluations required to reach the global optimum.

Furthermore, we also consider the computational cost implications of using a repair function in the algorithm. The stochastic nature of crossover and mutation may produce offspring that encode solutions violating one or more constraints even if both parents are feasible. This is a common phenomena of constraint optimisation where care has to be taken that an encoding represents a valid solution to the problem of interest. This issue is usually addressed by using specialised variation operators, penalty functions or repair procedures. A direct comparison of the latter two has shown that repair-based approaches are superior to penalty-based ones for the MKS [10]. Here we show that this result still holds if the cost of repair is properly accounted for. We approximate the cost of repair in terms of function evaluations and find a similar degree of success. This allows a direct comparison to other approaches that do not use an explicit repair to cope with the problem's constraints.

The remainder of this paper is organised as follows: Section 2 outlines the multiple knapsack problem in more detail. Section 3 describes ExGA I including a discussion of the algorithm's performance followed by a description of how the individual shortcomings have been addressed to yield ExGA II in section 4. The computational cost of the repair function is discussed in section 5. The experimental setup is described in section 6, and the results and analysis are presented in section 7. The final section concludes this paper and addresses potential future work.

2. THE MULTIPLE KNAPSACK PROBLEM

The multiple knapsack problem (MKS) is a widely studied combinatorial optimisation problem that has several direct counterparts in industry, such as the stock cutting problem. Well-established benchmarks for this NP-complete problem may be found online allowing a direct comparison to other approaches in the literature. The MKS is a generalisation of the single knapsack problem: Its objective is to fill a series of m knapsacks each of capacity c_i with any number of n items maximising the items' combined value. Each item has a weight w and a value v and is placed in all knapsacks simultaneously. The items' weights depend on the knapsacks and the sum of weights of all items considered for inclusion may not exceed any of the capacities. More formally, we wish to maximize the fitness

$$\sum_{i=1}^{n} v_i x_i \text{ subject to } \sum_{i=1}^{n} w_{ij} x_i \leq c_j$$

for $j = 1, 2, \ldots, m$ and where $x_i \in \{0, 1\}$.

We make use of the SAC'94 library of benchmark problems[1] and test our algorithm on the entire suite. The suite contains 55 problem instances ranging in size from 15-105 objects and 2-30 knapsacks. The algorithms are judged by their success rate at finding optimal solutions and the number of function evaluations required to reach the global optimum: The success rate measures the reliability of the algorithm as the percentage of times a problem instance has been solved in any given number of trials. This measure is complemented by the average number of function evaluations required to reach the global optimum, indicating the computational cost of the algorithm. The MKS is very popular in the GA community as it may be encoded using a binary string: Each bit in the binary vector indexes an item and the state of the bit (0 or 1) indicates whether the item is to be included in the solution.

3. EXGA I

As mentioned above, the design of the original ExGA I algorithm was inspired by certain aspects of molecular genetics: The majority of eukaryotic genes are composed of exons and introns in an alternating fashion. Once a gene is transcribed from DNA to RNA, it undergoes a processing step that removes all non-coding regions (introns). The remaining segments, exons, are finally translated into a polypeptide of amino acids (a protein). The modular composition of genes is essential to the evolution of complexity and numerous processes are in place that affect the traditional pathway of expression. As such, processes like alternative splicing or RNA editing may alter the RNA in several different ways to produce a variety of distinct proteins from the same underlying strand of DNA. The repair procedure of ExGA I was loosely inspired by RNA editing which selectively targets individual nucleotides for modification prior translation. Mitochrondrial DNA, for example, may contain premature stop codons that would cause the strand's degradation upon translation [4]. RNA editing selectively targets these premature stop codons to restore the DNA's functionality. This form of natural repair alongside the modular composition of eukaryotic genes inspired the design of ExGA I.

[1]This library is available online at `http://elib.zib.de/pub/Packages/mp-testdata/ip/sac94-suite/`.

