Published in final edited form as:

Inf Sci (Ny). 2015 June 10; 306: 88–110. doi:10.1016/j.ins.2015.02.012.

Forced evolution *in silico* by artificial transposons and their genetic operators: The ant navigation problem

Leonid Zamdborg¹, David M. Holloway², Juan J. Merelo³, Vladimir F. Levchenko⁴, and Alexander V. Spirov^{1,4,5,*}

¹ Department of Applied Mathematics and Statistics, The Stony Brook State University, NY

²Mathematics Department, British Columbia Institute of Technology, 3700 Willingdon Avenue, Burnaby, B.C., Canada, V5G 3H2

³Departamento de Arquitectura y Tecnología de Computadores, University of Granada, Granada, Spain

⁴The Sechenov Institute of Evolutionary Physiology and Biochemistry, 44 Thorez Ave., St. Petersburg, 194223, Russia

⁵Computer Science Department and Center of Excellence in Wireless & Information Technology, The Stony Brook State University, NY

Abstract

Modern evolutionary computation utilizes heuristic optimizations based upon concepts borrowed from the Darwinian theory of natural selection. Their demonstrated efficacy has reawakened an interest in other aspects of contemporary biology as an inspiration for new algorithms. However, amongst the many excellent candidates for study, contemporary models of biological macroevolution attract special attention. We believe that a vital direction in this field must be algorithms that model the activity of “genomic parasites”, such as transposons, in biological evolution. Many evolutionary biologists posit that it is the co-evolution of populations with their genomic parasites that permits the high efficiency of evolutionary searches found in the living world. This publication is our first step in the direction of developing a minimal assortment of algorithms that simulate the role of genomic parasites. Specifically, we started in the domain of genetic algorithms (GA) and selected the Artificial Ant Problem as a test case. This navigation problem is widely known as a classical benchmark test and possesses a large body of literature. We add new objects to the standard toolkit of GA - artificial transposons and a collection of operators that operate on them. We define these artificial transposons as a fragment of an ant's code with properties that cause it to stand apart from the rest. The minimal set of operators for transposons is a transposon mutation operator, and a transposon reproduction operator that causes

© 2015 Published by Elsevier Inc.

*Corresponding author: State University of New York at Stony Brook, Computer Science Department and Center of Excellence in Wireless & Information Technology, Stony Brook University Research & Development Park, 1500 Stony Brook Road, Stony Brook, NY 11794-6040, Tel: (631) 632-5582, Fax: (631) 632-4653, Alexander.Spirov@gmail.com.

Publisher's Disclaimer: This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final citable form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

a transposon to multiply within the population of hosts. An analysis of the population dynamics of transposons within the course of ant evolution showed that transposons are involved in the processes of propagation and selection of blocks of ant navigation programs. During this time, the speed of evolutionary search increases significantly. We concluded that artificial transposons, analogous to real transposons, are truly capable of acting as intelligent mutators that adapt in response to an evolutionary problem in the course of co-evolution with their hosts.

Keywords

evolutionary computations; genetic algorithms; artificial transposons; animats; ant pathfinder

1 Introduction

Many areas of Evolutionary Computation (EC), especially genetic algorithms (GA) and genetic programming (GP), were inspired by ideas from evolutionary biology. However, modern evolutionary biology has since advanced considerably, revealing that Darwinian evolution (microevolution) is apparently a particular case of the greater mechanisms of macroevolution. Many current branches of research in evolutionary computation implement the evolution of mechanisms, such as functional programs, neural networks, decision trees, cellular automata, L-systems, and finite state automata. The recent achievements in genomics seem more appropriate as an inspirational model for these domains than the classical set of Darwinian algorithms [2, 7, 13, 33, 37, 40].

This stimulates us to select, formalize and apply to EC new evolutionary mechanisms that may help to simulate the creative, heuristic and self-organizing character of biological evolution [64 - 76].

We concentrate on the mechanisms of natural genetic engineering, whose key actors are mobile genetic elements (MGE) [40, 41, 56, 57]. The basic idea can be outlined as follows:

- The genome of every organism has mechanisms for genomic rearrangement;
- These mechanisms are activated during periods of evolutionary crisis;
- The mechanisms cause multiple systemic rearrangements of genomes within a few generations.

The key players of natural genetic engineering are mobile genetic elements (synonymous or related terms are jumping genes, selfish DNA, transposons and retroelements) [21, 38, 41, 42]. Many biologists speculate that processes in the world of transposons, existing on a substratum of genomes of the greater biological community, are the main source of evolutionary creativity [10, 26, 56, 57].

It has been estimated that the majority of the DNA in higher organisms is neither translated into proteins nor involved in gene regulation, and is simply "junk" DNA. The bulk of junk DNA is composed of intermediate-repeats comprised of DNA elements that are able to move (or transpose) throughout the genome. These mobile DNA elements are sometimes termed "selfish", since they do not appear to directly benefit the host. Their behavior looks

parasitic: they jump between different sections of a genome in order to propagate themselves, a behavior that seems at first glance to be usually detrimental to their host. However, they do appear to benefit the host organism by providing genomic rearrangements that permit the evolution of new genetic networks.

