Alternative Splicing in Evolutionary Computation: Adaptation in Dynamic Environments

Philipp Rohlfshagen and John A. Bullinaria

***Abstract*— Natural organisms have to deal ef£ciently with changing environments and are thus a great source of inspiration to solve dynamic problems in arti£cial domains. Dynamic optimisation has gained a lot of interest lately as many real world problems are indeed dynamic. In this paper, we look at post-transcriptional processes and alternative splicing in particular: Although these biochemical processes are gaining increasing attention from the genetics community, they remain relatively unexplored in evolutionary computation. We suggest a simple abstract encoding that allows one to construct multiple expressions from the same template supporting quick adaptation to changes in the cost surface. This encoding enables the system to £nd, control and reuse groups of building blocks that are being shared by different environments. Tests on a modi£ed version of the dynamic knapsack problem show that it signi£cantly outperforms the canonical genetic algorithm as well as simple implementations of random immigrants and hypermutations.**

I. Introduction

Evolutionary computation (EC) has been successfully applied to many different classes of optimisation problems where conventional methods tend to fail. There has been a trend recently to extend the £eld of EC to cover problems that are of a dynamic nature. Dynamic optimisation deals with cost surfaces that change over time such that the algorithm is required to constantly track the moving optima. Dynamic optimisation has gained a lot of momentum recently as many real-life problems are indeed dynamic: Resources may have to be reallocated due to technical failures, traf£c congestion may alter the cost per distance, or new jobs are added to a schedule due to unforeseen events. In order to deal with the dynamics of a problem, numerous techniques have been developed to improve upon the canonical genetic algorithm (GA). Interestingly, natural systems are inherently dynamic and thus provide a valuable source of inspiration and it comes as no surprise that many novel approaches have indeed been inspired by nature (e.g. diploidy).

The approach presented here has been motivated by a phenomena in genetics known as alternative splicing (AS). AS is a biochemical process that allows the exploitation of alternative pathways in eucaryotic genes to fabricate a multitude of different proteins from a single template. It is a post-transcriptional process that controls the expression of RNA under the in¤uence of certain regulatory features such as cell type: Each cell contains the same sequence of DNA yet the cell type may greatly affect the splicing pattern such that different proteins are produced in different circumstances. This is very much related to dynamic optimisation where different conditions require quick adaptation to track the shift in the search space. Post-transcriptional processes have not yet received much attention in the EC community, although some work has been carried out (see, for example, [16] for an implementation of RNA editing). Here we present an abstract implementation of AS to be tested on dynamic optimisation problems. We will use the term 'AS' throughput this paper to refer to both the biochemical process and the suggested encoding.

The paper is structured as followed: Section II gives a brief introduction to dynamic optimisation and reviews previous work in this area. Closely related work is reviewed in section III followed by an explanation of AS in natural systems in section IV. The encoding is presented in section V followed by the description of the benchmark problem. The experimental setup is outlined in section VII followed by a discussion and analysis of the results which are concluded in section IX.

II. Dynamic Optimisation

EC has predominantly been applied to classes of static optimisation problems. Many problems are, however, dynamic in the real world: Noise, uncertainty and other factors may often contribute to a cost surface that changes over time. A problem of great complexity is either impossible or very costly to solve and in a changing environment, one should therefore attempt to utilise the best solution found so far to guide adaption to the shifted optimum. This should in general be more ef£cient than a complete restart of the algorithm. Problems of interest are therefore those whose states bear a degree of similarity before and after the change. Otherwise, no technique can outperform a random restart approach [2]. Classical implementations often fail to track changes suf£ciently quickly due to convergence and subsequent loss of diversity. To combat this shortcoming, many different approaches have been suggested in the past. These approaches either enhance diversity, use memory of previous states or multiple populations to track different areas of the search space concurrently. A brief overview of different techniques may be found in [3]. The remainder of this section outlines the main ideas.

Hypermutations [5] drastically increase the mutation rate in response to environmental changes in order to randomise the population. Random immigrants [11], on the other hand, merge randomly generated individuals with existing ones to

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

Philipp Rohlfshagen is with the Department of Computer Science, University of Birmingham, UK (email: P.Rohlfshagen@cs.bham.ac.uk).

John A. Bullinaria is with the Department of Computer Science, University of Birmingham, UK (email: J.A.Bullinaria@cs.bham.ac.uk).

prevent convergence at any one time. While this may help to increase diversity, it also disrupts the current search process and may subsequently prevent the algorithm from locating the global optimum. Recently, more emphasis has been on algorithms that use memory of previous states, either implicitly or explicitly. In the former case, the memory is embedded in the encoding, usually in the form of diploidy (e.g. [10]) or polyploidy (e.g. [12]). This has been used successfully for small numbers of distinct states. The structured GA [7] differs in its approach to implicit memory and is discussed in the next section. The interest in implicit memory has decreased lately as implementations only dealt with a very limited number of states, each of which is usually represented in the genome in its entirety. This makes implicit memory unsuitable for large problem instances.

