An Exonic Genetic Algorithm with RNA Editing Inspired Repair Function for the Multiple Knapsack Problem

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

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

Abstract

The multiple knapsack problem is a well-known optimisation problem to which numerous different techniques have been proposed in the past. The most successful approaches to date rely on the instance specific value-weight ratios of individual items. In this paper, we suggest a simple genetic algorithm that produces competitive results without the use of such information. The proposed algorithm has been inspired by the modular composition of genes and allows for an order independent representation of the problem space: Each of the knapsack's items is encoded as a movable element allowing it to group with likewise elements in one of numerous segments. Segments are, in turn, used to repair invalid solutions in a fashion similar to RNA editing. Results across a selection of instances from a well-known library of benchmark problems are of the same quality as those obtained by methods relying explicitly on problem specific knowledge.

1 Introduction

Nature inspired approaches, most notably techniques based upon evolutionary principles, have been applied successfully to solve a wide variety of optimisation problems. Problems with constraints are of particular interest as they occur frequently in real-life situations. Constraints essentially fragment the search space into feasible and unfeasible regions and considerable efforts may be wasted exploring the unfeasible space. This is especially true for population based approaches such as genetic algorithms (GAs) where genetic operators act freely upon encodings allowing unrestricted exploration of the search space. This issue has been approached by the scientific community in two different ways: Penalty terms and repair functions. A penalty term penalises illegal solutions to steer the search towards feasibility. Repair functions, on the other hand, convert illegal solutions into legal ones restricting the search to the feasible region of the search space. Either approach has its advantages and disadvantages: Penalty terms may be added easily to the standard GA framework but potentially suffer from the feasibility problem where the final population of the evolutionary run consists solely of unfeasible solutions. Special care has to be taken to overcome this issues [4]. Repair functions, on the other hand, keep the entire population in the feasible regions of the search space at all times. They may, however, be computationally expensive and often seem to rely on the use of domain specific knowledge, and such knowledge may not always be available or easy to utilise.

In this paper, we suggest a simple genetic algorithm with an efficient repair function that achieves the same degree of success as problem specific approaches but without the need for such knowledge. This is accomplished by allowing the repair function to co-evolve and adapt during the evolutionary process. We use this algorithm to solve instances of the multiple knapsack problem (MKS), a highly constrained class of problems of both theoretical and practical interest (see [11]). The results show this approach outperforming other techniques from the literature.

The paper is structured as follows: Section 2 describes the multiple knapsack problem followed by a comparison of penalty terms and repair functions in section 3, and a review of relevant work in section 4. Section 5 gives a brief background to molecular genetics. The proposed algorithm is presented in section 6 and the experimental set-up is described in section 7. Results and analysis may be found in section 8 followed by a discussion and brief overview of future work prospects in section 9.

2 The Multiple Knapsack Problem

The MKS problem, sometimes called the multi-constrained knapsack problem, is a generalisation of the single 0/1 knapsack problem. It is NP-complete and may be found in several real-life scenarios such as cryptography or the industrial cutting stock problem. The problem is to fill a series of knapsacks, each of capacity c_i, with any combination of items drawn from a fixed set of size n such that the sum of values of all items in the knapsacks is maximised without exceeding any of the knapsack's capacities. Each item is included in all knapsacks simultaneously with the item's weight depending on each of the knapsacks. More formally,

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

where $x_i \in \{0, 1\}$. The value of each item is denoted as v_i and w_{ij} refers to the weight of item i in regard to knapsack j. In this paper, we make use of the SAC'94 library of benchmark problems which is available online[1]. This library is a collection of solved MKS instances taken from different publications. These instances range in size from 15-105 items and 2-30 knapsacks.

A general review of numerous different optimisation techniques shows that approaches using problem specific knowledge are likely to outperform their "blind" counterparts (see [5]). While almost all algorithms use some understanding about a problem's structure in their design, only few use instance specific attributes. As section 4 will highlight, approaches 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. In the MKS, for example, it is not clear whether to use the average value-weight ratio of an item across all knapsacks or its minimum value. The algorithm described here does not rely on such instance specific knowledge. Instead, it relies upon a general assumption made about the class of MKS problems: Optimal solutions have, on average, minimal waste of capacity and contain items with higher value-weight ratios with increasing frequency. This intuitive concept is further illustrated in figure 1 and, as section 4 will show, has been implemented successfully elsewhere. The next section will investigate different techniques to handle constraints without the use of any problem specific knowledge.

