LU TP 16-29
June 2016

LINKING THE DYNAMICS OF GENETIC ALGORITHMS TO THE ENCODING OF INFORMATION

Bachelor's thesis

Henrik Åhl

Department of Astronomy and Theoretical Physics, Lund University

Thesis supervised by Carl Troein and Adriaan Merlevede

LUND UNIVERSITY

Wisely and slow; they stumble that run fast.

— *Friar Lawrence, to Romeo*

Abstract

Genetic algorithms are complex constructs often used as heuristic search methods in contexts ranging from combinatorial optimisation to *in silico* evolution. They draw inspiration from the principles of biological evolution by utilizing the concepts of mutation, reproduction and selection in order to improve a population of solutions. The solutions are often represented as abstract data sequences, e.g. as binary strings, though it has been shown that the choice of encoding alters the performance of the search.

In this thesis, the dynamics of four types of encodings used to evolve a set of integers are investigated: the previously researched bijective *Binary* and *Gray code* maps, as well as an introduced encoding scheme which includes non-coding data, denominated the *consensus* map. Coding parts of the consensus sequences are signified by pre-determined start sequences, after which subsequent code is interpreted via either the Binary or the Gray code scheme.

The performance of the genetic algorithm is measured by the ability to solve four problems with different search spaces. The dynamics are also investigated by identifying the effects the encodings have on the distribution of cost effects due to point mutations, as well as by the ability to produce good solutions under a range of different mutation rates.

It is found that the bijective maps give rise to similar distributions of cost effects, but significantly different performances. The consensus encodings instead allow for higher robustness with respect to different mutation rates, as well as a robustness towards highly negative effects due to point mutations. It is also shown that having an encoding–decoding map which allows for both smaller and larger steps in phenotype-space is an important part of being able to produce good solutions in a range of cases.

Populärvetenskaplig sammanfattning

Genetiska algoritmer utgör en grupp sökalgoritmer som hämtar inspiration från evolutionära processer i biologiska sammanhang. Genom att låta *populationer* av enskilda datasekvenser, exempelvis strängar av ettor och nollor, representera olika lösningar till ett givet problem, och sedan successivt låta datasekvenserna utbyta information sinsemellan, samt att genom mutationer modifiera dem, är det möjligt att producera optimala eller väldigt goda lösningar till problemet i fråga; allt som principiellt behövs är ett sätt att *tolka* en datasekvens och ett sätt att *utvärdera* hur bra eller dålig den i sammanhanget är. Populationer kan också växa, reduceras eller förändras innehållsmässigt över tidssteg – så kallade *generationer* – under sökningen. Vanligtvis ges populationsmedlemmar som utvärderats som bättre lösningar än andra större utrymme att föröka sig eller överleva, så att en partiskhet gentemot högpresterande datasekvenser förekommer, i enlighet med selektionstrycket inom just biologisk evolution.

På så vis skiljer sig genetiska algoritmer från många andra sökalgoritmer, som istället ofta använder sig av direkt matematiska metoder för att hitta bättre lösningar. Detta gör dem i synnerhet lämpliga för problem där det inte existerar någon direkt bakomliggande matematisk konstruktion, eller då det är ovanligt svårt att formulera den. De stora likheterna med biologisk evolution medför också att genetiska algoritmer kan användas som verktyg för att förstå just evolutionära processer, som ofta är oerhört komplexa och icke-intuitiva.

Det har dock visat sig att hur en lösning utläses från en datasekvens – eller motsvarande, hur informationen i sekvensen är inkodad – spelar roll för hur väl algoritmen presterar. Olika typer av informationsinkodningar medför också olika typer av evolutionär dynamik, vilket har konsekvenser för på vilket sätt populationen av lösningar successivt blir bättre.

För att ytterligare förstå på vilket sätt inkodningen är av vikt, och vilken sorts dynamik som medföljer en viss representation av information, undersöks i denna uppsats olika inkodningar och konsekvenserna som dessa får för mutationseffekter på datasekvenserna. Att i högre grad förstå denna komplexa dynamik kan i slutändan förhoppningsvis medföra en djupare förståelse för funktionerna hos genetiska strängar i framför allt datavetenskapliga sammanhang, men också hos deras motparter i naturen.

Contents

1 Background

1.1 The study of genetic algorithms

Genetic algorithms (GAs) make up a family of computational models inspired by the evolutionary principle of natural selection. They are used as population-based heuristic search methods, and provide wide applications for problems related to especially computational and combinatorial optimisation. Because of the similarities to biological evolution, they are also, along with other methods adhering to the principles of *Evolutionary computation*, such as *Evolutionary algorithms, Genetic programming* and *Evolution strategies*, occasionally also used to study the dynamics of evolutionary processes [1]. However, because many of the methods are principally alike, the distinction of what strictly defines GAs is often unclear, but they can in broader terms be viewed as all population based models which utilize *selection*, *recombination* and *mutation* operators to produce sample points in a search space. These three pillars of GAs are often referred to as *genetic operators* [2].

Implementations of GAs typically feature information encoded as a population of strings of bits, although also real-valued and non-binary encodings are common for solving many problems [3, 4]. The encoded information, denoted the *genotype*, serves as an abstraction of a candidate solution, normally referred to as the *phenotype.* Phenotypes are evaluated by a performance measuring function, such as a *fitness*, *cost* or *objective* function, which are all in principle equivalent, but are usually used to signify different measures. For example, the cost function stands as the opposite of fitness, i.e. as fitness increases, the cost decreases. The objective function is similarly simply a name for the measure of the ability to solve some specified objective.

Especially in optimisation settings, the strength of GAs often come from their lack of need of a derivative in order to traverse the search space. Instead, change and successive progression is determined by the genetic operators, which generate new points based on the previous state. Compared to other gradient-less global search methods, such as *simulated annealing*, GAs have been proven to perform better in certain cases of combinatorial optimisation, such as component placement in electric circuits, as shown by Manikas and Cain [5], as well as in telecommunications traffic routing, which Mann and Smith [6] demonstrated. On the other hand, Jennison and Sheehan [7] propose that GAs typically are not well suited for solving computational optimisation problems with large sets of variables of equal significance.

In settings which instead aim to investigate the dynamics of evolutionary systems, GAs, due to their similarities with the process of natural evolution, provide a natural approach for investigating various features of evolutionary behaviour. The systems investigated are not necessarily required to be biological, as they have been used in order to understand, among many other things, the evolution of grammar, communication, age preference between sexes, and evolvability itself [8–12]. Hu and Yang [1] additionally utilize GAs as tools to understand sequence evolution under constraints by solving the game of Sudoku. The underlying structure and theory of the genotype–phenotype map, which is often applied to understand GAs, has also been studied in other contexts, as by Stadler et al. [13] when questioning the current view of the link between the genotype and the phenotype space. In relation to this, it has been shown that how one encodes information, i.e. the phenotype, can be of importance for how well the algorithms perform. As GAs traverse a search space of possible solutions with varying fitness, i.e. a *fitness landscape*, the encoding of information is a determining factor for the shape of the landscape. Consequently, the encoding affects the ability of the GA to traverse different states and thus also the capability of finding the global optimum [14].