Explicit memory approaches keep a register of individuals and release them back into the population depending on their current £tness. The memory is of £xed size and different memory replacement schemes have been suggested to maintain a diverse set of individuals. Explicit memory schemes work well if states encountered are highly related to the points stored in memory, but may also misguide the current search process [2]. A hybrid approach is suggested by Yang [21] where individuals from memory are hypermutated before they are introduced into the population. Nevertheless, choosing a suitable size and replacement scheme for the memory may be dif£cult. Yet another class of techniques uses multiple populations to keep track of optima encountered in the past. Often, a central population keeps track of the overall search while multiple smaller populations diverge to track promising regions in the search space [4].

III. Related Work

We know of no other work that explicitly addresses the use of AS in arti£cial evolution. This may well be due to the fact that AS has only recently gained attention due to a better understanding. Nevertheless, there has been some work that follows similar principles, yet under different names and descriptions. Levenick's swappers [18], for example, are individuals that may switch on different parts of their genome given environmental stress. Although swappers are extremely simple encodings and use only 2 different segments, they somewhat capture the essence of AS. Diploid encodings are also related as they allow alternative use of implicitly stored information. This expression is controlled by a dominance scheme and a wealth of different approaches has shown this choice to be dif£cult. In addition, it is impossible to divide the genome into autonomous subsections without the use of linkage information and thus the entire genome is expressed alternatively in its entirety. Diploid approaches therefore fail to work in dynamic environments exceeding two distinct states [3].

Another related and slightly more sophisticated approach is the structured GA [7] which uses an embedded control region that switches groups of genes on and off: The number of active control genes is £xed, as is the position of the controlled segments. Mutations to the control sequence occur by swapping one bit for another. This encoding has been applied to a succession of two and three states, each of which has to be represented in the genome in its entirety, making this approach unsuitable for larger problem instances. Furthermore, only a single control gene may be active at any one time: Whenever a new environment is encountered, the control sequence is expected to express the part of the genome that implicitly stores that particular state. If multiple mutations would be required to £nd the correct control sequence, it is likely that an accumulation of neutral mutations in the implicit memory would alter the information content to such a degree that memory is lost.

Neutral mutations are a fundamental aspect of implicit memory schemes. They maintain diversity and encourage exploration of novel parts of the search space, yet also degrade memory of previous states. Any segment not currently being expressed in the phenotype is subject to such degradation. Reusing information, however, implies that larger proportions of the genome are active at any one time. If, for example, two different states are 95% similar and the encoding is able to capture this similarity, only ≈10% of the entire genome is subject to neutral mutations (95% shared, 5% for each state). For this to occur, the algorithm has to identify building blocks common to all states or subsets of states. This also reduces the size of the encoding considerably as the implicit memory is expressed through linkage rather than redundancy. This forms the basis of the suggested encoding which is discussed after a brief overview of AS in natural systems.

IV. Alternative Splicing in Nature

The carrier of information in almost all organisms is DNA which is being passed on from generation to generation. Information is interpreted in the context of the cell: Certain regions of double-stranded DNA are transcribed to single-stranded RNA and subsequently translated into a polypeptide chain of amino acids (a protein). These regions are generally known as genes. For long it was assumed that a single gene produces a single protein, but recent advances in genetics have shown that this is not the case. The latest data predicts 20,000-25,000 genes to be found in the human genome which is signi£cantly lower than previous estimates of 150,000 [6]. These results stress the importance of the biochemical processes that act upon RNA to produce multiple proteins from a single gene. These post-transcriptional processes are not yet very well understood, but recent studies reveal their great importance in the generation of complexity. AS is one of the processes that allows the selective recombination of different regions of RNA to produce large numbers of different proteins from the same sequence of nucleotides. Typically, DNA is transcribed to RNA which subsequently undergoes processing. RNA processing removes intervening non-coding regions (introns) and brings together all coding regions (exons) to produce a continuous strand of protein coding RNA. For the majority of eucaryotic organisms, all introns are spliced and all exons retained. Yet this is not always the case and special regulatory elements may cause introns to be retained and exons to be skipped or extended

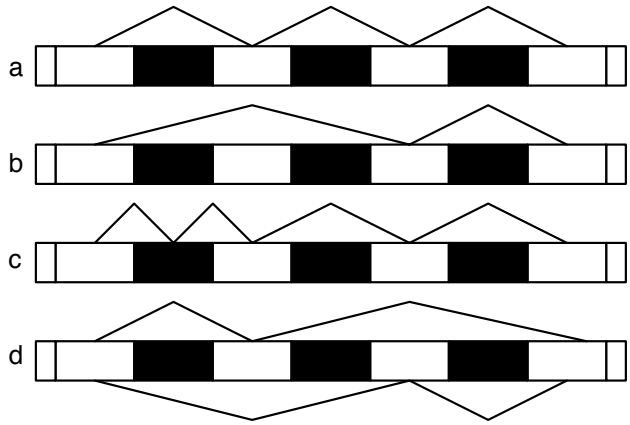


Fig. 1. Examples of splicing patterns in eucaryotic genes (white=exons, black=introns). (a) shows the normal splicing pathway exlcuding all introns and retaining all exons. In (b), the second exon is skipped while in (c), the £rst intron is retained. Finally, (d) shows 2 mutually exclusive exons. Alternative 3' and 5' splice sites are not shown.

by using different splice sites. The proteins produced are often related, yet there is no restriction and proteins may vary greatly. Probably the best example is the splicing pattern in *Drosophila melanogaster* where AS determines the gender: The female splice variant includes exon 4 while the male ones does not [1]. Herbert and Rich [15] describe eucaryotic genes as 'soft-wired' as the expression of a gene depends on its post-transcriptional processing. This is in contrast to procaryotic or 'hard-wired' genes which always produce identical protein products. Some common splice patterns are shown in £gure 1.