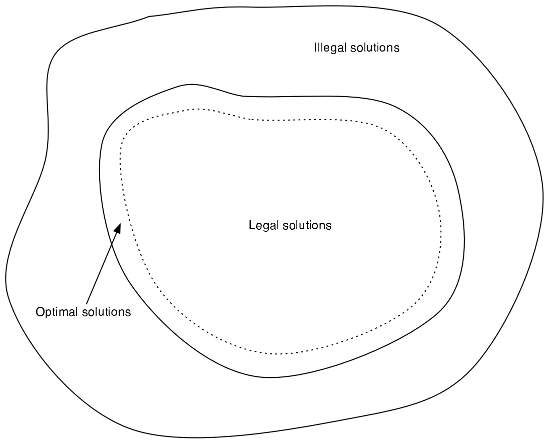

Figure 1: Illustrating the assumptions made about the MKS: A simplistic visualisation of the search space as viewed from a feasibility point of view. Once an solution violates any of the imposed constraints, it becomes illegal. Optimal solutions are likely to be situated very close to the boundary separating the feasible and unfeasible space where waste of capacity is minimised. Optimality is subsequently determined by the value-weight ratios of all items included in the knapsacks.

3 Genetic Algorithms: Penalty Terms versus Repair Functions

GAs ([8], [15]) are abstract implementations of evolutionary systems designed to be used as optimisation techniques: A population of individuals encodes potential solutions to a problem of interest. Selection, recombination and mutation allow for the population's fitness to increase over time until a satisfactory solution has been found or a stopping criterion has been met. GAs have traditionally been using binary encodings and are thus highly suitable for binary optimisation problems such as the MKS: Each individual is a binary vector with each digit indicating the inclusion (1) or exclusion (0) of the indexed item.

We have conducted a test to compare the performance of a GA using penalty terms to an identical GA using repair functions to solve instances of the MKS. We test the two most common penalty approaches, the violation product and sum of violations. The former, labelled pen1, subtracts from the encoding's score the

1. `http://elib.zib.de/pub/Packages/mp-testdata/ip/sac94-suite/`

number of constraint violations multiplied by the highest value of all items. The latter, pen2, subtracts the actual sum of all constraint violations. This is compared to two random repair techniques: rep1 removes items from the knapsacks at random (i.e. converts 1's to 0's) until feasibility is obtained. rep2 adds a second phase to rep1 by adding items at random, if possible, after the removal phase. A standard steady-state GA is used, with parameter settings as described in section 7. We found that a random selection of parents performs well for pen1, pen2 and rep1, while rep2 performs best if the hamming distance between parents is maximised. This may be explained by the loss of diversity using rep2: A repair function restricts the search process to the feasible region of the search space. The second phase of rep2 further restricts the search to solutions close to the boundary of the feasible space (see figure 1). This results in a general loss of diversity which is restored to some degree by maximising the hamming distance between parents prior crossover.

Penalty terms are employed more frequently in conjunction with GAs than repair functions (see [17]). The reason for this is likely to be the possible interference of repair procedure and crossover, both of which perturb the encoding they are acting on. However, the results from this experiment, summarised in table 1, show repair functions to outperform penalty terms on almost all instances, everything else kept equal. A similar result for a different problem was obtained recently by Gréwal et al. [6]: The authors found a repair function to outperform a penalty approach when used in conjunction with a GA on a set of digital signal processor (DSP) benchmark problems. These results encourage the use of repair based techniques for the MKS.

4 Previous Work

The MKS problem is probably one of the most widely studied constrained optimisation problems in the evolutionary computation community. Here we concentrate on approaches that deal with instances contained within the SAC'94 library of benchmark problems. We review four different approaches, two of which use domain specific knowledge. The earliest approach is by Khuri et al. [12] who suggest the use of a graded penalty term (violation product) to penalise invalid solutions based upon the number of constraint violations. The authors use a canonical GA with modified fitness function to focus the search process on the feasible regions of the search space. This concept is taken further by Kimbrough et al. [13] using a two-market GA. Two phases are used to obtain feasible solutions: The first phase is optimality improvement while the second phase is feasibility improvement. The authors test two different penalty terms, violation product and sum of violations, the former of which is identical to the graded penalty term suggested in [12].