However, even though GAs have been a subject of interest for many years, and featured a wide range of applications within various fields of science and engineering, their dynamics have not been thoroughly investigated. In this thesis it is therefore analysed how the encoding of information alters the ability of GAs to solve problems of different complexities. It has previously been proposed that the encoding should preferably be bijective in the sense that the genotype–phenotype map should have a one-to-one link [2, 15]; a proposition here investigated further by the introduction of encodings including "dead code", which has no direct effect on the phenotype, but instead allows for increased phenotypic accessibility and alternative evolutionary pathways. The effects which neutral encodings have on the search process have previously been demonstrated by Shipman et al. [16], but the fitness effects due to operations on encodings providing multiple-to-one genotype–phenotype maps remain to be investigated, and could possibly shed further light on the dynamical aspects of these types of encodings.

The analysis is performed by investigating the distribution of fitness effects due to point mutations on the genotype, as well as by evaluating performances with respect to the ability to solve specific problems with varying complexity. The encodings are also examined by their ability to produce good solutions for a range of mutation rates.

1.2 General structure

1.2.1 From code to fitness

How a GA is structured varies greatly on a case by case basis. This is largely due to the *No Free Lunch Theorem*, which states that no general purpose optimisation strategy can be optimal for all problems [17]. Consequently, a wide variety of GAs exist, where the specific implementations depend on what exact tasks the algorithms are applied to. Regardless, a general structure can still be established.

As stated, GAs are population-based in the sense that a set of candidate solutions are maintained during execution. The genotypes, which are also known as *chromosomes* or *genomes*, are typically encoded as a string of bits or real-valued numbers, but occasionally also as other types of data structures [18, p. 60–82]. The genotype sequence can be of either fixed or variable length, although fixed length genomes are far more prevalent [19, 20].

Genomes normally code for homologous types of information in subsections referred to as *genes*. For example, in a parameter optimisation setting, the individual genes can be chosen to code for separate parameters in an equation. Variations of GAs with heterogeneously encoded parameters, i.e. parameters of different, non-compatible types, within a single chromosome do, however, also exist [21].

Each individual genotype in the population has to be interpreted via an encoding–decoding map, or equivalently, genotype–phenotype map, in order for a solution to be evaluated. In imitation of how phenotypes in nature fare differently due to environmental factors, the phenotype is evaluated by a fitness, cost, or objective function, which measures how good a solution ultimately is. An example of the link between the genotype–phenotype–fitness stages is illustrated in Figure 1. Because every genotype gives rise to a phenotype, which is then given a fitness, the link between the genotype and fitness of a binary genome of fixed length can be seen as a mapping $\{0,1\}^N \to \mathbb{R}$, where N is the number of bases, i.e. sequence elements, in the genome.

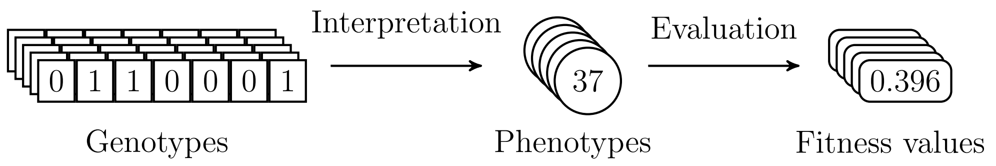

Figure 1: Mapping from genotype to fitness. The genotype codes for a phenotype which is then evaluated by a given fitness function. The fitness of a population is often determined by the best phenotype, although various choices can be made.

1.2.2 Genetic operators

GAs rely on three types of operators for successively improving a population of solutions: selection, mutation and crossover. These operators come in many variations, with the specific implementation depending on the problem at hand, as well as on the other settings of the GA. A conceptual interpretation of the flow of operations in GAs can be seen in Figure 2, although many implementations also feature a selection process for which individuals should be chosen for mutation and crossover operations.

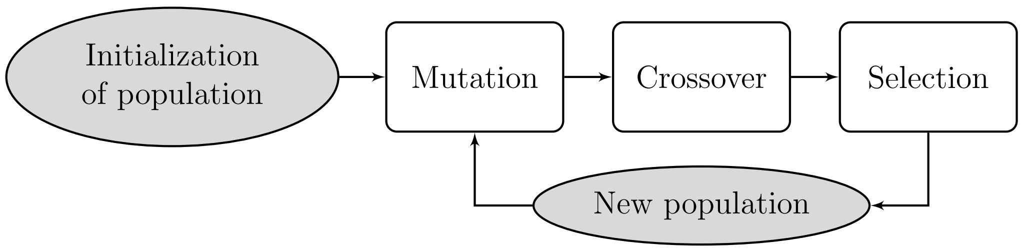

Figure 2: Process flow scheme of a typical Genetic Algorithm. If no terminating condition is invoked, the number of iterations in the process is indeterminate. Every pass in the loop increments the current *generation*.

Selection The selection operation can be applied in several parts of the process: in picking out individuals for reproduction, picking individuals to mutate, and picking individuals for death or survival. It is one of the most determining features for how the population will evolve as it defines the advantages an individual will have due to a higher fitness than another individual in the population. It also determines the degree of phenotypic diversity within the population.

Multiple choices of selection operators exist, although a few are more frequently applied than others. Common choices of methods are *elitism*, which unrestrictedly favours the strongest individuals; *roulette wheel selection*, which picks individuals with probabilities proportional to their fitness; and *tournament selection*, which picks out a random subset of the population, and then chooses a number of individuals from a geometric distribution, with the individuals sorted by fitness [22]. It should be noted that whenever elitism is used, it is often coupled together with other selection methods in order to maintain the genotypic and phenotypic diversity within the population.

Mutation Mutation operators alter the genome such that an alternative solution is produced. Often this is done while keeping the individual used as basis for the mutation unaffected. The process can thereby be likened to a type of asexual reproduction. Because of this, the terms *parent* and *offspring* are commonly used whenever new individuals are created from other.

Typical choices of mutation operators are operators performing *point mutations*, which alter the value of singular bases in the genome; *insertions*, which randomly produce or copy blocks of genotypic information that are inserted into the genome; and *deletions*, which delete blocks of the genome.

Implementations of point mutations on binary genomes typically flip every bit in the genome with a probability p, as illustrated in Figure 3.

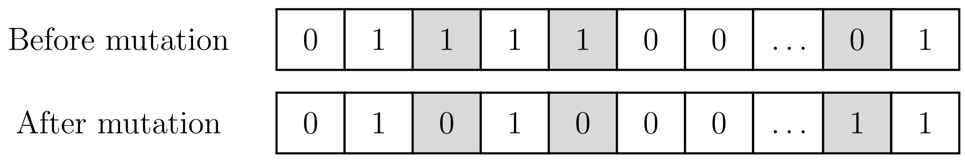

Figure 3: Example of a point mutation operation on a binary genome. Bits at every index are flipped with a given probability $p \in (0, 1]$. Gray boxes show changes in the genome due to the operation.

The mutation operator can be seen as the producer of genotypic diversity within the population, and is the major component for sampling new points in the search space, which allows the algorithm to find better solutions [22].

Crossover The crossover operation is the analogue of sexual reproduction in nature. Inspired by the recombination of biological chromosomes during meiosis, the process is an attempt to recombine features from different genomes.

Also for crossover operators, multiple variations exist, while the most common for binary genomes are single-point and dual-point crossovers, which cut the genome at one or two points respectively. The resulting parts are then recombined into new individuals [22]. An illustration of a one-point crossover is provided in Figure 4.

It should be noted that a crossover operator is not strictly necessary for evolving GAs as mutation operators are in principle sufficient for sampling new points in the search space.