The exon shuf¤ing theory [8] assumes exons to be subdomains of proteins that may be used in different contexts. This effectively allows AS to recombine exons and introns in different con£gurations to produce fully functional protein products. A well known example is the *Dscam* gene in *Drosophila* which may produce up to 38,000 different transcripts from a single template [19]. Comparing human DNA to that of other animals stresses the importance of post-transcriptional processes: It is now well established that an organism's complexity is not de£ned by its number of genes but by the size of its proteome (number of different proteins in an organism). The high frequency of alternative splicing in the human genome, currently estimated at 40-60% (see [14]), con£rms this reasoning. The regulation of alternative splicing is not yet well understood but a recent summary of results [17] shows small regulatory units contained within introns and exons controlling the splicing pathway of nearby splice sites. Luckily, not all biochemical details are required to formulate a sound and useful abstraction. The important attribute of AS may crudely be summarised as the ability to reuse information in different con£gurations depending on the environment. It is this view that inspired the encoding described in the following section.

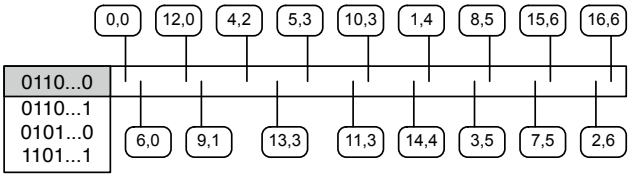


Fig. 2. Each individual consists of two parts, a memory of splicing patterns and an array of nucleotides. Only one splicing pattern is active at any one time and controls which segments of the gene are transcribed. Nucleotides (value, segment) are free to move along the gene and may thus either be included or excluded from the phenotype depending on their corresponding segment and the active splicing pattern.

V. An Abstract Implementation of Alternative Splicing

The encoding scheme we are proposing combines elements of implicit and explicit memory to represent multiple states concisely within a single non-redundant gene. The encoding consists of two parts, a memory of splicing patterns and a virtual gene which is divided into n segments. Nucleotides are modelled as pairs of value and segment number. Classically, GAs model evolution from a population dynamics point of view and not a biochemical one. Here, however, each individual represents a single gene (as opposed to a genome) and is thus composed of nucleotides (as opposed to genes). This distinction is important from a biological point of view but much less meaningful from a computational perspective. The splicing patterns are binary and indicate which segments contribute towards the phenotype (the bit's position references the segment while its value determines the segment's state). There are m splicing patterns, each of length n, only one of which is active at any one time. Further, inactive splicing patterns are shielded from mutation (explicit memory). Nucleotides are able to move between segments by an inversion operator that simply places a nucleotide from one segment into a randomly chosen one. Mutation to the active splicing pattern occurs by ¤ipping the bit. There is no restriction on how many segments are active at any one time. Whenever a change to the environment occurs, all splicing patterns are evaluated and the currently best one (as judged by the £tness of the resulting phenotype) is made active. The encoding bears a strong similarity to the messy GA [9] and linkage learning algorithms [13]. However, there is no under- or over-speci£cation and no crossover. The latter is partly substituted by the use of splicing patterns which have a similar effect as they combine different segments of the gene. Implementation is straightforward: The problem may be represented as a vector of integers representing the segment number of the indexed object. Further, a binary vector containing the splicing patterns is required. An example of the encoding is depicted in £gure 2.

In its simplest form, the encoding is precisely equivalent to the standard binary encoding: A single state requires two segments and a single splice regulator which silences one of the segments. The mutation operator thus moves items from

TABLE I

A DYNAMIC VERSION OF THE KNAPSACK PROBLEM: ALTERING 2 VALUES (ITEMS 3 AND 15), 4 STATES ARE BEING CREATED THAT ALL MOVE THE GLOBAL OPTIMUM SOME DISTANCE AWAY (THE CAPACITY IS HELD CONSTANT AT $c = 60$). IN ORDER TO REACH THE OPTIMUM, CERTAIN ITEMS NEED TO BE REMOVED FROM THE KNAPSACK FIRST TO MAKE SPACE FOR THE MODIFIED ITEM (I.E. THE ORDER OF CHANGES IS IMPORTANT). ALL SUCCESSIVE OPTIMAL SOLUTIONS DIFFER BY 4 BITS. WE WILL REFER TO THE DIFFERENT STATES BY THEIR NUMBERS AS SHOWN IN THE TABLE.