There are several groups of genomic parasites, with transposons forming the most sophisticated one. A transposon is a mobile piece of DNA that is flanked by terminal repeat sequences and typically bears genes coding for transposition functions. "...it is now recognized that a significant portion of the genome of any eukaryote is composed of "selfish" or "parasitic" genetic elements, which gain a transmission advantage relative to other components of an individual's genome, but are either neutral or detrimental to the organism's fitness" [21]. Arguments have been made that the structure of eukaryotic genomes, including the abundance of transposons, repetitive DNA and introns, provide high evolvability despite possible detrimental effects on the host [56, 57].

Transposons are ubiquitous and may comprise up to 45% of an organism's genome [21, 38, 41, 42]. DNA sequences of transposon origin can be recognized by their palindrome endings, flanked by short, non-reversed, repeated sequences resulting from insertion after staggered cuts. Many transposons have a unique DNA site that acts as a forwarding address, directing the transposon to a complementary site elsewhere in the host genome. There are usually multiple copies of any given DNA site in the host genome, and the transposon will attach to a proper site in a random manner. Transposons may grow by acquiring more sequences; one such mechanism involves the placement of two transposons into close proximity so that they act as a single large transposon incorporating the intervening code.

During the last two decades, results were obtained that give credence to the idea that transposable elements are a major source of genetic change, such as the creation of novel genes, the alteration of gene functions, and the genesis of major genomic rearrangements [16, 17, 21, 38, 41, 42, 50, 80]. The long coexistence of transposable elements in the genome is expected to be accompanied by host - transposon co-evolution. There exists an opinion that transposons should be treated as non-random (or rather "intelligent") mutators. Furthermore, transposons can be considered an evolutionary "SOS-crew", cooperatively acting "in emergency", during periods in which the host is experiencing genomic stress [38, 44].

Modern evolutionary biology and evolutionary genetics (and genomics) amassed a great deal of knowledge regarding the complex mechanisms of genomic rearrangement governed by transposons [1, 10, 21, 26, 41, 42, 56, 57]. From an EC point of view, the most sophisticated of these mechanisms may be treated as a local evolutionary search. By this, we mean the mutational hot spots generated by transposons. A promising approach is to include the world of selfish elements in the standard toolkit of EC, and to use the objects/procedures for maintaining and manipulating transposons as a new branch of EC. On the other hand, as was pointed in [7], EC "would enable simulations to explore how endosymbiosis or independently replicating transposable elements might affect the complexity of the genome."

Several teams have applied algorithms inspired by jumping genes (summarized in the next section). However, these have been mainly tested on benchmarks with binary or real number representations, and mainly on commonly known mathematical test-functions. In this article, we apply modern evolutionary ideas to well-known benchmark tests implemented (implicitly or explicitly) via symbolic strings. In particular, we propose an improvement on the animate pathfinder (artificial ant) test [23, 27], a commonly used benchmark test [8, 15, 18, 22, 27, 28, 34, 39, 79]. Several authors have recently started to use such animated navigation problems (e.g. the Tartarus problem, an extension of the ant-pathfinder) to implement ideas from current evolutionary biology [6, 25, 82]. We are positive that symbolic string chromosomes, in particular, are the most natural way to apply transposition inspired algorithms, as well as allowing us to analyze why the algorithms are effective.

Specifically, we enhance standard genetic algorithms by including artificial transposons and genetic operators that mimic some of the essential characteristics of real transposon behavior. Because populational and co-evolutional aspects of transposon activity are thought to be essential for understanding macroevolutionary mechanisms, we study in detail the population dynamics of artificial transposons for our version of the ant problem.

The paper is organized as follows: Background and related works are given in Section 2; our approach and the ant-pathfinder test are described in Section 3, Methods and Approach; the performance of our proposed MGE-approach is empirically studied in Section 4; future directions are outlined in Section 5; and finally, conclusions are given in Section 6.

2 Background and related works

The field of EC is attempting to make evolutionary algorithms more effective by applying techniques inspired by the latest achievements in biology. These attempts have been termed *genomic algorithms*, *bacterial algorithms* or even “genetic” (quotes intended) algorithms. Many other techniques taken from biology, such as transposition, host-parasite interaction and gene-regulatory networks have also been applied to evolutionary computation. There is no uniform nomenclature, however, and sometimes the same terms are applied to different methods, or the other way around.

These techniques can be divided into three broad categories:

- *Host-parasite methods*. These methods are based on the co-evolution of two different populations, one of them acting as a “parasite”, and the other acting as the “host”; the parasites usually encode a version of the problem domain, and the hosts encode the solution to the problem [14, 20, 24, 46 -48, 51 - 53]. These methods have been used mainly to evolve sorting networks, but similar approaches have been used, for instance, to evolve players of the *Othello* game. In this case, the approach is not significantly different from other co-evolutionary approaches, in which solutions and instances of a problem are co-evolved.
- *Transposition operators*. These are sometimes known as “bacterial” algorithms [19, 29 - 31, 45, 58 - 63]. The basic idea of this approach is to make intra-chromosome crossovers, that is, the crossover of a chromosome with another part of itself, or

else an asymmetric crossover, in which a donor chromosome transfers a part of its genetic material to an acceptor chromosome. Particularly, in the GP domain the algorithms use previously evolved GP trees as transposons that create copies of themselves to expand their own genotype [29 - 31]. In some cases, these operators seem to be better than classical genetic algorithms for combinatorial optimization problems.