Further improvement is achieved using instance specific knowledge: Cotta et al. [2] suggest a hybrid GA with a greedy construction heuristic that builds solutions by selectively choosing items of high value-weight ratios. The individuals in the algorithm's population represent perturbations of the problem instance by manipulation of the items' profits. Jun et al. [10] suggest an evolutionary game algorithm that also makes use of the value-weight ratios. This information is 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 includes items, if possible, in order of decreasing ratios. The superiority of this last approach over the previous ones demonstrates the validity of the assumptions made earlier: The repair function produces tightly packed solutions maximising the value-weight ratios of the items considered for inclusion. The results of all these studies are summarised in table 2 and are used for comparison to the results obtained here (see section 8).

5 Exons, Introns and RNA Editing

This section will provide a brief background in genetics highlighting the motivation for our work. Evolutionary computation has traditionally been inspired by population genetics but there has been a recent trend towards molecular genetics. This movement is partly driven by recent advances in genetics, most notably the genome sequencing projects, which have shed light on the great complexity of information processing in the cell. The approach suggested here has been inspired by some aspects of molecular genetics and hints at the great potential abstractions of biochemical processes have to offer.

The information processing architecture of the cell is highly complex and recent advances in genetics have shown how biochemical processes that occur after transcription but before translation add a significant level of complexity to the expression of DNA. The central dogma of

Instance	opt	pen1		pen2		rep1		rep2	
hp2	3186	36%	3172.2	0%	5645	0%	3119.5	100%	3186
pb6	776	79%	773.7	75%	773	100%	776	100%	776
pb7	1035	63%	1032.6	54%	1031.6	63%	1033.2	100%	1035
pet3	4015	100%	4015	0%	4230	100%	4015	100%	4015
pet4	6120	100%	6120	0%	6577	100%	6120	100%	6120
pet5	12400	99%	12399.9	0%	13298	89%	12398.8	100%	12400
pet6	10618	9%	10587.8	0%	13390	64%	10611.4	31%	10607.5
pet7	16537	4%	16484.5	0%	20594	5%	16476.8	84%	16533.1
sent01	7772	72%	7768.1	65%	7767.4	77%	7769.3	92%	7771.1
sent02	8722	20%	8710.7	37%	8714.9	32%	8715	25%	8713
weing7	1095445	4%	1095158.9	0%	1121698.2	2%	1095194.5	0%	1095033.1
weing8	624319	0%	908647.8	0%	1116698.2	55%	623850.3	2%	620611.7
weish12	6339	0%	6307.5	2%	6790.78	100%	6339	100%	6339
weish17	8633	5%	8575.5	2%	8747.7	98%	8632.8	100%	8633
weish21	9074	4%	9250.4	3%	9578.4	99%	9073.8	100%	9074
weish22	8947	2%	9396.7	2%	9507.6	96%	8946.3	69%	8941.4
weish25	9939	0%	9916.9	3%	10225.3	83%	9937	97%	9938.8
weish29	9410	2%	10417.5	0%	10252	97%	9409.4	95%	9408.9

Table 1: Comparing penalty terms to repair functions: For each approach, the percentage of times the global optimum has been found is shown alongside the final average value after 20,000 function evaluations. Repair functions perform better than penalty terms on almost all instances. It is also interesting to note that several of the final results obtained using the penalty approaches are situated clearly in the unfeasible regions of the search space.

			Approach							
			SGA		HGA		EGA		2-MGA	
Instance	#	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 2: Summary of results from the literature: Each instance is listed alongside its size (number of items/number of knapsacks) and optimal solution. For each technique, all available results are shown: The percentage of times the global optimum has been found alongside the final average value. The number of function evaluations used are shown below each of the names. SGA 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 two market GA. Details in text.

genetics tells us that double-stranded DNA is first transcribed to single stranded RNA which is subsequently translated into a polypeptide of amino acids. However, almost all eukaryotic genes contain non-coding regions called introns, which require splicing prior translation.

In general, eukaryotic genes are modular, composed of exons and introns in an alternating fashion. Exons are often synonymous with independent protein domains allowing for vital effects such as exon shuffling [3] to occur: Exons from different genes may be recombined to produce novel protein products (see [14]). Introns, on the other hand, are artefacts of exon mobility and also serve as a buffer for crossover to occur. Some introns, however, play a much more vital role and may occasionally be included in the final transcript by means of alternative splicing. In other cases, introns may serve as editing templates for adjacent exons in a process called RNA editing. Loosely speaking, RNA editing selectively targets individual nucleotides for modification to yield a fully functional protein fabrication of which would otherwise be prevented by processes such as non-sense mediated decay.