																		optimum
	item	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
	weights	12	5	20	1	5	3	10	6	8	7	4	12	3	3	20	1	2	
values	1	2	3	**9**	2	4	4	2	7	8	10	3	6	5	5	7	8	6	
	2	2	3	9	2	4	4	2	7	8	10	3	6	5	5	**25**	8	6	
	3	2	3	**21**	2	4	4	2	7	8	10	3	6	5	5	25	8	6	
	4	2	3	21	2	4	4	2	7	8	10	3	6	5	5	**7**	8	6	
states	1	0	**1**	**0**	1	1	1	0	1	1	1	**1**	**1**	1	1	0	1	1	71
	2	0	**0**	0	1	1	1	0	1	1	1	**0**	**0**	1	1	**1**	1	1	84
	3	0	0	**1**	1	**0**	1	0	**0**	**0**	1	0	0	1	1	1	1	1	86
	4	0	0	1	1	**1**	1	0	**1**	**1**	1	0	0	1	1	**0**	1	1	80

the active to the inactive segment and vice versa. This is equivalent to inverting a single bit. The mutation operator here, however, allows for items to be linked together by assignment of identical segment numbers. The regulatory sequence in turn allows for tightly linked groups of nucleotides to be combined. This allows the encoding to tightly link elements found in certain subsets of states in a dynamic optimisation problem. This implicit memory may then be constantly reused. In order to successfully encode 2 states, one would need 2 regulatory sequences[1] and 4 segments: 1 segment for items contained in none of the states, 1 segment for items contained in both states and 2 segments for items speci£c to either state. It is simple to encode the knapsack problem as each nucleotide points to a particular item. Binary problems may be encoded by translating all nucleotides (the value of which indicates its position in the binary array) in active segments as ones and all others as zeros.

This approach effectively allows one to view dynamic problems as static. There is one globally optimal solution to solve the problem independent of its current state. Once that solution is found, no further evolution is required to take place unless noise and uncertainty is introduced into the system. Each individual is initialised completely at random. The number of segments theoretically required is

$$1 + \sum_{k=1}^{n} \frac{n!}{k!(n-k)}.$$

This is the upper bound that accounts for all possible combinations of states. In reality, however, the number of segments required is likely to be much lower, as the problem in this paper illustrates: There is, for example, no genetic material exclusively shared between states 1 and 2. Instead of the theoretical maximum of $1 + \sum_{k=1}^{4} \frac{4!}{k!(4-k)} = 19$ combinations, only six actually occur. Rather than using the theoretical maximum, it is often more useful to approximate the number of segments required. Such approximations also allow us to use the encoding even when the number of different states is unknown a priori.

[1]The results show that fewer regulatory sequences may be needed than there are states.

VI. THE DYNAMIC KNAPSACK PROBLEM

The dynamic knapsack problem, originally suggested by Goldberg [10], has been used frequently to test novel approaches to dynamic optimisation. The knapsack problem is NP-hard and is found in several real-life scenarios such as cryptography or the industrial cutting stock problem. The static knapsack problem is stated as follows: Fill a knapsack of capacity c with any combination of n items such that the total value of all items in the knapsack is maximised without exceeding the knapsack's capacity. More formally,

$$max \sum_{i=1}^{n} v_i x_i \text{ subject to } \sum_{i=1}^{n} w_i x_i \leq c \text{ where } x_i \in \{0, 1\}.$$

This problem is made dynamic by altering the knapsack's capacity over time. The capacities are usually drawn at £xed intervals from a small predetermined set. This approach has been criticised by Branke [2] because the change from higher to lower capacities renders all top-quality solutions invalid. A penalty function penalises any individual that encodes an invalid solution (i.e. a solution that exceeds the knapsack's capacity) by reducing that individual's £tness, and the choice of penalty function thereby greatly affects the overall performance of the algorithm. Any algorithm that requires more generations to recover at this particular stage (i.e. making illegal encodings legal) is disproportionally penalised due to the usually huge penalty term. This blurs the algorithm's performance in subsequent generations even if adaptation is comparatively quicker. Furthermore, we found that knapsack problems with alternating weight constraints are fairly simple to solve using the classical genetic algorithm. The knapsack problem is, however, still a good candidate for dynamic optimisation. Here we suggest a new approach of making the knapsack problem dynamic

TABLE II

THE OPTIMAL SOLUTION FOR THE PROPOSED PROBLEM: 4 SPLICING PATTERNS, ONE FOR EACH STATE, CONTROL THE ACTIVITY OF ANY OF 6 SEGMENTS CONTAINING ITEMS BELONGIN TO ANY OF THE IDENTIFIED SUBSETS OF STATES.