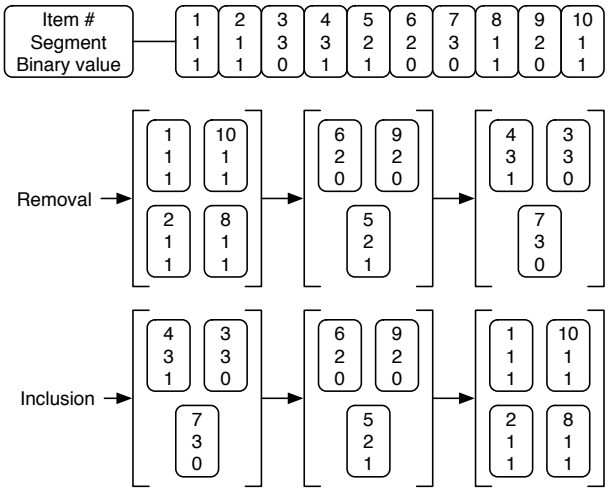

Figure 1: A schematic view of the exonic encoding and repair approach: Each encoding may be viewed consisting of triplets each of which contains the item to be represented, its segment number and binary value. This information is then used to repair the encoding if necessary. Groups are prioritised but items in each group are chosen at random for removal/inclusion.

The basis of ExGA I is a modular encoding defined in terms of a number of "segments" as shown in figure 1: Conceptually, each knapsack item is represented as a triplet composed of the item's number, its segment number, and a binary value indicating whether that item is to be included in the solution (1) or not (0). These triplets are encoded as movable elements and are thus free to move from one segment to another allowing the grouping of variables. The encoding is thus composed of stationary segments whose contents are dynamic. The encoding's dynamics are used to repair infeasible solutions using a two-phase repair technique: First, items are removed (invert 1 to 0) until all constraints have been satisfied. This phase proceeds by considering the segments in order from left to right choosing items within the same segment at random. The dynamics of the encoding therefore evolve a partial ordering in which items in different segments are chosen deterministically while all items within the same segment are chosen stochastically. In a similar manner, the second phase of the repair procedure attempts to add items (invert 0 to 1), if possible, proceeding in opposite direction, from right to left. The repair approach is also shown in figure 1. It should be noted that the exclusion phase considers as many segments as necessary to restore the encoding's feasibility. If the encoding is feasible prior to repair, the exclusion phase is skipped. The inclusion phase, on the other hand, considers all of the segments. This technique produces tightly packed feasible solutions in a manner similar to [6] without using instance specific knowledge. This adaptive approach has further advantages over a static repair procedure as discussed in [10].

The remaining algorithmic framework that surrounds the aforementioned encoding employs standard GA processes throughout, slightly adapted to the MKS. A steady-state

update scheme is used where a single parent is selected at random and the second parent is chosen as the individual in the population that is maximum hamming distance from the first. Although this concept is counterintuitive from a biological point of view (crossover may only occur amongst homologous species), it slows the population's loss of diversity. Standard uniform crossover is used followed by a mutation operator that either inverts a bit or moves an item from one segment to another. The offspring generated compete in a tournament with a randomly chosen parent and are subsequently placed into the population if their fitness is at least as good as that of their competitor (i.e. one of the parents).

ExGA I produces results of quality equal to or superior to other evolutionary approaches in the literature [10]. In particular, it shows improvement over the EGA algorithm of Jin, Xiande and Lu [6], which uses a static version of the same repair procedure that utilises instance specific value-weight ratios to repair infeasible solutions. This superiority was demonstrated on on a subset of 18 problem instances chosen from the SAC'94 library. The success rate for ExGA I is 100% for most of these instances, although a significant drop in performance is evident in a few cases. This prompted further analysis of the algorithm's behaviour on these instances of poor behavior, and several improvements to the algorithm's design were identified as discussed in the next section.

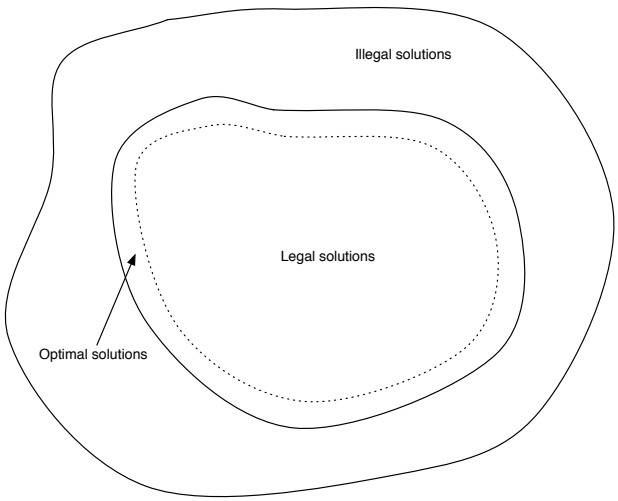

Figure 2: A schematic visualization of the search space from the point of view of feasibility. Solutions violating any of the imposed constraints are illegal. Optimal solutions are likely to be situated very close to the boundary separating the legal and illegal solutions, as there the waste of capacity is minimised.