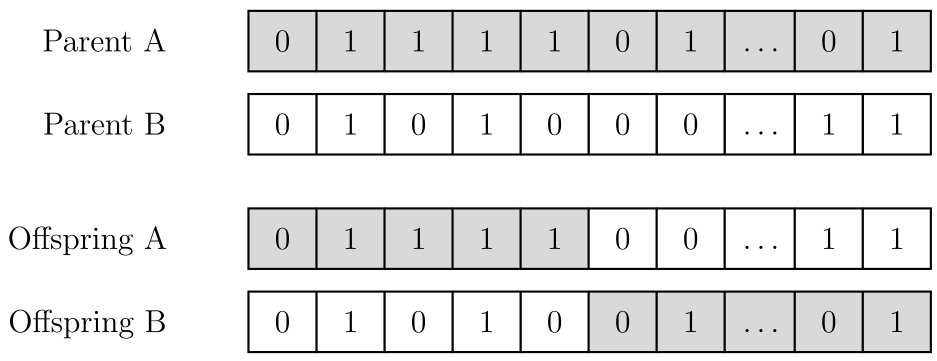

Figure 4: Illustration of a one-point crossover on two parental genomes. A random or pre-chosen cutting point determines which sections of the genomes should be assigned to the corresponding offspring.

1.3 The search space

GAs perform a search by selectively traversing an N-dimensional search space, where N is the number of bases in the genome. Every configuration of the genome defines a point in the search space. The population based mechanism then allows for performing implicit, parallelised local search [2, 23]. Because the fitness is what ultimately determines whether a point in this search space is good or not, the fitness function is a defining factor of the shape of the search space. However, also the encoding plays a significant role.

1.3.1 The encoding of information shapes the search space

It has been shown that how the search is performed depends largely on the encoding of information, as the encoding determines the available phenotypes of a given state, and thereby affects the connection between points in the fitness landscape. This implies that the encoding is a significant component in how well the algorithm in turn performs on a particular problem [14]. Because of this, the effects of the choice of encoding are in a general sense inherently non-trivial to clarify. However, in the specific case of encoding integers as binary strings, interpreting the effects on the dynamics of the GAs' temporal progression becomes markedly simpler. Traditionally, integers are normally encoded via either a *Binary* (henceforth capitalized when referencing the encoding), or a *Gray code* scheme.

Binary The Binary encoding–decoding map is a straightforward and intuitive way of interpreting and encoding an integer from and to a sequence with a binary base. In the encoding scheme, every base value represents a power of two, as is common in particular for computer-like devices. However, the Binary scheme allows for the existence of so-called *Hamming cliffs* – occurrences when two adjacent points in integer space are well separated in the genotype space. In GAs, this can significantly hamper the progression of the algorithm. Consider for example the complementary binary strings `1000` and `0111` which in the Binary scheme code for the integers 8 and 7 respectively. Although these are adjacent in the ordered integer space, they require four bit flips in order to assume each other's phenotypes. This limitation, however, can be circumvented by the use of the Gray code scheme.

Gray code Gray code, after its inventor Frank Gray (1887–1969), is an alternative way of encoding integers such that Hamming cliffs cannot occur. It does so by encoding integers in such a way that two adjacent states always lie precisely one base change away from each other, i.e. at a distance of one in genotype space. This simple requirement allows for the existence of multiple types of Gray code, which then have to be translated and interpreted via the Binary scheme in order to render an integer. This process makes the use of Gray code less tractable and somewhat computationally cumbersome, but even so, it has been shown that Gray code offers a significant increase in performance in certain cases [14]. Like the Binary scheme, the Gray code map provides a bijective genotype–phenotype map, but Gray coded sequences are not guaranteed to undergo a translation in integer-space corresponding to a power of two when a single bit is changed.

1.3.2 Neutral evolution

Literature on GAs often mentions the importance of using an encoding–decoding map which is bijective; usually with the argument that such a choice decreases the size of the search space and lessens the number of states required to be investigated [2, 15].

Even so, several authors have argued that multiple genotypes coding for the same phenotype allows for increased phenotypic accessibility, and therefore also alternative evolutionary pathways [13, 16, 24, 25]. *Neutral evolution* is the inheritance of traits which do not have any effect on the fitness of an organism, both in biological and *in silico* settings. Contrary to the notion of preferably having a bijective encoding–decoding map [15], the argument for neutral evolution being a benign feature of these maps has recurrently been made, as it allows phenotypes to be more accessible via *silent mutations*, i.e. mutations on the genome which do not alter the phenotype. The case for neutral evolution is then that it allows for increased evolutionary flexibility, as well as – in biological settings – robustness towards external effects which might be harmful to the organism [13, 24–26].

In biological genetic strands, significant portions of the genome are known to not have any direct functional importance. The parts which do are instead often marked by some type of signifying sequence, to which enzymes and similar molecules bind. The sequences are often approximate, so that complexes can bind in spite of local variations; the typical, or the average such sequence, is called a *consensus sequence* [27]. In other words, these types of sequences are biological genetic constructs which in principle signify the importance of adjacent genetic material, and can thereby in abstraction be seen as to separate significant from non-significant code within the genome.

2 Methods

2.1 Algorithm setting and genotype–phenotype maps

A population of 40 strings of bits was used in the implementation of the GA. In every generation, all individuals were replicated and mutated by point mutation. For the mutation, the probability for each base be inverted was set to unity divided by the genome length, e.g. $p = 0.01$ for a genome of length 100. To limit the scope of investigation on the dynamical effects, no crossover operator was applied. For survival, a tournament selection scheme was used, picking two individuals at random and selecting the most fit out of those. This was done without repicking of a previously chosen individual, and reiterated until 40 individuals had been picked for the new population.

Four different encoding–decoding maps were used in order to interpret bitstrings as integers: Binary (B), Gray code (G), Consensus Binary (CB) and Consensus Gray (CG). In the Binary and Gray code settings, bitstrings of length 100 were spliced into 10 separate genes, with every individual gene coding for an integer $I \in [0, 1023]$. In both settings using consensus-like encodings, the genome and gene lengths were set to 100 times the length of the non-consensus variants, i.e. 10000 and 1000 bits respectively. The sequence `110011` was used to signify the start of a coding sequence within the gene, where then the 10 following bits encoded an integer, interpreted either via the Binary or the Gray code scheme. If multiple starting sequences were found within a gene, the corresponding values were averaged, rounding to the nearest integer.

The process of finding and decoding significant code in the consensus encodings was set up so that no two coding sequences were allowed to overlap. Genes not containing any start sequences were penalized by an added 10000 in cost in order to severely bias towards strings with consensus sequences in every gene.

2.2 Algorithm objective

The chosen objective for the GA was set to find a randomized set of 10 target integers, with elements $T_i \in [0, 1023]$. The cost for each separate individual was subsequently evaluated as its sum of smallest pairwise distances, as depicted in Figure 5. In the implementation, the pairing was performed from lowest to highest value over every gene in the bitstring; an integer targeted by two genes at the same distance would thereby be paired with the one of lowest value.

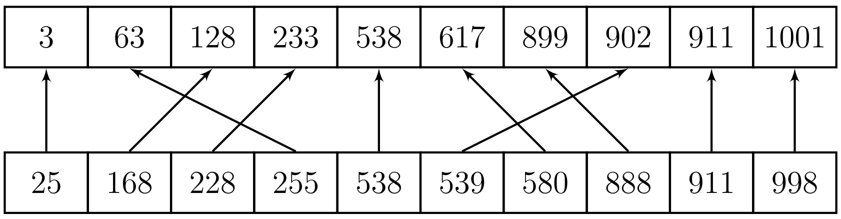

Figure 5: Illustration of the problem setting used, with the individual bitstring's phenotypic genes (lower array) sorted by value. The upper array shows the target integers, likewise sorted by value. The bitstring is evaluated by the sum of smallest pairwise distances, with every gene linking to the integer in the target set closest in integer-space, as long as no other gene in the individual with a smaller distance to that specific target integer is present. The distance between the strings on the whole is evaluated as the sum of the distances. This particular example thus describes a total distance of 673.