Splicing Patterns				Group	Items	State
0	0	0	0	0	0 6	none
1	1	1	1	1	3 5 9 12 13 15 16	all
1	0	0	0	2	1 10 11	1
1	1	0	1	3	4 7 8	1 2 4
0	1	1	0	4	14	2 3
0	0	1	1	5	2	3 4

overcoming the previously mentioned concerns. Rather than changing the knapsack's capacity, we selectively change an item's value. We therefore change the value/weight ratio and possibly introducing a new global optimum without affecting the validity of currently optimal solutions. Increasing the value of an item not currently in the knapsack forces the encoding to free some space £rst. This attribute is of a deceptive nature as it forces the algorithm to move away from the previous global optimum by decreasing its £tness. A local hillclimber should not be able to track the change. Small and well-chosen alterations to the capacity may further be utilised to introduce some noise into the system. We constructed a test-problem with 4 states (labelled state 1 to 4) using the same 17 item knapsack problem previously used in the literature (e.g. [10], [7]) including the same penalty function. We use this problem as we deem it suf£ciently dif£cult to solve, yet easy enough to fully analyse. The problem is shown in table I and the optimal solution to the problem encoded as AS is shown in table II.

VII. EXPERIMENTAL SETUP

Dynamic optimisation problems are usually controlled by two independent variables, the magnitude p and frequency t of change. The magnitude of change for this problem is £xed at 4 bits, although other instances may easily be designed to allow for an arbitrary distance between states. Although 4 bits may seem trivial, the ordering amongst bits makes the problem dif£cult. The frequency of change is usually de£ned by the number of generations allowed for the algorithm to locate the new optimum. We de£ne a generation as ps function evaluations where ps is the size of the population. We tested for intervals of 10, 50 and 100 generations and a total of 100 changes per run. These choices are consistent with other experiments carried out elsewhere in the literature. The AS encoding requires multiple function evaluations per individual after a change and thus the frequency criteria is adjusted to ensure the same number of function evaluations between changes for each encoding (we reduce the number of generations per interval by $m-1$). Each trial is averaged over 30 distinct runs using identical seeds across all experiments. The performance is measured as the percentage of times the global optimum has been found. The mutation rate is set to 1/17 for both encodings. The splicing patterns are mutated with a much lower probability as multiple nucleotides are affected simultaneously. We therefore lower the mutation rate by a factor of 10. The AS encoding uses 4 splicing patterns and 10 segments. The population size is set to 100 and a steady state GA with binary tournament replacement has been used (see [20]). In addition, a two-point crossover with probability 1 has been used for the classical binary approach (SGA).

In order to test the different attributes of the encoding, the algorithm is subjected to different scenarios. The simplest case is the continuous succession of the 4 states. A slight variation thereof randomly chooses the next state with replacement (i.e. the frequency of states changes also). A third experiment tests each encoding on the £rst two states and introduces the other two states after 50 changes. Random immigrants (RI) and hypermutations (HM) have been tested on the sequential succession of states also and are included in the comparison. We set the rate of hypermutation to an average of 4 mutations per individual. The random immigrants are introduced only after a change occurred which we found much more ef£cient than introducing a small number of random immigrants after each generation. The replacement is random and set to 30%.

VIII. RESULTS AND ANALYSIS

The percentages of times the global optimum has been found for the sequential succession of all 4 states are shown in table III and are also depicted in £gure 3. The graphs show the algorithm's behaviour over time with each of the 4 states indicated by a horizontal dotted line. A t-test has been used to determine statistical signi£cance when comparing AS to any of the other approaches ($+$ and $-$ have been used to indicate signi£cantly better and worse performance respectively at a signi£cance level of 0.005). AS outperforms the three other approaches except in one case where RI is better than AS for intervals of 10/7 generations. As with the SGA (where performance is identical for 10/7 generation intervals), this is due to the 'learning' phase of AS as can be seen in £gure 3(a). Comparing only the second half of the run shows AS to signi£cantly outperform both the SGA and RI. It can be seen in the graph that the SGA fails to locate state 3 at all. Furthermore, although states 2 and 4 are consistently tracked, state 1 is lost again after several generations. AS also encounters some degree of degradation with state 3, but to a lesser degree. The degradation is, however, related to the number of generations between changes. Although most encoded information is being used all the time, inactive segments may still accumulate some neutral mutations, the effect of which is magni£ed by the length of the interval.

The second trial was executed to determine whether the order of states or their frequency would affect the performance of AS, and whether AS can learn additional states after convergence to a subset of states. The results are shown in table IV and £gure 4. AS again outperforms the SGA in all cases as its performance is consistent across all experiments. AS is also capable of learning states 3 and 4 after convergence to states 1 and 2. Interestingly, the SGA