4. EXGA II

A full analysis of the ExGA I results reveals several issues affecting its performance when used across a wide range of different problem instances. The major issue was found to be related to the population's diversity. The repair procedure is very greedy and limits the search to a small subset of the search space, as indicated in figure 2. This leads to a significant proportion of offspring being generated by crossover that match an encoding already present in the population. A simple checking procedure that removes these duplicates slows down the loss of diversity within the population and subsequently increases the algorithm's performance. However, there is an obvious computational cost associated with this approach: While it may be verified efficiently whether an individual is already in the population, any duplicates are discarded after the costly repair process has already been carried out. Moreover, since the repair process is partly stochastic, it is impossible to compare individuals prior to repair. The algorithm's improvement in performance is thus associated with an increased computational cost in repair which needs to be accounted for in the comparison of the two algorithms.

It proved difficult to identify reliable predictors as to which problem instances the ExGA I algorithm would perform poorly on. However, tests revealed how different instances react differently to the repair procedure. Some of the more difficult instances were solved more efficiently by allowing the extent of the repair to be adaptive. This may be achieved by using two binary control sequences, one for each phase of the repair. The binary control sequences are part of the encoding and are thus subject to crossover and mutation allowing their adaptation throughout the algorithm's run. Each sequence is translated to its integer value, which subsequently dictates the number of segments to be considered during repair as shown in figure 3: In ExGA I, the exclusion phase, which proceeds from left to right, considers individual segments in turn until feasibility is obtained. The number of segments considered is thus determined by the encoding's number of constraint violations. Now, the control sequence may force the repair process to consider segments in addition to those necessary. The exclusion phase will consider the y first segments where y is the integer value of the first control sequence. For those segments, items are not considered one by one but instead all items' binary values are set to 0. In case the encoding remains unfeasible after all y segments have been targeted, the standard exclusion phase proceeds as normal from segment $y+1$. The same principle applies to the inclusion phase which only considers the first z segments where z is the integer value of the second control sequence. In this case, however, items are considered one by one. The use of a binary control sequence implies that the number of exons used has to be to the power of 2. This limitation may be overcome easily by allowing the control sequence to express a percentage of exons to be considered rather than an absolute number. In this work, however, we use the former case and instead develop a meaningful relationship between n and the number of exons as explained next.

It is very difficult to design a generic algorithm that performs equally well across a wide range of different problem instances. Some instances react very differently to certain parameter settings than others. A large body of research addresses the issue of using adaptive parameters that evolve during the algorithm's execution in response to the instance being solved (e.g. [1]). Here we do not consider such adjustments, but investigate whether there is a meaningful relationship between the number of items and number of exons that should be used. Systematic experimentation reveals that it is beneficial to have approximately 2 items per segment. As the number of segments is a number to the power

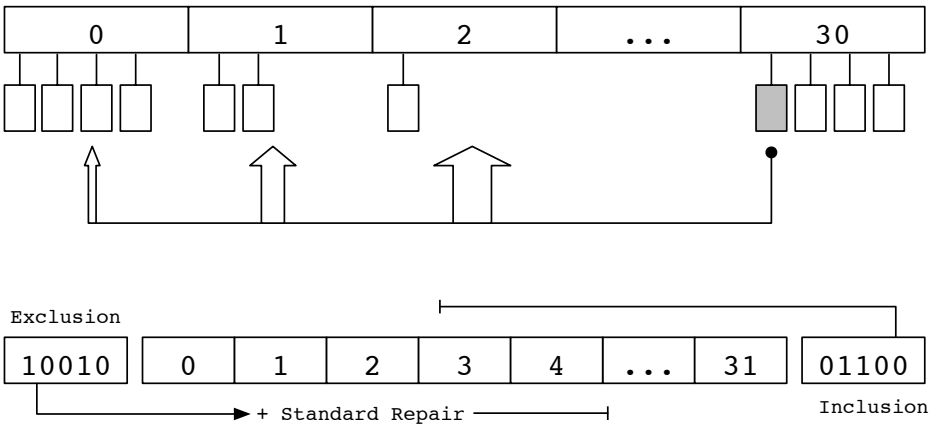

Figure 3: ExGA II introduces two nature inspired additions to ExGA I. Top: The mutation operator has a bias to consider for inclusion those segments containing the least number of items first. Bottom: The repair procedure is influenced by two adaptive control sequences that determine the number of segments to be considered for exclusion and inclusion.

of two, we take the closest approximation possible. An instance with 20 items thus has 7 exons while an instance with 100 items has 63.