- *The "jumping gene" paradigm.* The authors of the jumping gene (JG) paradigm have studied the transposon phenomenon in more detail and proposed a series of computational JG operators [12, 35, 36, 54, 55, 81]. They found that the JG paradigm is a compatible evolutionary computing scheme when speed in convergence, diversity, and accuracy are simultaneously required [12]. JG has also been applied to some real-world optimization scenarios, such as passive circuits systems, planar UWB monopole antenna design, power voltage control systems and wireless local area network optimization [see 36 for further references].
- *Other biological approaches.* Luke et al. [40] use a method similar to genetic regulatory networks to evolve finite state automata that represent a language grammar, a class of objects that cannot be easily represented using serial, bitstring, genetic algorithms [43]. Burke et al. [11] try to make the evolutionary process closer to its real molecular genetics base, by having a 4-letter genetic alphabet, transduction processes, and variable-length genetic algorithms. In both cases, it is not a general-purpose technique; they only draw some elements from biology to solve problems that would be difficult to solve in other cases.

2.1 Key Characteristics of Transposons as an Inspirational Model

Transposons act as non-random mutators for host genes, and can cause a number of different types of mutations, via different mechanisms:

1. *Non-random insertions.* The insertion of transposons is in some cases targeted rather than random [41, 42]. Certain regions of the host DNA are favored for insertion; these are referred to as *hot-spots* and tend to be specific to a single transposon or a family of such. The specificity of insertion is based on factors encoded by transposon genes.
2. *Recombinational hot-spots.* Transposable elements serve as recombinational hot-spots, allowing the exchange of genetic material between unrelated chromosome sequences [41]. Because they are so abundant, they can mediate large scale chromosome restructuring by means of homologous recombination between similar transposable elements at distant locations.
3. *Mutational hot-spots.* Transposable elements also serve as mutational hot-spots, allowing an increase in the rate of mutation at the level of single genes. For instance, two key enzymes controlling the generation of the inexhaustible variety of antibodies in the immune system are relics of an ancient transposon [1]: much of antibody variability is due to the way antibody genes are assembled, by joining three separate sequences (denoted V, J, and D; each of which comes in many

variants) into a single antibody gene. The enzymes doing this assembly, Rag1 and Rag2, work just like the transposase enzymes that mobilize transposons.

4. *Rearrangements of host genetic networks.* In addition to moving themselves, transposable elements occasionally move or rearrange neighboring DNA sequences of the host genome. It has been estimated that 80% of spontaneous mutations are caused by transposons [41, 42]. Repeated sequences, resulting from the activity of mobile elements, range from dozens to millions of copies per genome. Transposons “strew” the genome with different regulatory sequences [80]. By means of this, transposons can rearrange hosts’ genetic networks [41, 42, 80]. Transposons may capture genes and move them wholesale to new parts of the genome. They make possible DNA shuffling that can place genes in new regulatory contexts and, thus, possibly, give them new roles. But what is more, when two transposable elements recognized by the same transposase integrate into neighboring chromosomal sites, the DNA between them can be subjected to transposition. As far as this process provides a particularly effective pathway for the duplication and movement of genes, these elements can help to create new genetic ensembles.

Transposons co-evolved with their host genomes as a result of selection favoring transposons, which introduced useful variation through gene rearrangement. “Smart” genetic operators evolved in the same manner [56, 57]. It is likely that transposons represent these highly co-evolved genetic operators. The possession of smart genetic operators must have contributed to the explosive diversification of higher organisms by providing them with a capacity for natural genetic engineering.

In designing artificial evolution, it would be worthwhile to introduce such genomic parasites, in order to facilitate the sharing of the code that they bring about. We have previously suggested implementing artificial transposons to improve a GA search [64 -66], and this idea has received some response [12, 29, 30, 35, 36, 54, 55, 81]. Meanwhile, other authors have proposed and tested operators that imitate some of the properties of biological transposons [58 - 63]. The distinguishing feature of our approach is the exploitation of the deeper evolutionary aspects of transposons [72, 74 - 76]. For example, imitation of the co-evolution genomic parasites experience with the host genomes that they populate. Every genome, and as such every host, has a set of mobile mutational agents (genomic parasites) of its own. Host and parasite populations co-evolve, with sometimes this co-evolution taking the form of an “arms race”.

2.2 The Vansant – Reynolds concept

The evolutionary biology findings of Vansant and Reynolds [80, 16, 50] have been a major inspiration for us in this project. In particular, they found that one of the most common types of genetic “junk” in the human genome turns out to contain a working copy of a genetic switch that activates other genes. This transposable element, Alu, could have played an important role in the evolution of primates on the way to human beings. From time to time, this element inserts copies of itself randomly into the genome. Over the past 30 to 60 million years, these insertions have occurred repeatedly, leaving roughly a million copies of Alu scattered throughout the human genome. Elsewhere in the genome, there are nearly identical

sequences that function as anchor points for proteins that bind to hormones and that, therefore, provide a way for hormones to turn genes on and off. The authors found that the Alu sequence also binds to a hormone receptor (for a factor called retinoic acid, RA), which activates genes at the appropriate times during development. The results have provided clear evidence that most Alus could have the potential to regulate human genes. With a million Alus dispersed randomly through the genome during the course of primate evolution, at least some are likely to have landed where they could regulate a nearby gene. Usually the effect would be harmful, but occasionally it might produce a beneficial genetic novelty.