In order to modify the complexity of the problem, i.e. to transform the search space, the measure of the total distance was passed through four different functions of different styles, as described in Table 1 and visualized in Figure 6. The cost function used to evaluate the individuals was chosen as the transformed distance.

Table 1: Transformation functions for changing the search landscape. All functions are defined $\forall x : x \geq 0$, with the boundary condition that the function returns 0 at $x = 0$.

Function name	**Expression**
Linear	x
Stairs	$x - x \pmod{11} + \frac{11}{2}$
Moguls	$x + 11 \sin \frac{x\pi}{11}$
Sharktooth	$x + 22 \left\lvert \sin \frac{x\pi}{11} \right\rvert$

The `Stairs` modifier complicates the landscape by not allowing individuals to discern how far they are from the solution within the boundaries of the plateaus. The `Moguls` modifier, named after the bumpy trails in moguls skiing, instead complicates the search by alternatingly letting individuals perceive the distance as closer respectively further from the solution than it actually is. This implies that the hill-climbing abilities of the algorithm are more important than in the `Linear` case. The `Sharktooth` modifier, again named because of the shape, is another take on the `Moguls` complexity, where the cost affiliated with evaluation is always greater than, or equal to, the non-modified distance.

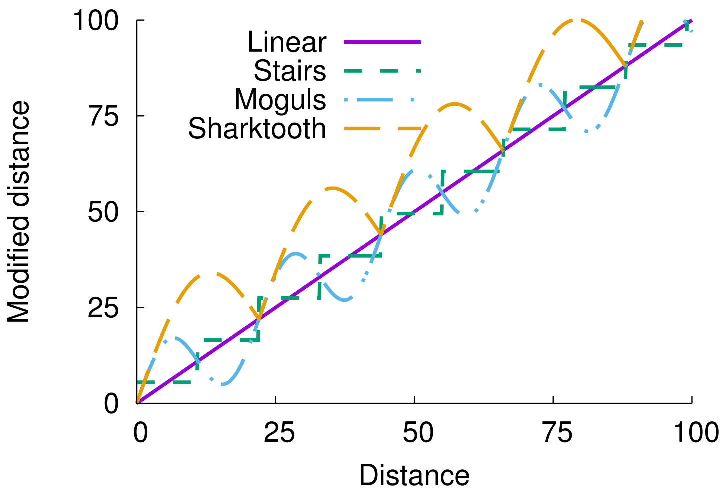

Figure 6: Functions used to transform the search space of the GA. The mathematical expressions used can be found in Table 1.

3 Results

3.1 Analysing performance

The Gray code scheme proved to in all of Figure 7A-D be superior in solving the designated objective. However, the average performance varies significantly with the distance transformation functions, while the corresponding effect for the other encodings is not as noticeable. The Binary scheme is with respect to the different transformation functions the most stable, as it consistently ends up at roughly the same cost level in every complexity case. However, the only encoding whose performance noticably trends towards a slope of zero is the Gray code scheme, although the consensus sequences show tendencies to do so in the `Stairs` transformation as well.

Considering the error, it is also noted that the consensus encodings are affiliated with larger fluctuations than the bijective encodings in all cases. The consensus encodings are furthermore initially at a higher cost level than the bijective encodings. More precisely, the Binary and Gray code schemes start with a cost of slightly less than 500, while the consensus encodings begin just under 2500.

The consensus sequences required a factor of 10 more computation time to reach generation 100000, the consensus sequences took roughly an order of magnitude more time to finish on average. The Gray code performed slightly worse than the Binary in the same analysis.

For the consensus encodings, it is also noted that the cost level of the best individual in the population is improved by adding or removing start sequences in 90.6 ± 1.3 % of the cases for the Consensus Binary, as well as 90.0 ± 1.5 % for the Consensus Gray encoding.

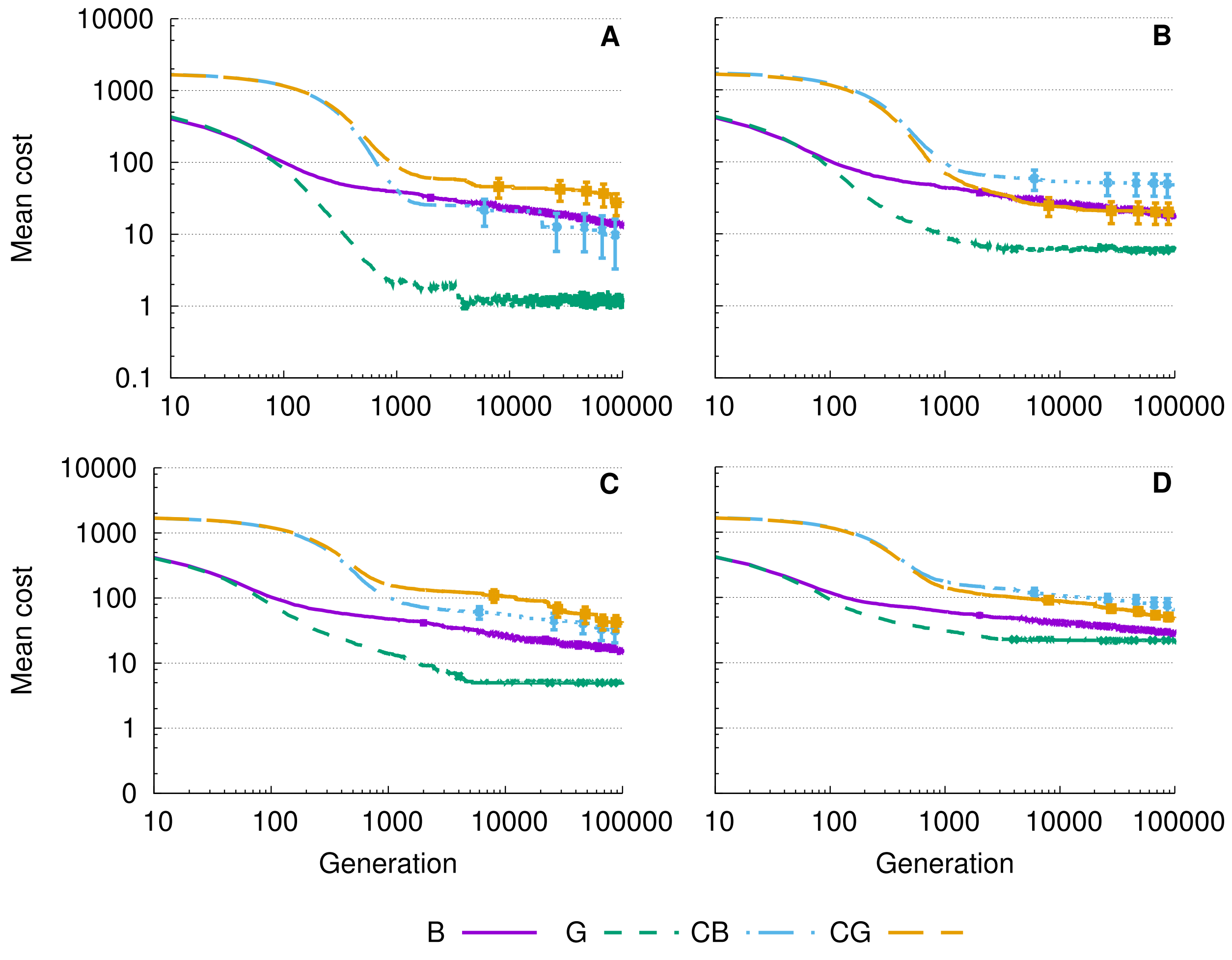

Figure 7: **A: Linear B: Stairs C: Moguls D: Sharktooth**
Evolution of the mean cost trajectories for the four different transformation functions, sampled over 100 separate simulations, along with the standard error of the mean. The cost for each separate simulation is given by the transformed distance for the best individual in the corresponding population.