The final addition introduces a bias to the mutation operator. It has been shown that mutations in natural systems have a positional bias and that the genetic code, a mapping used during translation, has adapted according to this bias (see [3]). We found it useful to bias the mutation operator that moves items from one segment to another: If a variable is chosen to be moved to an alternative segment, the new mutation operator considers segments in non-decreasing order of their content: Items are always placed into segments containing the least number of items with some probability allowing to skip a segment. This bias generates a more even spread of items across all segments and hence reduces the randomness of the repair procedure. Furthermore, the control sequences work more efficiently if all segments contain a similar number of items.

In summary, there are four fundamental changes between ExGA I and II: Two changes that affect the algorithm's overall design are the detection of duplicates in the population and the relationship between the number of items n and the number of exons. The additional two changes inspired by molecular genetics are the use of control sequences to make the repair procedure more adaptive, and the use of a biased mutation operator that facilitates a more even spread of items across all exons. It should be noted that almost all other parameter settings are identical to the experiments carried out in [10]. However, it is possible that the improved performance of ExGA II is due to a better choice of parameters alone, or that a single extension is solely responsible for an increase in the algorithm's performance. This was checked and found not to be the case: Systematic elimination of any of the aforementioned changes always caused a degradation in performance on one or more problem instances. We therefore conclude that these additions are mutually required to produce the performance improvements shown in section 7.

5. THE COST OF REPAIR

As mentioned in section 1 there are several ways of dealing with constraints: Penalty based approaches, in which constraint violations map to reductions in solution fitness, have the advantage that standard GAs may be used without the need for alteration. However, the design of the penalty function may greatly affect the algorithm's ability to explore the feasible regions of the search space. Specialised variation operators or repair functions, on the other hand, restrict the search to the feasible regions and also allow the use of problem specific knowledge. Here we address the computational overhead of using a repair function to determine whether our results still hold if this cost is properly accounted for. The remainder of this section explains how this cost can be approximated.

The fitness of an individual is calculated as the sum of values of all items indicated to be in the knapsacks. Each problem instance has n items and m knapsacks and the fitness may be obtained at a cost of $O(mn)$. A penalty may subsequently be applied to decrease an individuals fitness proportionally to the number or magnitude of constraint violations. The 2-phase repair procedure, on the other hand, considers each variable for modification to ensure the encoding's feasibility: First, the individual's fitness is calculated. This is followed by two loops considering every bit in turn and inverting it if possible. Every time an inversion occurs, the sum of weights has to be recalculated. The exclusion phase considers all bits equal to 1 while the exclusion phase considers all other bits. The total cost of repair is therefore $O(mn+n)$, and the total cost of an individual's evaluation is $O(2mn+n)$. However, this is an upper bound: The removal phase stops as soon as enough items have been removed to make the encoding feasible and the inclusion phase may be constrained by the control sequence. Our experiments confirm this as the actual number of weight recalculations is much lower than the given bound. In order to get a realistic estimate on the true cost of repair, an explicit counter is used during the experiments. The resulting count is divided by mn to obtain an approximate cost in terms of function

evaluations which are subsequently added to give the total effective number of function evaluations required.

6. EXPERIMENTAL SETUP

The experimental setup is almost identical to the one employed in [10]: A steady-state GA is used with a population of size 150. The probability of the uniform crossover operator is 0.9. The mutation operator either inverts a bit in the gene or control sequence or moves one item from one segment to another. The binary inversion occurs with a probability of $0.5/l$ where l is the length of the gene (number of items) plus the length of both control sequences. The probability of an item being shifted is $0.5/n$ where n is the number of items. The bias of the mutation operator considers segments in non-decerasing order of their content. A probability of 3/4 is used to skip a segment for consideration. The algorithm is run 100 times on each problem instance with the number of function evaluations limited to $20,000$. The number of exons used (which is a power of 2) is a simple function of n such that average content of each exon is closest to 2 items. This, in conjunction with the biased mutation operator, allows for the stochasticity of the repair function to be controlled. For each problem instance, we note the number of times this instance has successfully been solved within the upper limit on the number of function evaluations. We also note the computational cost, in terms of function evaluations, required on average to do so.