An interpretation of these biological findings in terms of evolutionary computations are summarized in Figure 1. For a particular example, suppose that the Alu transposons carry elements of code preconditioning a gene for RA action. If a given gene gains such an element at an appropriate position, it could then acquire a new regulatory connection (activation or repression by RA in this case). During human evolution, Alus have multiplied and spread their regulatory elements through the human genome, setting up preconditions for more and more genes to become targets for RA regulation. In other words, Alus bias the human gene network evolution toward establishing more and more RA-dependent regulatory connections.

It appears that more diverse target elements spread in host genomes by different transposons [16, 50], and that the biasing of host evolution by transposons is a general and common mechanism of biological evolution. We consider this mechanism to be a way of constraining the evolutionary search space. In this study, we aim to test this mechanism within EC.

This publication is our first step in the direction of developing a minimal assortment of algorithms that simulate the role of genomic parasites. Specifically, we started in the domain of genetic algorithms (GA) and selected the artificial ant problem as a test case. This navigation problem is widely known and possesses a large body of literature. We have added new objects to the standard toolkit of GA; these new objects are artificial transposons and a collection of operators that operate on them.

2.3 The Ant Problem

We choose the ant-pathfinder problem for two reasons. First, it is a classic and well-characterized benchmark test, and, second, it can be easily implemented via symbolic strings, either implicitly (Finite-State Automata ant, FSA-ant) or explicitly (our version of the ant problem, with symbolic string chromosomes). Such symbolic string representations are more similar to how nature encodes transposons as mutators, as compared to other implementations.

The artificial ant problem is a simulation of ant navigation aimed at passing through a labeled trail (Figure 2; nicknamed “The John Muir Trail” in the UCLA experiment [23]). In [23], the artificial ant has to follow the John Muir trail placed on a grid. The ants are simple FSA or an Artificial Neural Network (ANN) that can move along the grid and test their immediate surroundings. The trail starts off as quite easy to follow, and gradually becomes more difficult as the turns become more unpredictable and gaps appear. Each black (labeled) cell is numbered sequentially, from the 1st, directly neighboring the starting cell, through to

the 89th, the last cell. The ant's task is to follow this trail and move across each black cell in sequence.

The artificial ant problem for two variants of trail (the "Santa Fe" and the "Los Altos") is well tested and discussed in publications. In both cases, the trails are carefully designed such that they are easy to navigate at the start, but as the ant proceeds through the trail progressively more gaps and unexpected turns emerge. By the end of the trail there are more gaps than labeled cells. At the beginning, the successful tactic is very simple - only to pick up trail and to move forward. A little further, the ant will need to learn the trick of turning to the right with the trail. Later, it will need to learn the secret of turning occasionally left, and so on.

The ant stands on a single cell and is oriented to the north, south, east, or west. It is capable of sensing the state of the cell directly in front of it. In each time step, the ant must take one of four actions. It may turn left, turn right, move forward one step, or stand still. (When an ant steps onto a black cell, the cell turns white.) After the ant performs its action, it shifts to a new state. Only the internal state of the ant and the state of the cell in front of the ant are used as inputs to the decision table that determines (a) the ant's action and (b) the state it will assume in the next time step. The decision table itself is coded as the ant's genotype (Figure 3B). The ant's score is the number of cells passed by the ant in a given time period.

3 Methods and Approach

3.1 Artificial Transposons

Artificial transposons act as intelligent and sophisticated mutators. They can generate arbitrary procedures of manipulations with the chromosomes of their hosts. In general, these operators can be unitary, binary or plural. Each host has a population of mutators that operate on it. In the simplest case, transposons are the only source of the host's mutations. If a transposon finds a good mutation strategy, then both host and parasite will get a chance for reproduction, in essence producing a virtual co-evolution of hosts and their intelligent mutator-parasites.

It is appropriate here to make some notes on terminology. The *Mobile Genetic Element (MGE) technique* comprises procedures for initialization of mobile genetic elements and procedures for operating with these elements. Hereafter, mobile elements will be referred to as "transposons", whereas the procedures operating on them will be referred to as "MGE operators".

An artificial transposon is a fragment of chromosome which is specially marked, i.e. which has specific coding characteristics sufficient for its recognition. The code block marked as an artificial transposon still functionally belongs to the host, and is part of the host's genetic code; however this code block is transmittable from host to host and is able to mutate and grow according to certain rules, which differ from the other parts of the host's code.

We tested three ways to distinguish transposon code from non-transposon: usage tags (1), marks (2) or definitions (3). We found a *transposon definition* to be the best method in our tests. This *definition* is a description of its sequence. The definition can be static, i.e.

predetermined from the very beginning by the programmer or user, or dynamic, i.e. changing during run-time according some rules predetermined by the programmers. The search procedure called by an operator scans for sequences satisfying this definition; once the sequence is found, the operator performs an action. For instance, one such definition of a transposon sequence is: all palindrome sequences are transposons. In such a case, the search procedure scans chromosomes to find palindromes. If it finds any, then the operator treats that part of the chromosome as a transposon.

Another example of a transposon sequence definition is periodicity of the sequence. That is, let a transposon be any periodic sequence with period N , from $N_{\min}$ ($N_{\min} - 2$) to $N_{\max}$ ($N_{\max} - CL * 0.5$, where CL is Chromosome Length). If so, transposon growth will be the growth of the periodic sequence.