RNA editing itself is catalysed by double-stranded RNA adenosine deaminase (dsRAD) which converts adenosine (A) to inosine (I). Inosine is subsequently interpreted as guanosine (G) making it effectively an A-G exchange. This allows for 26 of 61 codons to be altered, a proportion that encodes 12 of the 20 amino acids [7]. The full biochemical details of RNA editing are highly complex and beyond the scope of this paper (see, for example, [1]) but fortunately not required to formulate a sound and useful abstraction of this post-transcriptional process. The algorithm alongside its repair function is discussed in the next section.

6 The Exonic Genetic Algorithm

The review of past literature highlights the benefit of using an item's value-weight ratio. The approach suggested here makes use of the same repair function as in [10], but instead defines the relative order of items considered for either removal or inclusion and not the actual order. In other words, if we view the encoding as a directional vector, we always start considering the left-most items for removal first. The inclusion phase, on the other hand, proceeds from right to left. Items are represented as movable elements allowing for their actual order to evolve during the evolutionary process. In order to enhance scalability, we relax the requirement for an exact ordering of items: The encoding is composed of segments with items being allowed to move from one segment to another. The order of segments defines the order of consideration for the two phases of the repair function. All items within the same segment, however, are chosen at random. This also increases the population's diversity as identical unfeasible individuals may differ after the repair phase. This principle is illustrated in figure 2. Each item is represented as

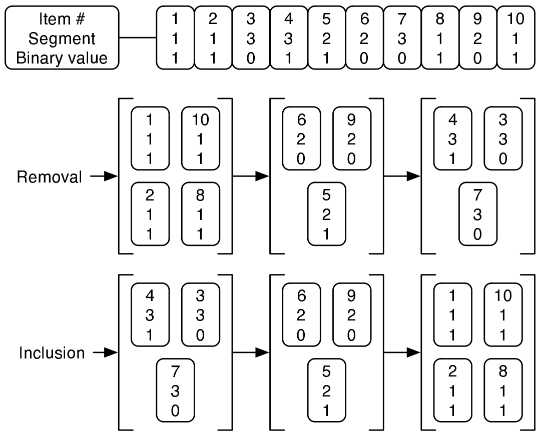

Figure 2: The exonic encoding and repair approach: Each encoding consists of triplets containing the item to be represented, its segment number and binary value. This information is then used to repair the encoding if necessary. Groups are prioritised but items in each group are chosen at random for removal/inclusion.

a triplet composed of the item's number it represents, the current segment number and a binary value indicating whether the item is to be used in the solution (1) or not (0). Two mutation operators are used: One moves items from one segment to another while the other inverts the element's binary value.

The proposed algorithm is an extension to the canonical steady-state GA inspired by the modular organisation of genes. We have shown in previous work how a modular encoding based upon the structural properties of eukaryotic genes may greatly improve the canonical GA on a dynamic version of the knapsack problem [16]. The repair function suggested here bears noticeable resemblance with the process of RNA editing where individual nucleotides within an exon are targeted for modification to produce viable protein products. Such modifications are required, for example, in mitochrondrial RNA to remove stop codons which would otherwise prevent fabrication of active proteins [7]. Mitochrondrial RNA may thus be considered unfeasible prior to its repair similar to the unfeasible encodings of the MKS. GAs with RNA editing have been suggested previously [9] but those implementations bear only little resemblance to the approach suggested here. Apart from the newly proposed encoding and repair function, the GA uses standard components throughout: Each iteration consists of the common selection, crossover, mutation and replacement cycle. The first parent is selected at random while the second parent is the individual in the population with maximum hamming distance to the first. This is to enhance diversity during the algorithm's execution. Uniform crossover is used followed by the mutation operators described earlier. In order to use uniform crossover we have to fixate the order of items represented by each individual. Mutations to segment numbers are used to establish movability. In other words, the first element of an individual always represents the first item of the instance to be solved, but its segment number allows for a different ordering to be establish during the two-phase repair process. For replacement, each of the two offspring competes with one of its parents, chosen at random, in a binary tournament.