7. RESULTS AND ANALYSIS

The results of all our experiments are presented in table 1. ExGA II has a very high degree of success, solving all but two instances in all 100 trials. Furthermore, ExGA II improves on all 4 instances where ExGA I failed to reach a success rate of 100%, although Weing7 and Weing8 remain below 100% at 83% and 95% respectively. Nevertheless, in addition to a greater reliability, ExGA II also achieves a significant reduction in the number of function evaluations required in 20 of the 55 instances. On the other hand, ExGA II requires significantly more function evaluations in 12 cases. Interestingly, the performance of the two algorithms seems to be related to the number of variables in the problem instance. ExGA II outperforms ExGA I as n increases and we see a distribution of significant reductions in function evaluations that is skewed towards higher values of n. This could indicate better scaling properties of ExGA II and further testing on larger problem instances is required to validate this trend. This is also consistent with the reasoning that the smaller and simpler test cases do not require any advanced computations, while the larger cases do.

Table 1 also includes two columns that explicitly address the cost of repair in terms of function evaluations. Using the measure as explained in section 5, the last two columns of the table show the performance in terms of success rate and number of function evaluations accounting for the cost of repair. Although there is an increase in the number of function evaluations required to reach the global optimum, the success rate remains at 100% across almost all instances. The only decreases occur for for cases that already show less than perfect performance: Weing7 and Weing8, with 13% and 5% reductions respectively. Nevertheless, this data confirms that the algorithm remains highly competitive despite the additional computational cost of repair.

Naturally, it is important to compare our algorithm against other approaches. In [10], we compared the performance of ExGA I against a number of other evolutionary algorithms in the literature that have also been tested on instances from the SAC'94 library as used in our study. The first approach due to Khuri et al. [7], SGA, is a simple GA with standard penalty function which penalises invalid encodings proportionally to their number of constraint violations. A similar approach is due to Kimbrough et al. [8] who used a 2-market GA, 2-MGA, that employs two phases, optimality improvement and feasibility improvement, to generate valid solutions of high quality. A greater success rate on the same set of instances is achieved using problem specific knowledge: Cotta et al. [2] suggest a hybrid GA, HGA, that uses a greedy construction heuristic making use of the item's value-weight ratios. The last approach in our comparison is the most successful one and uses a repair procedure closely related to the one proposed here: The evolutionary game algorithm, EGA, by Jun et al. [6] uses a two phase repair procedure that first removes items in increasing order of their value-weight ratios followed by a phase that includes items, if possible, in order of their decreasing value-weight ratios. A direct comparison of all these approaches, together with a random repair technique, Rep2, described in [10], is shown in table 4. We see that ExGA II performs better on almost all instances compared to the other approaches, including the most promising approach, EGA. The only instance of worse performance is on the weing7 problem instance, which proves difficult for all algorithms except EGA.

Finally, a graphical comparison of the computational costs is shown in figure 4. Predictably, the required number of function evaluations tends to increase as n increases, indicating that larger problem instances are solved with greater computational cost (although there are notable exceptions as evident in the peaks). It is interesting to see that ExGA II is slightly worse on the smaller instances, and better on the larger ones, while there is no significant difference for the majority of medium sized instances. Again, this may be explained by the fact that the additional features of ExGA II do not pay off if the problem is solved easily in the first place. The graph also indicates the better scaling properties of ExGA II with a slower increase in the number of function evaluations in proportion to n than ExGA I.

ExGA II shows a robust and reliable performance across all instances. However, two instances remain that are not solved 100%. It is interesting to establish whether this is solely due to the difficulty of the problem or a result of the limit imposed on the number of function evaluations. Table 3 shows the results using different limits and as it turns out, all instances of WEING7 may be solved if the limit is increased. Only a slight increase is required and the worst case is solved within 35,000 function evaluations. For WEING8, however, there is only a fractional increase in the number of successful trials. Despite a limit of 500,000 function evaluations, 2 trials remain unsuccessful. Nevertheless, a simple restart procedure is used to overcome this: If the global optimum has not been located within 20,000 function evaluations, the population is reinitialised. This allows for all trials to be successful.