A conceptually similar transposon definition was used by Simoes and Costa [58 - 63]. In that approach, the first (left) flanking sequence of the first transposon in a given chromosome is chosen at random, while the second flanking sequence must be identical or inverse to the first one. The transposon is formed of all the elements between the first element X , leftmost to the first flank and the last element of the second flank.

3.2 Operators for Mobile Genetic Elements

The MGE technique includes a set of operators (called MGE operators) for the manipulation of artificial transposons. These genetic operators can be an alternative to, or used in addition to the classical crossover and point mutation mechanism for genetic material recombination. This technique can be applied equally to GA and Genetic Programming (GP), but in this work we concentrated our efforts on testing the applications of MGE operators in GA.

The minimal set of MGE operators should include the two-place transposition operator and the one-place mutation operator. The two-place MGE operator defines the transmission of a transposon from one host to another, thus realizing the reproduction procedure of that transposon in the gene pool of the host population. The one-place MGE operator in the simplest case is a point mutation of the transposon sequence. The characteristics of the specific test problem (Royal Roads functions, the Ant problem, TSP, etc.) dictate some modifications of MGE operators.

3.3 ANT Programs

Two versions of the ANT problem implementation was used in this study. The first one, the Finite State Automata (FSA) implementation was chosen for historical reference to the original implementation in the UCLA team experiments [23]. In tests with FSA, we also ran a Symbol-String Chromosome (SSC) implementation to complement the original approach. This simplified implementation helped elucidate some of the crucial details of the actions of the artificial transposon and transposition algorithms which were not clear in the original, more complicated FSA model. Both these versions are described below.

While the ant test has been implemented in at least in two different C++ libraries [84], we gave preference to the Patrick Brennan version [9]. This “ANT prog am” was designed in such a way that it isolates, as far as possible, the components of the genetic algorithm from

the trail-following experiment and the ant representation. This allows an easy transition of the GA code out of the ANT application into other applications and tests. Our C version of FSA ant is freely available at <https://github.com/ant-2014/Pathfinder2014>. The Free Pascal version of SSC ant is available upon request.

3.3.1 Finite State Automata Implementation—Patrick Brennan’s ANT program [9] is a PC version of the Join Muir Trail experiments performed by the UCLA team [23]. In the original UCLA experiment, a very large population of ants was used, and the ants were alternatively typed as FSAs or ANNs. For each type of ant, a genome encoded the relevant parameters of the ant, either in the form of a state transition table for the FSAs, or connection weights for the ANNs. Brennan’s ANT program duplicates the part of the John Muir Trail experiment that uses FSAs [9]. The program consists of three major subroutines: *Expose*, *Select*, and *Reproduce* (Figure 3A).

In the *Expose* subroutine, each ant in the population is run against the John Muir Trail. Each ant’s score is recorded.

In the *Select* subroutine, statistics are generated for the previous *Expose* run. Each ant’s score is compared to the maximum score attained in the population. One of two selection strategies is employed to choose a given ant for reproduction. If the user has selected a *truncation* strategy, only the ants with the highest scores are marked for reproduction. If the user has selected a *roulette-wheel* strategy, an ant with a higher score has a greater chance of being marked, but there is a chance that even the highest-scoring ants may not be marked.

In the *Reproduce* subroutine, the genes of those ants that are not marked for reproduction are overwritten by copies of the genes of those ants that are, and then crossover, mutation and MGE-operators are applied.

3.3.2 The Symbol-String Chromosome Implementation of the Ant Problem—

Our SSC implementation of the Ant problem [74] is simplified from the original UCLA FSA experiments [23], and allows us to analyze the transposon dynamics in detail. The SSC algorithm uses sequences of navigation commands, F, R and L (move one step Forward, turn Right and turn Left). The sequences have variable lengths. The sequences are implemented as variable SSCs (Figure 4). Each such chromosome is a string of length n , $\text{Min} < n < \text{Max}$, from a three-letter alphabet, F, R and L. Each ant moves in accordance with a series of commands that are closed in a loop, and the overall number of the commands to perform is restricted to a predefined number ($> \text{Max}$, typically from 330 to 430).

An initial population of the SSC-ants is generated as random 3-letter strings of arbitrary length (between the limits of MinLength and MaxLength). A special procedure arbitrarily adds and deletes letters in the chromosome (within the limits of MinLength and MaxLength).

Implementation of SSC-ant generally follows the FSA-ant implementation in [74], but uses its own versions of the artificial transposon and the transposon algorithms.

3.4 MGE technique for the Ant Problem

We developed new operators to perform the processes of replication, mutation and invasion of transposons into specific sites on chromosomes, as well as for interactions of transposons with the chromosome (interrelations of parasite-host type). These are described below.

3.4.1 Artificial Transposons—The FSA implementation of the ant problem is characterized by a complicated genotype-phenotype mapping. The ant genotype is a binary, integer, or symbolic string, while the ant's phenotype is based on the decision tree encoded by the chromosome. In turn, the table can be represented as a state graph (with nodes being states and edges being transitions). Whereas the chromosome is linear, the corresponding transition graph in the general case is non-planar. Similar implementations are used for such evolutionary tests as the *Tartarus*, the *Bulldozer* and the *Busy Beaver* [3, 49, 77, 78].