3.2 Distribution of cost effects

Figure 8, which shows the distribution of cost effects at different cost levels, displays a clear distinction between the dynamics for the consensus and non-consensus encodings. Due to parts of the genome not affecting the phenotype, the amount of neutral mutations is higher in the non-bijective than in the bijective cases, which is signified by the smaller areas enclosed by the curves. Mutations on the consensus encodings also tend to produce strings which have lesser negative effects, i.e. mutations which cause a higher cost, than mutations on the bijective encodings. In other words, the ratios for the bijective maps tend to evolve towards assuming a wider tail near the lower end of the range, whereas the same cannot be said for the consensus encodings. Still, being more robust towards highly negative mutations does not imply having more positive effects. Instead, the bijective encodings produce mutations which are positive to a larger extent than the non-bijective encodings. Furthermore, both consensus encodings do not initially (< 1000) have the same distribution of cost effects, but assume the same shape for lower cost levels, with minor deviations. In observing the distribution of fitness effects by generation, as in Figure 10, the consensus sequences behave the same with respect to each other for each separate search space. This is even though the mean cost for the separate encodings often is different.

The bijective encodings behave similarly with respect to the shape of the distributions at different cost levels. The Gray code encoding produces slightly mildly negative effects more frequently, whereas the Binary encoding is prone to produce more effects which are closer to the lower end of the range. In most intervals and transformations, the Gray code scheme produces slightly more positive mutations, although largely such that the effects are only minorly positive.

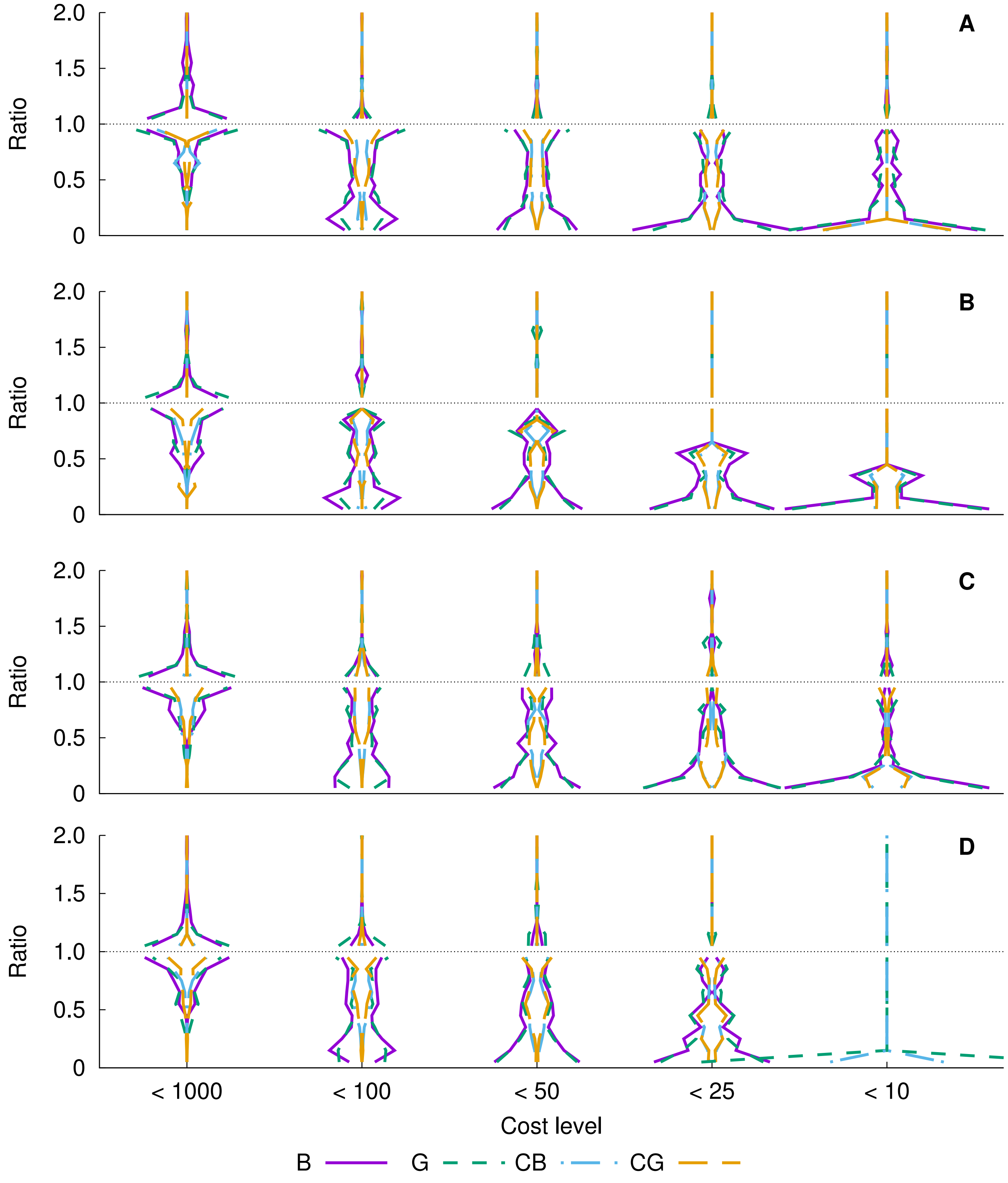

Figure 8: **A: Linear B: Stairs C: Moguls D: Sharktooth**
Distribution effects in binned intervals of 0.1. The ratio denotes the cost of the individual before mutation, divided with the cost after. The inner horizontal axis gives the frequency of mutation effects for the corresponding ratio. All data are produced by mutating every individual with a parent from a higher cost level 10000 separate times, for a maximum of 100000 data points. Intervals do not overlap; for example, < *1000* signifies the interval $(100, 1000]$. Neutral mutations as well as effects of magnitude > 2 are excluded in this figure. The lack of data in **D** stems from the fact that no simulations of the missing encodings manage to reach that cost level.

3.3 Robustness to various mutation rates

Varying the mutation rate from the standard rate, as in Figure 9, shows that the Binary and Gray code schemes are more susceptible to higher mutation rates, whereas the consensus encodings show significantly more robust traits. The consensus sequences contain a number of coding sequences correlating with the mutation rate, scaling from fewer coding sequences on average for higher ones, to more for lower ones (figure not included).

The Gray code scheme appears as to have a preference for lower mutation rates compared to all other encodings. In Figure 9C and D, it does however show a negative trend with a much lower mutation rate ($\times 1/10$), as indicated by the slope of curve. The slightly lower rate ($\times 1/2$) performs on par with the standard rate in all cases, which is not the case for the Binary encoding. For the higher rates, both Binary and Gray perform equally bad.

The consensus sequences show tendencies to be biased toward a slightly higher mutation rate ($\times 2$) in all of Figure 9A-D. Even so, they manage to on average reach roughly the same cost level for all mutation rates.