7 Experimental Setup

The experiments have been set-up to comply with the settings found in previous work. We use 18 instances of the MKS chosen from the SAC'94-library. For each instance, we run the algorithm 100 times with the number of function evaluations limited to 20,000. The crossover rate is set to 0.8 and a low mutation rate of $0.5/n$ is chosen, where n is the number of items in the problem instance. The mutation rate is identical for both mutation operators. The population size is 150 and the number of exons is set to 10. These parameters have been established after some initial, although by no means exhaustive, testing. For each instance, we note the number of times the optimal solution has been found and how many function evaluation were required on average to do so. We also note the total average result found across all 100 runs after 20,000 function evaluations. The results are shown and discussed in the next section.

8 Results and Analysis

All results are shown in table 3 and compared to those discussed in section 4. We compare performances solely based upon the information available in the literature and do not implement the other approaches to generate further data. The ExGA completely outperforms SGA and 2-MGA being at least as good across all instances. It is interesting to note the relatively poor performance for instance weing7, the only case where the ExGA performs worse than HGA. In all other cases, however, ExGA performs better than HGA. Comparing ExGA to EGA, the best of the other approaches, we note a slightly superior performance: The two approaches perform equally well across the majority of instances while the ExGA performs better on 3 instances and worse on 2.

The number of function evaluations required to reach the optimum solution is generally much lower than the limit of 20,000 and, in the case of pet3, as low as 706. Most striking is the poor performance for weing7. On the other hand, performance on sent02 and weing8 is superior to EGA by 23% and 17% respectively. The results also indicate the method seems to scale well and that performance is dependent upon the specific attributes of the problem space rather than the number of items. For example, weish29 with $n = 90$ is solved every time while pet6 with $n = 39$ is solved in only 82% of all cases. Further extensive testing is required as the number of instances considered here is insufficient to validate this assumption. Figure 3 shows the algorithm's performance in respect to the number of items in each problem instance indicating no correlation between performance and problem size.

9 Conclusion and Future Work

We have shown that a "blind" search algorithm may produce results competitive with those obtained using algorithms with explicit domain specific knowledge. The exonic genetic algorithm (ExGA) incorporates a 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 same priority being selected at random during the repair process. This improves the algorithm's scaling performance, at least across the limited number of test cases considered here. The adaptive approach offers at least two advantages over the other approaches in the literature: Firstly, problem 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 remove such item if the solution is unfeasible. The ExGA, however, may place such item in a position such that it will never be considered for removal.

As mentioned earlier, further testing is required to firmly establish how well the algorithm will scale. It is also of interest to investigate why there is a performance drop on some instances, most notably weing7. Addi-

	ExGA			Differences				
Instance	Solved	FunEvals	Average	SGA	HGA	EGA	2-MGA	Rep2
hp2	100%	5386.4	3186	-	-	±0%	±0	±0%
pb6	100%	2515.7	776	-	-	±0%	+45.8	±0%
pb7	100%	6288.2	1035	-	-	+0.4	+2	±0%
pet3	100%	705.6	4015	+17%	±0%	±0%	-	±0%
pet4	100%	1450.9	6120	+67%	+6%	±0%	-	±0%
pet5	100%	2193.4	12400	+67%	±0%	±0%	-	±0%
pet6	82%	8888.8	10615.6	+78%	+22%	−17%	-	+51%
pet7	100%	9918.2	16537	+99%	+54%	±0%	+50.4	+16%
sent01	100%	9471	7772	+95%	+25%	±0%	+2.2	+8%
sent02	92%	15305	8721.9	+90%	+53%	+23%	+1.5	+67%
weing7	15%	16257.9	1095317.4	+15%	−25%	−85%	+590.4	+15%
weing8	90%	13456.1	624178.2	+84%	+61%	+17%	+550.4	+88%
weish12	100%	5919.3	6339	-	-	±0%	±0	±0%
weish17	100%	5661.1	8633	-	-	±0%	±0	±0%
weish21	100%	8809.2	9074	-	-	±0%	±0	±0%
weish22	100%	10776.9	8947	-	-	±0%	±0	+31%
weish25	100%	11957.2	9939	-	-	±0%	±0	+3%
weish29	100%	11743.1	9410	-	-	±0%	+206.8	+5%
Instances worse				0	1	2	0	0
Instances equal				0	2	13	6	9
Instances better				9	6	3	8	9

Table 3: For each instance, the number of times the optimum has been found is shown and how many functions evaluations on average were required to do so. The total average after 20,000 function evaluations is shown also. Where possible, the number of times the optimum has been found is compared. In all other cases, the numerical difference after 20,000 function evaluations has been used.