The ant binary string-chromosome codes the state transition table of an FSA (see Figure 3B-C). We use a 32 state automaton (ranging in numerical order from state #0 to state #31, Figure 3B) to facilitate comparison with other projects which have used 32 states [9, 23]. All operators start reading and interpreting the table from state #0. For example, state #0 determines one of four actions or instructions, with the number of the next state depending on the binary input value ("0" or "1"). This finite state automation can be represented as a state transition diagram and interpreted as a decision tree but, since references to already-passed states are permissible, the tree can have loops.

Transposons, both in nature and in our approach, are associated with string-chromosomes. We could use artificial transposons to manipulate the ant's chromosome, but this would not be a good analogy to transposon-chromosome interactions in nature. In reality, the interaction of transposons with their host's chromosomes includes the search and recognition of specific sequences (as detailed in the section 2). In nature, transposons scan chromosome sequences for target signs, and a transposon's action aims at changes in host-gene functions favorable for the transposon. For this reason, we found it more promising to use transposons to manipulate the state-transition graph (Figure 3D). While it is impossible to convert such a graph into a linear sequence of states or actions, by definition, it is possible to formulate a set of rules to extract such a sequence from the graph.

One set which we found to be suitable for our purposes consists of the following rules: Let the first element of the extracted string be the ant's action corresponding to the transition from state 0 under condition input=0. The next state will be state *I*. The second element of the extracted sequence will be the action corresponding to the transition from state *I* under condition input=0. These procedures repeat until the sequence traced forms a loop. The extracted sequence can have different possible lengths but will have two possible forms (for simplicity we use alphabet letters, from A to Z) (Figure 5A-B).

These symbol strings serve as "chromosomes" for our transposons, which they manipulate. Each time an MGE operator is applied to a given ant's chromosome, an extraction of the sequence of actions take place. As a result of the MGE-algorithm's operations, the action string is changed. For this reason, another procedure performs the inverse task to write the changes back into the chromosome.

In other words, we consider the half of the decision tree corresponding to a zero binary input value, i.e. when the ant sees a white (empty) square in front of it. This part of the ant's transition state governs its search for black squares. In so doing, we reduce the ant's genome to a symbolic string and substantially simplify the problem. An example of such a (closed) symbolic string (in the L,R,F alphabet) is shown in Figure 5C. The navigation program encoded by this string is optimal for a classic trail (Figure 2).

Within the maximum length of 32 symbolic elements, it is remarkable that these strings are variable-length. This is due to the presence of cycles. For instance, the loop-string LRRLF (Figure 8A) is shorter but functionally equivalent to the one in Figure 5C, because it instructs scanning of the same three neighbor cells and moving one step forward.

For our approach to the ant problem, we *define* a transposon (see section 3.1) as a sequence of symbols with the following properties:

1. The sequence should include M symbols ($M \leq CL$, where CL is Chromosome Length).
2. The sequence (excluding the last element) should not contain NOP elements and internal cycles.
3. The sequence should finish with a NOP command or with a reference to one of the states in this sequence, i.e. an internal cycle.

Examples of symbolic strings with transposons that match this definition are shown in Figure 5C-D.

Because our definition allows two cases of transposon sequences, we found it convenient to distinguish them by name. If the fitting sequence is a cycle, it will be referred to as a *mature transposon*. However, if the fitting sequence terminates with a NOP, we will call it an *immature transposon*. Examples of mature and immature transposons are given on Figure 8 and Figure 5C,D.

Figure 6 outlines the search procedures and classification of transposons described in this section. The C codes for those procedures are in the PATTERN.C subroutine available at <https://github.com/ant-2014/Pathfinder2014>.

During each iteration of the ANT program's execution loop, apart from the MGE operators, the standard mutation and crossover operators act on an ant's binary chromosome as in the Brennan source code [9]. We don't apply any special rules to the "black" half of the decision table (input = 1; See Figure 3B): this is acted on only by standard crossover and mutation operators.

3.4.1.1 SSC-Implementation of Artificial Transposons: As with the FSA implementation (Cf. previous section and section 3.4.2), the SSC implementation of transposons exploits the periodicity in the symbol strings. In other words, the SSC transposon's general view is $(A_1, \dots, A_n) * K$, where $n = 2, 3, 4, 5$; $K=2, 3, 4, 5$. Symbol K is the period, and A_n is an arbitrary symbol (x). The definition is illustrated in Figure 7.

During an evolutionary search, the program subroutines search the transposons in the chromosomes in accordance with the transposon definition presented here. If the transposon sequence is found, then it will be processed by the transposon operators.

3.4.2 Transposon-caused mutagenesis—Transposon activity is interpreted in terms of mutational/recombinational hot spots, in accordance with natural, biological transposon activity (section 2.1). Specifically, one of the very special features of transposons is that they are jumping hot spots. Because of this, artificial transposon activity is a strong example of local searching within a subset of the global GA search. Namely, the initial population of artificial transposons is subjected to mutation and disseminate short code blocks, until they find the first building block. The transposons that cover this building block (for example, the above-mentioned case of the RLLF loop) would have a selective advantage that allows them to spread and become dominant in the host population. In the long run, another transposon will find an additional, or larger, or better building block and will receive a new selective advantage, and so on.

The simplest way to perform such a local search would be with random point mutagenesis of transposon sequences coupled with horizontal transmission of transposons.