An analysis of the distribution of items confirms that the biased mutation operator allows for an even spread across all segments. The notable exception is the leftmost segment, which in almost all cases, contains significantly more items

Instance	n	m	Optimum	I %	II %	I Avg	II Avg	S	II C %	II C Avg
FLEI	20	10	2139	100	100	2054.7	3122.42	-	100	3294.99
HP1	28	4	3418	100	100	3707.2	4314.6		100	4894.56
HP2	35	4	3186	100	100	5386.38	6104.34		100	6936.15
PB1	27	4	3090	100	100	2909.12	4055.52	-	100	4616.87
PB2	34	4	3186	100	100	5596.46	5966.7		100	6769.28
PB4	29	2	95168	100	100	3123.14	3479.68		100	4532.43
PB5	20	10	2139	100	100	2214.34	2852.36		100	3011.41
PB6	40	30	776	100	100	2515.72	2723.96		100	2784.09
PB7	37	30	1035	100	100	6288.24	5891.72		100	6007.81
PET2	10	10	87061	100	100	150.8	204.46	-	100	214.16
PET3	15	10	4015	100	100	705.6	1062.54	-	100	1105.44
PET4	20	10	6120	100	100	1450.86	1690.24	-	100	1766.94
PET5	28	10	12400	100	100	2193.42	2904.48	-	100	3018.97
PET6	39	5	10618	82	100	10888.84	7012.38	+	100	7706.72
PET7	50	5	16537	100	100	9918.24	9602.48		100	10424.45
SENT01	60	30	7772	100	100	9470.98	8032.94	+	100	8179.30
SENT02	60	30	8722	92	100	15680.6	11464.7	+	100	11637.29
WEING1	28	2	141278	100	100	3888.24	3944.14		100	4787.49
WEING2	28	2	130883	100	100	3484.64	3518.44		100	4335.70
WEING3	28	2	95677	100	100	2920.62	2669.46		100	3377.47
WEING4	28	2	119337	100	100	2327.44	3050.18	-	100	3858.61
WEING5	28	2	98796	100	100	2555.64	2499.06		100	3058.89
WEING6	28	2	130623	100	100	4211.92	3716.2	+	100	4546.49
WEING7	105	2	1095445	15	83	19438.68	15465.86	+	70	18091.00
WEING8	105	2	624319	90	95	14110.46	12091.46	+	90	15498.31
WEISH01	30	5	4554	100	100	3963.68	3376.26	+	100	3708.11
WEISH02	30	5	4536	100	100	3423.8	3730.48		100	4101.72
WEISH03	30	5	4115	100	100	2554.82	2804.54		100	3090.68
WEISH04	40	5	4561	100	100	1509.82	1934.28	-	100	2141.52
WEISH05	30	5	4514	100	100	1111.6	1675.6	-	100	1859.25
WEISH06	40	5	5557	100	100	4929.62	5085.86		100	5577.00
WEISH07	40	5	5567	100	100	4329.5	4717.74	-	100	5188.56
WEISH08	40	5	5605	100	100	4692.26	4632.32		100	5077.52
WEISH09	40	5	5246	100	100	3637.08	3863.9		100	4252.32
WEISH10	50	5	6339	100	100	6597.96	6144.16		100	6755.51
WEISH11	50	5	5643	100	100	4115.02	4716.66	-	100	5219.05
WEISH12	50	5	6339	100	100	5919.34	5677.68		100	6248.86
WEISH13	50	5	6159	100	100	5030.8	5273.22		100	5806.75
WEISH14	60	5	6954	100	100	6430.8	6450		100	7107.74
WEISH15	60	5	7486	100	100	6463.2	6457.18		100	7169.02
WEISH16	60	5	7289	100	100	7382.08	6926.04	+	100	7577.20
WEISH17	60	5	8633	100	100	5661.14	6476.74	-	100	6943.84
WEISH18	70	5	9580	100	100	10520.24	9144.02	+	100	9911.98
WEISH19	70	5	7698	100	100	7997.62	7087.9	+	100	7830.04
WEISH20	70	5	9450	100	100	9491.18	8689.68	+	100	9564.39
WEISH21	70	5	9074	100	100	8809.24	7916.42	+	100	8760.60
WEISH22	80	5	8947	100	100	10776.86	8500.5	+	100	9355.75
WEISH23	80	5	8344	100	100	9167.98	8649.38		100	9564.78
WEISH24	80	5	10220	100	100	12494.06	10167.58	+	100	11034.73
WEISH25	80	5	9939	100	100	11957.18	10448.68	+	100	11496.02
WEISH26	90	5	9584	100	100	12249.78	9705.28	+	100	10711.76
WEISH27	90	5	9819	100	100	10253.76	8786.56	+	100	9672.29
WEISH28	90	5	9492	100	100	11498.32	9617.54	+	100	10628.08
WEISH29	90	5	9410	100	100	11743.14	9361.74	+	100	10348.27
WEISH30	90	5	11191	100	100	11839.54	10942.72	+	100	11939.40

Table 1: All instances are listed by name and size (number of items n, number of knapsacks m) and optimal solution. I and II stand for ExGA I and ExGA II respectively and % indicates the percentage of solved trials. Avg indicates the average number of function evaluations required and column S indicates whether ExGA II requires significantly more (-) or less (+) function evaluations than ExGA I. The last two columns account for the cost of repair: The first column gives the success rate and the second column indicates the average number of function evaluations required.