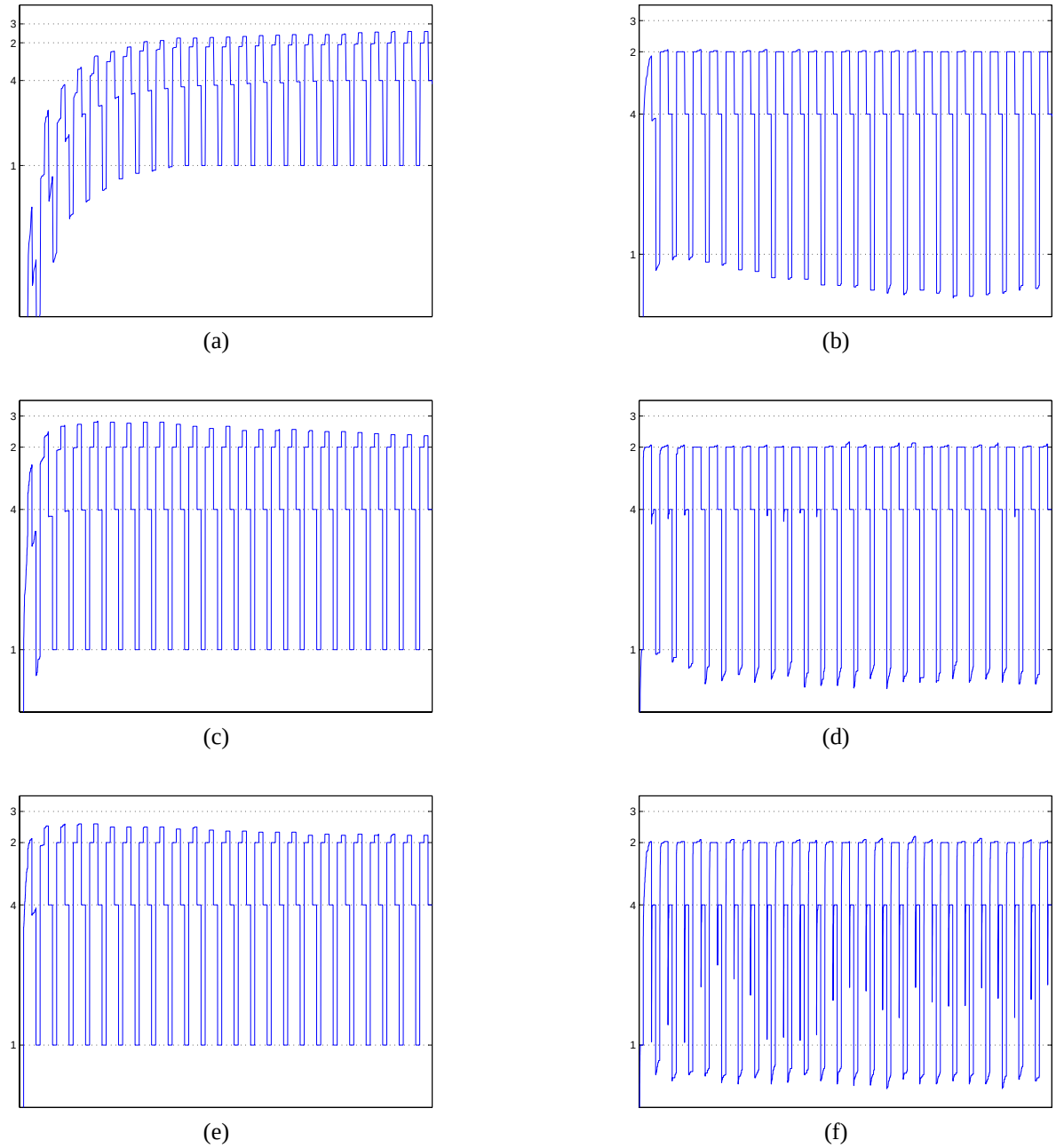
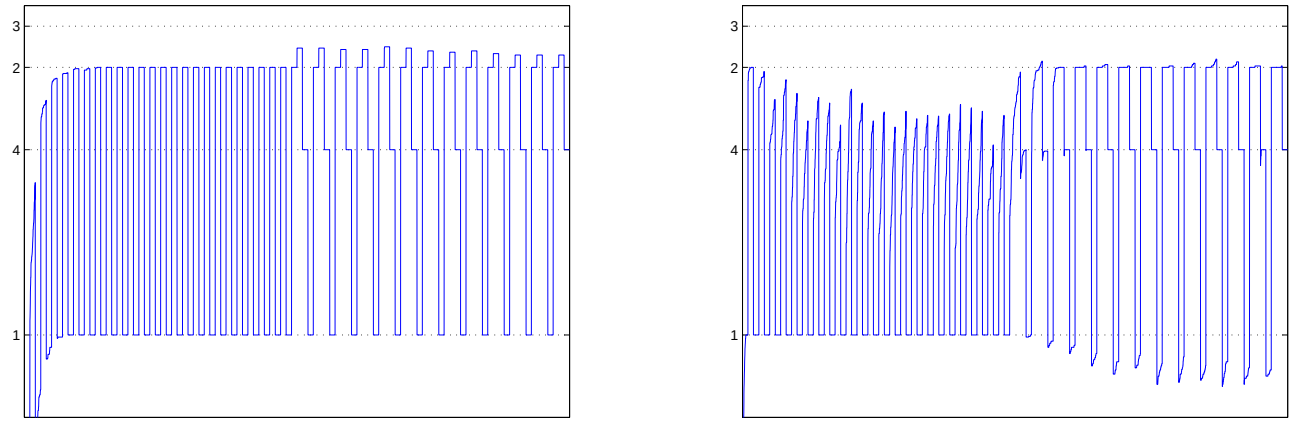