Using a definition for transposon sequences, we also have to define rules to change the size and code of a transposon. It is simplest if these rules change a transposon's code such that the code still satisfies the transposon definition. For example, a type of point mutation would be to substitute any symbol in a transposon's code with another from the list L, R, F (Figure 8E).

Our preliminary tests indicate that such an algorithm of transposon mutagenesis makes some improvements to ant learning abilities (results not shown and will be published elsewhere). However, in order to study in detail the population dynamics of artificial transposons as sophisticated mutators, here we maximally restrict the variability of transposons. In the SSC approach, we implement a rule in which transposon mutagenesis does not target all transposon elements, but rather targets the rightmost element of any given transposon. Only the last element of the sequence undergoes change. Unlike standard point mutation, this algorithm is not truly random; it is an example of constrained or context-dependent point mutagenesis.

We use the periodicity of transposon sequences (section 3.4.1.1): the action of the last transposon element is substituted for the action of the K^{th} element of the sequence, counted in reverse order. We tested K values from 3 to 8. To our surprise, this very restricted and targeted form of mutagenesis appears sufficient to improve the ant's evolutionary learning. In the rest of the article, we use $K = 5$ (this is not “magic” and preliminary tests with $K = 4$ and $K = 6$ gave comparable results; data not shown). This transposon definition drastically restricts the transposon's search space and facilitates monitoring of transposon population dynamics. By applying this restrictive definition we exploit a very small part of the manifold of all possible transposon sequences. We tested this restriction and found it to be appropriate for our purposes. As a secondary benefit, this definition makes transposon marking unnecessary.

3.4.3 Transposon growth—With the restricted definition, we do not use any additional algorithms for transposon growth. Preliminary tests showed that mutagenesis on the rightmost element was sufficient for growth of transposon sequence length.

If a terminating NOP element is substituted with an L, R or F, then in most cases the code is changed such that its transposon properties are lost. However, this procedure also gives a chance for the transposon to absorb more elements and grow, as illustrated in Figure 8D.

3.4.3.1 SSC-Ant: mutations caused by transposon growth: Based on our earlier findings [74], our aim was to use the SSC implementation as a simplification of the FSA implementation. In particular, by the restricted rules in the previous section, SSC combines transposon mutagenesis with transposon growth.

By definition, the growth of the SSC transposon is the extension of its periodic sequence to the right along the chromosome. In other words, starting with a sequence satisfying the transposon definition, say, FFxFFx (period=3, length of two periods; Cf. Figure 7), then the transposon will grow, each time by one more element (substituting the host element by the transposon element). In other words, initially FFxFFxxx ..., then FFxFFxFxx ..., then FFxFFxFFx ..., then FFxFFxFFxFxx ..., as illustrated in Figure 9. By doing so, the transposon damages a good navigation sequence of commands, but it may possibly correspond to a new good navigation program.

3.4.4 MGE operators—MGE operators scan a predetermined quota of chromosomes in a population. Successively decoding binary chromosome records, the operator seeks sequences that are identified as transposons (mature or immature, in FSA). Two-place MGE operators allow transmission of a transposon from one ant to another, thus implementing the reproduction procedure of that transposon in the gene pool of the host (ant) population.

3.4.4.1 The FSA-ant Implementation: The one-place MGE operator is a type of point mutation, realized under particular conditions. We define this as a context-dependent (*intelligent*, in a sense) *mutator*. In detail, the operator acts in the following way: if it finds a sequence in the predetermined length range, terminated by a NOP, then this instruction is substituted for one of the three other actions (FWD, RGT or LFT). Specifically, this NOP is substituted for the action from the K^{th} element of the sequence ($K \in 3, 4, 5, 6, 7, 8$), counted in order ($K = 5$ used in all numeric tests in this article). Figure 10A shows a selection from an output file generated by the ant program in verbose mode, illustrating this kind of transposon mutation.

However, if the located sequence is terminated by a reference to the one of the elements inside the sequence (internal cycle, immature transposon), then the MGE operator performs the following operation: the action of this element is substituted for the action of the K^{th} element ($K = 5$), counted backward from the end, the reference being substituted for a random reference to an element outside the sequence. This type of mutation is illustrated in Figure 10B. Note that the mutator uses the same rules for mature and immature transposons. The types only differ in whether the reference from the last symbol is to the first one (mature) or whether the cycle is broken by making a reference from the last state to any state

other than the first (immature, generally destroying the transposon). Transposon growth can proceed for either type. The action of the intelligent mutator results in the accumulation of large length transposons in the population.

For the two-place MGE operator, a pair of ants is first chosen at random. Then, one of them has their chromosome scanned in search of a transposon. If a transposon is found, it is replicated in the other ant's chromosome in the same position as in the source, irrespective of the current content of that location in the chromosome. The chromosome scanning starts from line zero (state#0) and goes on until the first transposon is discovered, at which point scanning ceases irrespective of the content of the remaining, un-scanned chromosome portion. If no transposons are located, scanning terminates when the chromosome record ends. We should emphasize that the two-place MGE operator operates only with mature transposons (See Figure 8).

Figure 11 outlines the procedures for the one-place and two-place MGE operators described in this section. C codes for those procedures are in the VIRUS.C subroutine available at <https://github.com/ant-2014/Pathfinder2014>.