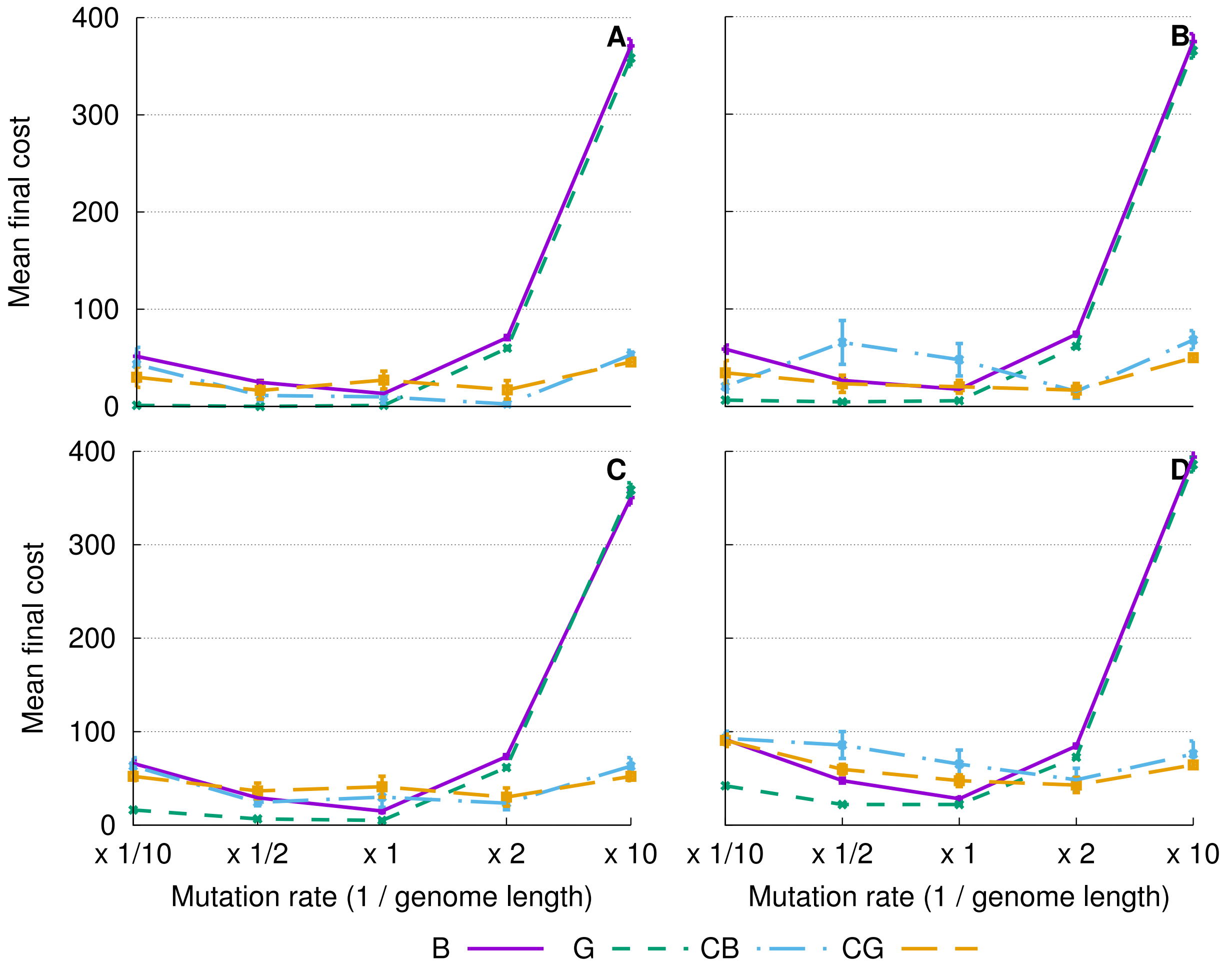

Figure 9: **A: Linear B: Stairs C: Moguls D: Sharktooth**
Mean final cost values over 100 separate simulations per encoding. The mutation rate is modified by a given multiplicative factor (horizontal axis). Error bars show the standard error of the mean for the cost values at generation 100000.

4 Discussion

4.1 Evolution is determined by the accessible phenotypes

4.1.1 Binary and Gray code encodings evolve by a different distribution of phenotype alterations

In comparing the different cases, a clear result is the superiority in the ability of the Gray code scheme to solve the problems given; it in particular fares significantly better than the consensus encodings, and in all but the `Sharktooth` transformation better than the Binary encoding. In contrast to the Gray code scheme, the Binary scheme appears to get its strength as an encoding primarily by being able to take larger steps in phenotype-space. As Figure 7 shows, when the search space changes under the different transformations, the Binary code performs qualitatively the same in most cases, but with a slightly higher mean cost in Figure 7D. This indicates that the Binary encoding is able of overcoming the induced barriers mainly by taking large steps in phenotype space in all cases.

For all but the `Linear` transformation, the landscapes appear to in various degrees remove the benefit of the Gray code scheme, i.e. not having Hamming cliffs, as it struggles with the altered complexities. However, the Gray code scheme is like the Binary encoding able to perform large translations in phenotype-space as well, which makes it probable that the reason for the Gray code outperforming the Binary scheme is the ability to take *both* small and large steps, whereas the Binary has a more rigid distribution of possible steps. When the search spaces are transformed in especially Figure 7C and D, an advancement in cost requires larger changes in the phenotype to overcome the induced barriers, which explains why the Gray code scheme is somewhat hampered – the importance of being able to make small phenotype changes is reduced. Still, being able to take smaller steps which do not have drastically negative consequences, i.e. having near-neutral mutations, allows the population to spread out near the current local optimum to a larger extent than for the Binary scheme. As taking both small and large steps is a significant separator of the Gray code from the Binary, it is an important factor for the Gray code producing better results.

Figure 9 shows that the Gray code scheme generally performs equally well with a slightly lower mutation rate ($\times 1/2$) as with the standard one. This indicates that the scheme primarily evolves by manipulating single bases, which would explain the Gray code encoding being able to outperform the Binary in the `Stairs` transformed search space, which effectively forces the bijective encodings into having neutral mutations also when the genotype is affected. In particular, this shows the significance of neutral mutations in a somewhat unexpected sense; the Gray code encoding being able to perform a cost neutral random walk in the vicinity of the current state gives it a higher probability of reaching a better optimum. In other words, the Gray coded strings can then effectively random walk until a state from which a translation to a lower plateau is reached. The Binary scheme would in principle be able to do the same, but is because of its design hampered with respect to what adjacent states are reachable from the current one, i.e. the Binary scheme is unable to randomly walk within the vicinity of the current state as proficiently as the Gray code scheme. It instead relies more on multiple bit flips in order to attain desired states. The dynamics of the encodings can be emphasised further inspection of fig. 8, where the Binary and the Gray code schemes have very similar distributions with respect to the cost effects. Regardless of this, the Gray code performs better, indicating that it is not solely the distribution of possible jumps that is important, but also the connection between states.

4.1.2 Consensus encodings evolve by adding and removing random numbers

The consensus encodings behave similarly under all search space transformations in that they all take about the same number of generations before starting to effectively decrease in cost. This indicates that the consensus sequences have a tendency to evolve *evolvability*, and that two things are likely needed to be balanced: 1) the number of coding sequences, and 2) the relative difference between the coding values. As these two points determine the available translations in phenotype-space that the individuals are able to undergo, it is a significant factor for the ability to improve.