	ExGA II			Differences				
Instance	Solved	FunEvals	Average	SGA	HGA	EGA	2-MGA	Rep2
hp2	100%	6104	3186	-	-	±0%	±0	±0%
pb6	100%	2724	776	-	-	±0%	+45.8	±0%
pb7	100%	5892	1035	-	-	+0.4	+2	±0%
pet3	100%	1063	4015	+17%	±0%	±0%	-	±0%
pet4	100%	1690	6120	+67%	+6%	±0%	-	±0%
pet5	100%	2905	12400	+67%	±0%	±0%	-	±0%
pet6	100%	7012	10618	+96%	+40%	+1%	-	+69%
pet7	100%	9603	16537	+99%	+54%	±0%	+50.4	+26%
sent01	100%	8033	7772	+95%	+25%	±0%	+2.2	+8%
sent02	100%	11465	8722	+98%	+61%	+31%	+1.6	+75%
weing7	83%	15466	1095435.1	+83%	+43%	−17%	+708.1	+83%
weing8	95%	12092	624158	+89%	+66%	+22%	+530.2	+93%
weish12	100%	5678	6339	-	-	±0%	±0	±0%
weish17	100%	6477	8633	-	-	±0%	±0	±0%
weish21	100%	7916	9074	-	-	±0%	±0	±0%
weish22	100%	8501	8947	-	-	±0%	±0	+31%
weish25	100%	10449	9939	-	-	±0%	±0	+3%
weish29	100%	9362	9410	-	-	±0%	+206.8	+5%
Instances worse				0	0	1	0	0
Instances equal				0	2	13	6	9
Instances better				9	7	4	8	9

Table 2: Comparing ExGA II to other evolutionary approaches in the literature: All approaches are compared in terms of how frequently an instance was solved across all trials. The only exception is the comparison to the 2-market GA where the average fitness of the best individual in the final generation is used to compare performance. A positive number indicates by how much ExGA II performed better than the approach named on top of the column. A negative number indicates an inferior performance of EXGA II.

Instance	Opt	%	Avg
Limit at 200,000 FE			
WEING7	1095445	100	16142.58
WEING8	624319	97	13149.57
Limit at 500,000 FE			
WEING8	624319	98	15130.20
Restart at 20,000 FE			
WEING8	624319	100	12668.02

Table 3: Increasing the imposed limit on the number of function evaluations (FE) allows to solve the remaining two instances, Weing7 and Weing8, to 100%.

than the remaining segments. This may be explained by the fact that the removal phase starts from the left, removing all items within a segment. As the exclusion phase has a direct impact on the inclusion phase, it is vital that items not being part of the global optimum are removed to make space for additional items. Furthermore, the positioning of items towards the far left make an inclusion during the second phase of repair highly unlikely. In some cases, there is a very slight increase in items per segment towards the right. However, as the inclusion phase may consider all segments (depending on the control sequence), an item is not necessarily required to be positioned towards the right to be included, although such trend would increase the item's likelihood of inclusion. The adaptation of the control sequences, which have a direct impact upon the repair process, should reflect these characteristics of distribution. This, alongside additional analysis of the distribution of items across segments, especially over time, and the impact of the mutation operator will be carried out in the near future.

8. CONCLUSION

Our previously presented algorithm, ExGA I [10], inspired by the modular composition of genes, has been improved by auxiliary inspirations drawn from the field of molecular genetics. The resulting algorithm, ExGA II, implements additional features that improve the algorithm's success rate and efficiency on a large benchmark set of multiple knapsack problems. These additions include the use of adaptive control sequences regulating the repair procedure and a biased mutation operator. The algorithm's overall performance indicates good scaling properties, although further testing on larger problem sets is required to fully validate this claim. The cost of using a repair function has been addressed and it has been shown that the gain of using such function exceeds its computational cost. The success rates are identical in all but two out of 55 cases when the additional computational cost of repair is taken into account.