Fig. 3. Showing the algorithm's behaviour over time: The left side (a,c,e) shows AS for 7, 47 and 97 generation intervals respectively. The right hand side (b, d, f) shows SGA for 10, 50 and 100 generation intervals. AS exhibits a 'learning' phase for the short succession of states as it discovers common factors to all states. Some degradation is noticeable for state 3, especially for the longer intervals. SGA on the other hand fails to discover state 3 at all and degrades heavily on state 1 once the other states have been found.

seems to struggle with just two states as its performance is worse than over the entire set of states. This may be due to a lack of diversity. It may also hint at the dif£culty of the problem, however: With just two states, there is exactly one path to reach the next state (from state 1 to 2, remove items $\{2, 11, 12\}$ then add item $\{15\}$). With three states, however, there are multiple paths from state 1 to state 3 if state 2 is not found. Also interesting to note is the degradation of state 1 once states 3 and 4 are introduced which may be explained by the same phenomena. The SGA fails to discover state 3 at all. AS also shows some degradation as before with state 3. The two state experiment also shows that the choice of parameters may be ¤exible: 4 splicing patterns and 10 segments have originally been chosen for 4 states, yet the algorithm performs equally well on 2 states using the same settings. Choosing the right settings may turn out to be dif£cult for large problem instances or simply impractical depending on the size of the problem ($m \times n$



(a) (b)

Fig. 4. Behaviour of AS (a) and SGA (b) over an entire run where states 3 and 4 are introduced only after half the run. Intervals in this example are 47/50 generations long. AS is able to locate both new states although degradation with state 3 is again evident. SGA on the other hand struggles with two states and fails to locate state 3 at all while degrading heavily on state 1 once states 2 and 4 are found.

TABLE III

COMPARING THE PERFORMANCE OF ALTERNATIVE SPLICING (AS) TO THE CANONICAL BINARY ENCODING (SGA) AND SGA IN COMBINATION WITH RANDOM IMMIGRANTS (RI) OR HYPERMUTATIONS (HM) FOR DIFFERENT INTERVAL SETTINGS t.

t	AS	SGA	t-test	RI	t-test	HM	t-test
10/7	55	59	*	65	– *	6	+
50/47	85	63	+	68	+	60	+
100/97	82	58	+	63	+	74	+

* AS is + for second half of run

TABLE IV

COMPARING THE PERFORMANCE OF AS TO SGA FOR A RANDOM SUCCESSION OF STATES (RANDOM) AND FOR THE CASE WHERE STATES 3 AND 4 ARE INTRODUCED AFTER 50 CHANGES OF STATES 1 AND 2 ONLY (2+2 STATES).

Experiment	t	AS	SGA	t-test
random	10/7	54	59	*
	50/47	85	60	+
	100/97	86	62	+
2+2 states	10/7	59	67	*
	50/47	87	66	+
	100/97	86	70	+

* AS is + for second half of run

bits for the regulatory elements are required alongside the integer vector). The second experiment did show, however, that AS still performs well if the settings are not so well chosen. Looking at the use of splicing patterns over time (see £gure 5) shows that the algorithm predominantly uses only two splicing patterns (1 and 3) in the £nal third of the run.

In this work we explicitly use knowledge about changes in the environment to trigger the evaluation of the explicit memory. If such knowledge is not available, the constant evaluation of a signi£cant number of splicing patterns would be very costly unless true parallel processing is available. This may pose a problem for some domains. A compromise would be to trigger the evaluation of the explicit memory at £xed, and possibly adaptive, intervals. If the intervals are chosen suitably well, even slow continuous change may be tracked ef£ciently. In some domains it is also possible to trigger the evaluation of the explicit memory once the population's £tness degrades in response to environmental change. Preliminary experiments using a single splicing pattern showed that worthwhile improvement may still be achieved, but these results require further study. A more detailed analysis on the effect of the number of splice sites and segments will be carried out in the near future.

IX. CONCLUSION AND FUTURE WORK

Our abstracted formulation of alternative splicing (AS) applied to evolutionary computation has been shown explicitly to be able to £nd sub-solutions common to subsets of problem states and to reuse that information when required. The encoding therefore provides a very compact representation of entire dynamic landscapes. The dynamics are effectively removed from the problem as there is now a single static optimal solution. Our experimental results showed AS to perform signi£cantly better than the standard binary encoding, hypermutations or random immigrants. This has been demonstrated for a variety of different experiments across which AS exhibited a steady and robust performance throughout. Although the reuse of information may imply a fragility towards mutation (as each mutation affects all states

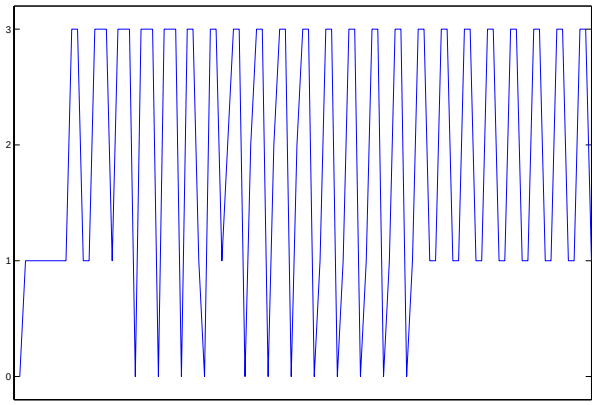


Fig. 5. The frequency distribution of active splicing patterns for a random sample of the £rst experiment (7 generation cycle). All four splices are used (with increasing regularity after the 'learning' phase) showing rhythmic activity for about half the run. The remainder of the run only uses splicing patterns 1 and 3 hinting at the possibility to reduce the number of splicing patterns below the number of states.

the encoding refers to), it has been shown that this is not the case: Some degradation was noticed but to a lesser degree than the binary encoding. In order to prevent loss of memory, a repair mechanism is suggested that allows the genome to repair itself using a statistically generated template. This is analogous with RNA editing and experiments will be carried out in the near future to test this idea.