To expand on this, the consensus encodings do not show significant enough differences in performance. Instead, the reason for their ability to evolve and solve the problem seems to depend on their inherent structure, i.e. by being able to average over a set of coding sequences. As the evolution is mostly governed by the destruction and creation of coding sequences, the phenotype alterations essentially consist of adding and removing quasi-random numbers.

In principle, the consensus sequences should be able to perform both small and large jumps in phenotype space directly by their structure by either manipulating the coding bases to cause a significant change within the separate coding sequence (larger translations in integer-space), or by modifying moderately-valued bases (shorter translations). In practice, however, the consensus encodings do not have close access to the appropriate phenotypes, and thus fail to perform on par with the Gray code scheme. They can also not, due to the averaging of the coding sequences, take as large steps as their bijective counterparts, which could be a limiting factor for their ability to evolve. In particular, because the random numbers which can be generated by introducing respectively destroying coding sequences depend on the current state, the consensus sequences are like the Binary encoded strings limited in what phenotypes are possible to be obtained practically. The low probability of inverting any of the coding bits required to drive the evolution makes it too hard for the consensus encodings to competently evolve, even though they in principle can assume all phenotypes the bijective encodings can assume, in multiple ways.

Finally, that the consensus encodings have trajectories which have a higher initial cost (see Figure 7) is due to many genes being initialized with multiple coding sequences, so that the phenotypic value of the genes average at around $1023/2 \approx 512$. With a genome length of 1000 bits as well as a start sequence length of six bits, a start sequence can be estimated to occur on average roughly once per $(1/2)^{-6}$ bases. Accounting for sequences not being allowed to overlap, as well as the length of the coding sequences themselves, gives an estimate of roughly 12 start sequences per gene. Because of this, the initial cost of the consensus sequences is as high as it is.

4.2 Small differences in the distribution of mutation effects cause significant differences in performance

From the distributions seen in Figure 8, although the Binary and Gray code schemes initially behave differently, they ultimately assume roughly the same behaviour. However, even though the distributions show small differences between the Gray code and the Binary schemes, the encodings still perform significantly differently from each other. The amount of neutral mutations on the individuals can also be seen in the difference in the area under the curve between the bijective and non-bijective encodings.

When the Gray code and Binary schemes nonetheless do ultimately perform roughly similar to each other, as in Figure 8D, the Gray code can still be seen to consistently produce mutations with better results than the Binary scheme, indicating that the Gray coded strings are the only ones undergoing significant improvement at that cost level. Even so, also the Binary consensus encoding manages to reach the lowest cost level (< 10), even though no significant positive mutation effects are visible in the next-to-lowest cost level.

4.3 The consensus structure allows for higher robustness

Consensus sequences are clearly favoured with respect to their ability to evolve under various mutation rates. Especially under high rates, both the bijective encodings fare far worse, whereas the consensus encodings appear to be biased towards a slightly higher mutation rate. The bias is likely due to consensus encodings having a proportionally lower amount of bases which individually affect the phenotype. They therefore require a slightly higher mutation rate to invert significant bits; however, as the mutation rate stands in relation to the genome length, the slightly higher rate ($\times 2$) for the consensus sequences corresponds to a sub-standard mutation rate for the bijective maps.

Because of the averaging approach, if many genes are present, not only are individual inversions of bits less significant, but the least significant bits in each coding sequence become redundant. It is also because of this approach, i.e. the collaboration between coding sequences, that the consensus sequences are able to adapt.

5 Concluding remarks

It has here been shown that the choice of encoding further plays an important role for the performance of GAs. In agreement with previous statements, bijective maps seem favourable in many cases, as the Gray code scheme comes out as superior in performance in all investigations when the mutation rate is set to unity of the genome length or slightly less. However, that the consensus encodings manage to produce results on par with the Binary encoding signifies that bijectivity is not the sole aspect of choosing an encoding.

Instead, *how* the search space is traversed, and what types of translations in phenotype-space can be performed by the encoding is also of essence. That the Gray code scheme comes out so superior seems to be because of its ability to take both smaller and larger jumps in this space, and in particular because of being able to access adjacent phenotypes. Introducing an encoding which is able to do so with another distribution of accessible states depending on the current one might be a possible topic for future research.

It has also been noted that the consensus sequences produce more stable results under different mutation rates than the bijective encodings, which emphasises their adaptive structure. The Gray code and Binary schemes are, because of their bijectivity intrinsically unable to adapt to environments with higher, and presumably also lower, mutation rates. The dynamic structure of the consensus encodings makes the consensus encodings more flexible, and therefore possibly better in changing environments.

The analysis has shed further light on the dynamics of both bijective and non-bijective encodings, as well as their similarities and differences. Hopefully, this can be used to broaden the understanding of the deep complexity of GAs, but also of evolutionary dynamics in general.

Acknowledgments

Many thanks to my main supervisor, Carl Troein, who gave me such great independence with my project, and who showed me that details do matter – sometimes the most. I am also grateful for Klara Berg, to whom I could always talk to about anything. Also my office colleagues deserve a mention for the many inspiring and interesting discussions had during the course of the project; in particular Frank Johansson and Mårten Bertenstam, thank you. However, my greatest gratitude belongs to Adriaan Merlevede, who was always there, and prevented me from trying to fly too high. Oh, and thank you, Julian Melgar – thank you!