Future work should address whether this algorithm may be applied successfully to other constraint optimisation problems such as the maximum clique or the degree-constrained minimum spanning tree problem. The mixture of deterministic and stochastic ordering seems a promising approach generating a diverse set of high quality solutions to constrained optimisation problems. Furthermore, it should be possible to abstract additional phenomena from molecular

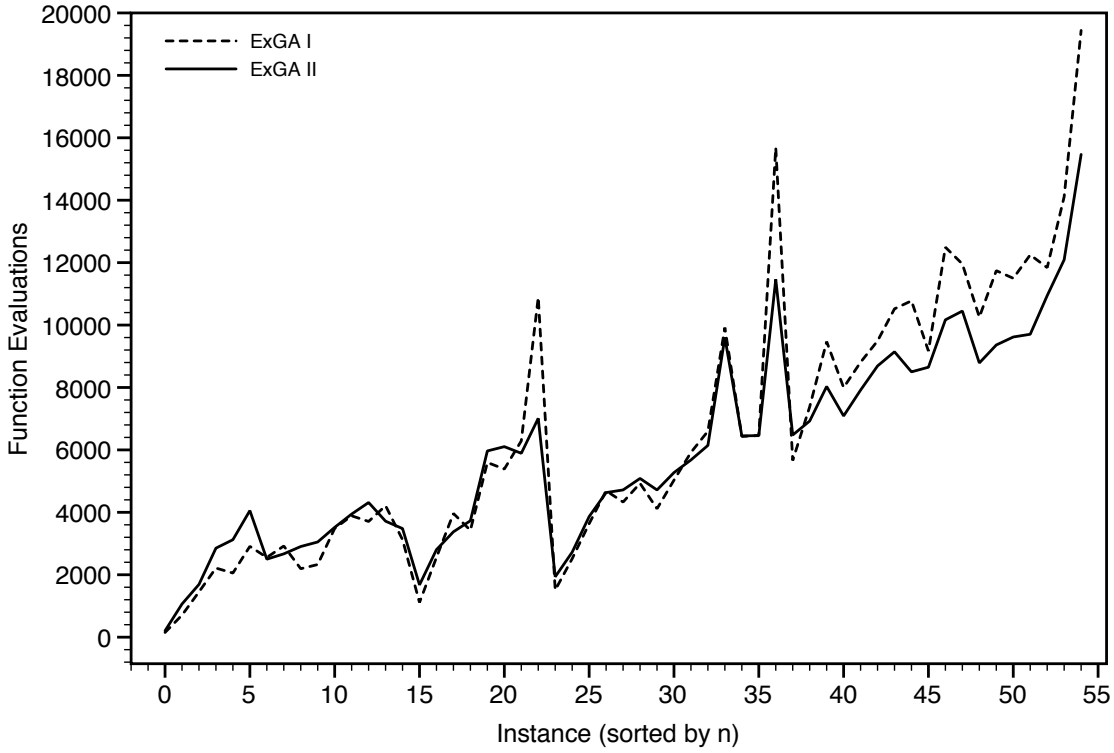

Figure 4: A comparison of the average number of function evaluations required by ExGA I and ExGA II to reach the global optimum: ExGA I performs slightly better on the smaller instances while ExGA II outperforms ExGA I as n increases. The is no significant difference between the two algorithms for the majority of medium sized instances.

genetics taking further advantage of the algorithm's encoding. For example, it should be possible to design a more sophisticated crossover operator and to introduce further mechanisms for maintaining diversity throughout the algorithm's run. The results presented here are promising and should encourage further exploration of molecular processes as inspiration to the design of evolutionary algorithms.

9. ACKNOWLEDGMENTS

This work was supported by a Paul and Yuanbi Ramsay scholarship.

10. REFERENCES

[1] T. Back, A. E. Eiben, and N. A. L. van der Vaart. An empirical study on gas "without parameters". In *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, pages 315 – 324. Springer, 2000.

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

[3] S. J. Freeland and L. D. Hurst. The genetic code is one in a million. *J. Mol. Evol.*, 47:238–248, 1998.

[4] A. Herbet and A. Rich. RNA processing and the evolution of eukaryotes. *Nature Genetics*, 21:265–269, 1999.

[5] J. H. Holland. *Adaptation in Natural and Artifical Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

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

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

[8] S. Kimbrough, M. Lu, D. Wood, and D. J. Wu. Exploring a two-market genetic algorithm. In *GECCO0-2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 415–422, California, 2002. Morgan Kaufmann.

[9] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1996.

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