Interestingly, the results seem to imply that an explicit memory scheme may not work for this particular kind of problem: As the canonical binary encoding is unable to reach the global optimum in several of the cases, a memory may only recover suboptimal solutions. The ability of AS to locate optima in the £rst place hints at its applicability for static optimisation problems. We are currently investigating the performance of AS on a set of multiple knapsack benchmark problems: The alternative splice forms may generate diversity using well-established optimal sub-solutions to produce better solutions with high probability.

Our initial results are promising yet, much work remains to be done. A question of particular interest is the scaling capability of the encoding. Also of interest is the choice of parameters and their effect on the overall performance. Nevertheless, it is shown that once again, natural systems have found a mechanism to deal effectively with the environments they are exposed to.

REFERENCES

[1] B. S. Baker. Sex in ¤ies: The splice of life. *Nature*, 340:521–524, 1989.

[2] J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. In *Congress on Evolutionary Computation CEC99*, volume 3, pages 1875–1882. IEEE, 1999.

[3] J. Branke. Evolutionary approaches to dynamic optimization problems - updated survey. In *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, pages 27–30, 2001.

[4] J. Branke, T. Kauler, C Schmidt, and H Schmeck. A multi-population approach to dynamic optimization problems. In I. C. Parmee, editor, *Adaptive Computing in Design and Manufacture (ACDM 2000)*, pages 299–308. Springer, 2000.

[5] H. G. Cobb. An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependant nonstationary environments. Technical Report AIC-90-001, Naval Research Laboratory, Washington, USA, 1990.

[6] International Human Genome Sequencing Consortium. Finishing the euchromatic sequence of the human genome. *Nature*, 431:931 – 945, 2004.

[7] D. Dasgupta and D. R. McGregor. Nonstationary function optimization using the structured genetic algorithm. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature 2*, pages 145–154, Amsterdam, 1992. Elsevier.

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

[9] D. E. Goldberg, D. E. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis and £rst results. *Complex Systems*, 3:493–530, 1989.

[10] D. E. Goldberg and R. E. Smith. Nonstationary function optimization using genetic algorithms with dominance and diploidy. In J. J. Grefenstette, editor, *2nd International Conference on Genetic Algorithms*, pages 59–68. Lawrence Erlbaum Associates, 1987.

[11] J. J. Grefenstette. Genetic algorithms for changing environments. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature 2*, pages 137–144, Amsterdam, 1992. Elsevier.

[12] B. S. Hadad and C. F. Eick. Supporting polyploidy in genetic algorithms using dominance vectors. In *6th International Conference on Evolutionary Programming*, volume 1213, pages 223–234. Springer, 1997.

[13] G. Harik. *Learning gene linkage to ef£ciently solve problems of bounded dif£culty using genetic algorithms*. PhD thesis, University of Michigan, Ann Arbor, 1997.

[14] E. D. Harrington, S. Boue, J. Valcarcel, J. G. Reich, and P. Bork. Estimating rates of alternative splicing in mammals and invertabrates. *Nature Genetics*, 36(9):915–917, 2004.

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

[16] 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.

[17] A. N. Ladd and T. A. Cooper. Finding signals that regulate alternative splicing in the post-genomic era. *Genome Biology*, 3(11):1–16, 2002.

[18] J. R. Levenick. Swappers; introns promote ¤exibility, diversity and invention. In FL Orlando, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. M. Honavar. Jakiela, and R.E. Smith, editors, *GECCO-99: Proceeding of Genetic and Evolutionary Computation Conference*, volume 1, pages 361–368, San Francisco, CA,, July 1999. Morgan Kaufmann.

[19] D. Schmucker, J. C. Clemens, H. Shu, C. A. Worby, J. Xiao, M. Muda, J. E. Dixon, and S. L. Zipursky. Drosophila dscam is an axon guidance receptor exhibiting extraordinary molecular diversity. *Cell*, 101(6):671–684, 2000.

[20] F. Vavak and T. C. Fogarty. A comparative study of steady state and generational genetic algorithms for use in nonstationary environments. In T. C. Fogarty, editor, *AISB Workshop on Evolutionary Computaing, Lecture Notes in Computer Science*, volume 1143, pages 297–304. Springer, 1996.

[21] S. Yang. Memory-based immigrants for genetic algorithms in dynamic environments. In *Proceedings of the 2005 Genetic and Evolutionary Computation Conference*, volume 2, pages 1115–1122. ACM Press, 2005.