A Appendix

A.1 Distribution of cost effects by generation

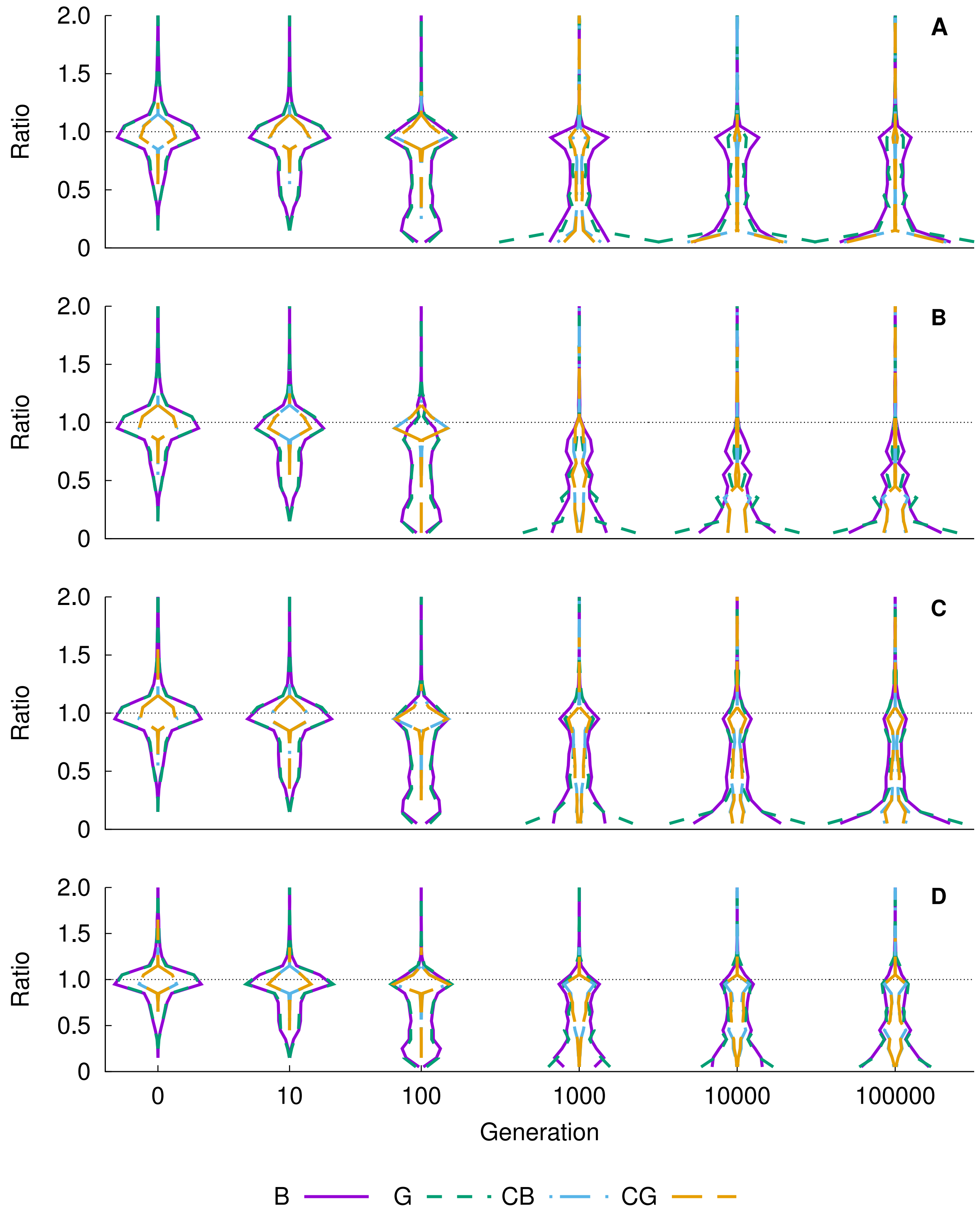

Figure 10: **A: Linear B: Stairs C: Moguls D: Sharktooth**
Violin plot analogous to Figure 8, but with the mutation effects given by generation.

References

[1] Yucheng Hu and Gongrong Yang. “Sequence Evolution Under Constraints: Lessons Learned from Sudoku”. In: *Journal of Computational Biology* 23.00 (2016), pp. 1–11.

[2] Darrell Whitley. “A Genetic Algorithm Tutorial”. In: *Statistics and Computing* 4.00 (1994), pp. 65–75.

[3] David E. Goldberg. “The Theory of Virtual Alphabets”.

[4] David E. Goldberg. “Real-coded Genetic Algorithms, Virtual Alphabets, and Blocking”. In: *IlliGAL Report* 90001 (1990).

[5] Theodore W. Manikas and James T. Cain. “Genetic Algorithms vs Simulated Annealing: A Comparison of Approaches for Solving the Circuit Partitioning Problem”. In: *Technical Report* 96.101 (1996).

[6] John W. Mann and George D. Smith. “A Comparison of Heuristics for Telecommunications Traffic Routing”. In: ed. by V.J. Rayward-Smith et al. Wiley, 1996, pp. 235–254. ISBN: 0471962805.

[7] Christopher Jennison and Nuala Sheehan. “Theoretical and Empirical Properties of the Genetic Algorithm as a Numerical Optimizer”. In: *Journal of Computational and Graphical Statistics* 4.00 (1995), pp. 296–318.

[8] Michael O’Neill and Conor Ryan. “Grammatical Evolution: Evolving Programs for an Arbitrary Language”. In: *IEEE Transactions on Evolutionary Computation* 5.4 (2001), pp. 349–358.

[9] Michael Levin. “The Evolution of Understanding: A Genetic Algorithm Model of the Evolution of Communication”. In: *BioSystems* 36.00 (1995), pp. 167–178.

[10] Christopher W. Beck et al. “A Genetic Algorithm Approach to Study the Evolution of Female Preference Based on Male Age”. In: *Evolutionary Ecology Research* 4.00 (2002), pp. 275–292.

[11] Günter P. Wagner and Lee Altenberg. “Complex Adaptations and the Evolution of Evolvability”. In: *Evolution* 50.3 (1996), pp. 967–976.

[12] Lee Altenberg. “The Evolution of Evolvability in Genetic Programming”.

[13] Bärbel M. R. Stadler et al. “The Topology of the Possible: Formal Spaces Underlying Patterns of Evolutionary Change”. In: *Journal of Theoretical Biology* 213.00 (2001), pp. 241–274.

[14] Keith E. Mathias and Darrell L. Whitley. “Sequence Evolution Under Constraints: Lessons Learned from Sudoku”. In: *Journal of Computational Biology* 23.00 (2016), pp. 1–11.

[15] Xinjie Yu and Mitsuo Gen. *Introduction to Evolutionary Algorithms.* Springer Publishing Company, Incorporated, 2012. ISBN: 144712569X, 9781447125693.

[16] R. Shipman, M. Shackleton, and I. Harvey. “The Use of Neutral Genotype-Phenotype Mappings for Improved Evolutionary Search”. In: *BT Technology Journal* 18.4 (2000), pp. 103–111. DOI: 10.1023/A:1026714927227.

[17] D. H. Wolpert and W. G. Macready. “No Free Lunch Theorems for Optimization”. In: *Trans. Evol. Comp* 1.1 (1997), pp. 67–82. URL: http://dx.doi.org/10.1109/4235.585893.

[18] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* 1st. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN: 0201157675.

[19] Riccardo Poli et al. “Genetic Programming: 5th European Conference, EuroGP 2002 Kinsale, Ireland, April 3–5, 2002 Proceedings”. In: ed. by James A. Foster et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. Chap. Allele Diffusion in Linear Genetic Programming and Variable-Length Genetic Algorithms with Subtree Crossover, pp. 212–227. ISBN: 978-3-540-45984-2. DOI: 10.1007/3-540-45984-7_21. URL: http://dx.doi.org/10.1007/3-540-45984-7_21.

[20] R. Srikanth et al. “A Variable-length Genetic Algorithm for Clustering and Classification”. In: *Pattern Recognition Letters* 16.8 (1995), pp. 789–800. DOI: http://dx.doi.org/10.1016/0167-8655(95)00043-G.

[21] Monica Patrascu, Alexandra Florentina Stancu, and Florin Pop. “HELGA: A Heterogeneous Encoding Lifelike Genetic Algorithm for Population Evolution Modeling and Simulation”. In: *Soft Computing* 18.12 (2014), pp. 2565–2576.

[22] Lothar M. Schmitt. “Theory of genetic algorithms”. In: *Theoretical Computer Science* 259.1–2 (2001), pp. 1–61. DOI: http://dx.doi.org/10.1016/S0304-3975(00)00406-0.

[23] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* Cambridge, MA, USA: MIT Press, 1992. ISBN: 0262082136.

[24] S. Kauffman P. Alberch J. Campbell B. Goodwin R. Lande D. Raup L. Wolpert J. Maynard Smith R. Burian. “Developmental Constraints and Evolution: A Perspective from the Mountain Lake Conference on Development and Evolution”. In: *The Quarterly Review of Biology* 60.3 (1985), pp. 265–287.

[25] Walter Fontana and Peter Schuster. “Continuity in Evolution: On the Nature of Transitions”. In: *Science* 280.5368 (1998), pp. 1451–1455. DOI: 10.1126/science.280.5368.1451.

[26] Jeremy A. Draghil et al. “Mutational Robustness can Facilitate Adaptation”. In: *Nature* 463.1 (2010), pp. 353–355. DOI: doi:10.1038/nature08694.

[27] Kiersten Lo and Stephen T. Smale. “Generality of a Functional Initiator Consensus Sequence”. In: *Gene* 182.1–2 (1996), pp. 13 –22. ISSN: 0378-1119. DOI: http://dx.doi.org/10.1016/S0378-1119(96)00438-6.