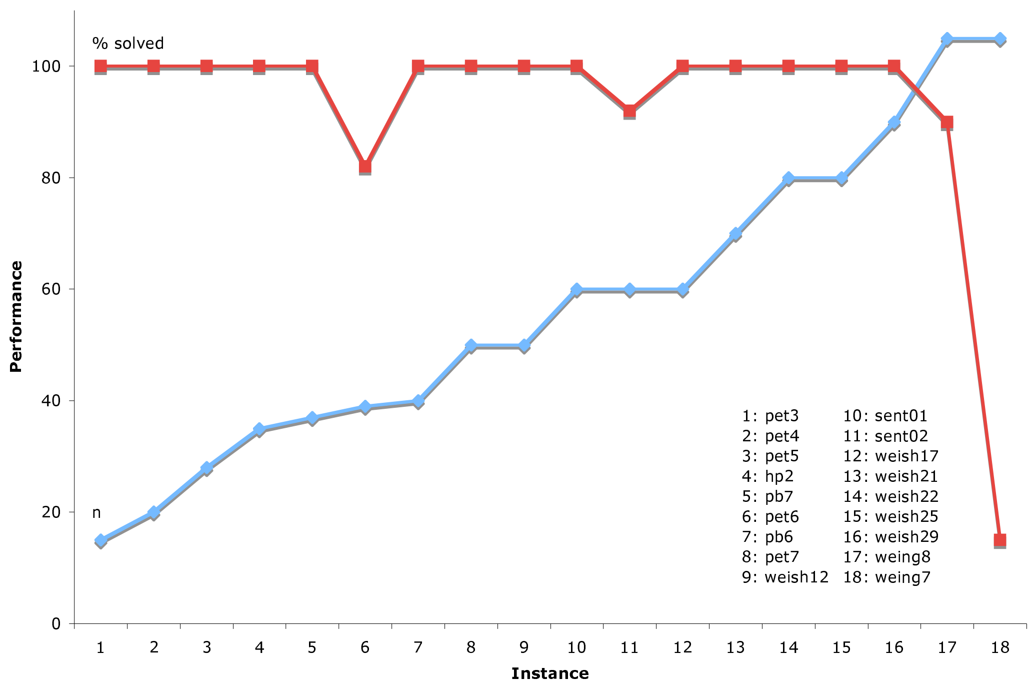

Figure 3: Performance of ExGA across all instances: The top line shows the number of times the instance has been solved (out of 100 runs) while the lower line shows the number of items in each instance.

tional future work should also attempt to determine whether the suggested approach could be applied to other constrained problems that so far have relied upon the explicit use of instance specific information. The multiple knapsack problem is of great interest and we have shown here how it may be solved efficiently making some assumption about the class of problem in general but refraining from the use of instance specific information. The success of the suggested approach also supports the notion that algorithms exploiting insights from molecular genetics are worthy competitors to the well established traditional approaches that are based on population genetics.

Acknowledgements

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

References

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

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

[4] J. Gottlieb. On the feasibility problem of penalty-based evolutionary algorithms for knapsack problems. In E. J. W. Boers, J. Gottlieb, P. L. Lanzi, R. E. Smith, S. Cagnoni, E. Hart, G. R. Raidl, and H. Tijink, editors, *Applications of Evolutionary Computing: EvoWorkshops 2001: EvoCOP, EvoFlight, EvoIASP, EvoLearn, and EvoSTIM*, Como, Italy, 2001.

[5] J. J. Grefenstette. Incorporating problem specific knowledge into genetic algorithms. In L. Davis, editor, *Genetic algorithms and simulated annealing*, pages 42–60. Morgan Kaufmann Publishers, 1987.

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

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

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

[9] C.-F. Huang and L. M. Rocha. Exploration of RNA editing and design of robust genetic algorithms. In *Proceedings of the 2003 IEEE Congress on Evolutionary Computation.* IEEE Press, 2003.

[10] 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.

[11] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems.* Springer, 2004.

[12] 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.

[13] 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.

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

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

[16] P. Rohlfshagen and J. A. Bullinaria. Alternative splicing in evolutionary computation: Adaptation in dynamic environments. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computing (CEC 2006)*, Piscataway, NJ, 2006. IEEE.

[17] O. Yeniay. Penalty function methods for constrained optimization with genetic algorithms. *Mathematical and Computational Applications*, 10(1):45–56, 2005.