3.4.4.2 The SSC-ant Implementation: As a simpler alternative to the two-place MGE transposition algorithm, we developed a procedure depending on fitness in the population. We imagine that a given transposon accumulates in the population if its hosts have good scores and are accumulating in the population. Two parents (Dad and Mom) are chosen at random from the best part of the population, while two more individuals (Kid1 and Kid2) are chosen at random from the worst part of the population. Then, Kid1 is substituted by a copy of Mom, and Kid2 by a copy of Dad. 75% (usually) of the offspring just undergo simple point mutation. In the remaining 25%, if Mom (and Kid1) or Dad (and Kid2) has a transposon, then the offspring undergo the transposon growth procedure (substituting the right-most element that mismatches the definition of the transposon, see Figure 9). This reduces the procedures to a minimum and is substantially simpler than the FSA implementation. (Specifically, procedures don't have to discriminate between mature and immature transposons, or identify NOPs.)

4 Results and Discussion

The test trail used in this work is illustrated in Figure 12. It can be seen that up to the 64th cell our trail coincides with the John Muir Trail (Figure 2), but afterwards includes chaotically scattered elements of high difficulty. Being trained on the much simpler early trail portion, the ant is not prepared to surmount the subsequent, complicated sector. More specifically, problems arise at attempts to get over gaps between the 64th and 65th, or the 67th and 68th cells.

4.1 Transposons Accelerate the Evolutionary Search

The accelerating effect of transposons is especially noticeable in small populations, where the probability of finding an effective navigation algorithm by applying standard crossover and mutation operators is low. The following experiments were therefore carried out on populations of 100 ants. The choice of such a small population also helps in carrying out a

comprehensive analysis of transposon dynamics. This would not be feasible for large populations of ants because of the greater number of transposons. For example, thousands of transposons appear in a population of only 100 ants over 1000 generations.

Besides limiting the ant population abundance, the following parameters were fixed in all test and control runs: the maximum number of an ant's steps (330) and the percentage of the population permitted to reproduce (15%). A *truncation strategy* of reproduction was used, where copies of chromosomes with scores exceeding the average value replaced all chromosomes having scores below the average. An analysis of preliminary runs showed that the results were not sensitive to variation in these three parameters.

To investigate the efficiency of the MGE technique, we performed 100 independent runs of the program, for 5000 generations each (Figure 12). The percentage of the ant population that is subjected to the action of the two-place MGE operator in a generation was 50. The results of the test and control runs (populations with transposons and without transposons respectively) were compared in several series with different values of the standard mutation parameters. The MGE technique increases the probability of finding an effective navigation algorithm for small populations and low numbers of generations. (In this section, we specify that an effective navigation algorithm should exceed a score of 64 within 330 time steps.) Figure 13 indicates that the mean and the best-of-generation score in both experiment and in control grew roughly linearly in time, but the slope of the growth with transposons was substantially higher than in the control.

To test whether MGE operators raise ant variability primarily in a nonspecific manner, by supplementing the mutation effect of standard operators, we did two experimental runs increasing the standard mutation frequencies by factors of two and ten (from the values in the control series, Figure 13). The high standard mutation rates did not raise the efficiency of the navigation algorithm, in fact we saw decreased efficiency. In addition, if MGE operators were disabled, there was no pronounced effect on evolutionary search efficiency from any mutation value or crossover parameter combination. This indicates that the increased efficiency seen in Figure 13 is due to the action of the transposon operators.

4.1.1 The SSC-ant Results—Computational experiments with the SSC-ant implementation (Figure 14) reveal behavior that is qualitatively similar to the FSA-ant (above): the transposon growth operator (the only operator on SSC transposons) produces a statistically significant increase in the efficacy of the evolutionary search compared with the control runs (operator off).

The striking difference between these results and the FSA-ant results (Figure 13) is that the SSC-ant populations prematurely converge very fast and become stagnant. We believe this a sign that more operators (as is the case for FSA-ant) are required to avoid premature convergence,

4.2 The time complexity of the MGE algorithms

Here we analyze the efficacy of the MGE approach compared to standard GA, as a function of trail length. Tests were performed with both FSA and SSC ants. The test trail was

designed specifically for this study (Figure S1). The trail is composed of elements of high difficulty (difficulty elements are the gaps between cells 7-8, 13-14, 20-21, 26-27, 33-34, 40-41, 46-47, 52-53, 59-60, 65-66, 72-73, 78-79, & 84-85). While both ant implementations behave comparably, in general, some details are rather different. FSA ants tend to show a nearly exponential dependence of efficacy (number of average evaluated cases) on trail length over a significant range (trail length from 10th to 64th cell, Figure N1A). At large trail length, the MGE approach is more efficient (less evaluations) than standard GA. Short trails, up to the end-cell #40, were easy for both techniques. Success rates are comparable between FSA and standard GA, beginning to fall steadily above trail lengths of 40. With ten times smaller populations (= 200), results are qualitatively the same (results not shown). Hence in the input limits, where the search became difficult for both techniques (the success rate is less than 100%, Figure 15), the MGE-approach acts more effectively.

Similar to FSA, SSC ants were more efficient than standard GA. SSC showed several differences with FSA. SSC showed a non-smooth dependence of efficacy on trail length, proceeding in discrete jumps rather than a smooth exponential. The jumps correspond to difficult elements in the trail (between cells 52-53, 59-60, and 65-66). While SSC was less efficient than FSA for long trails (took more evaluations), SSC had very high success, with no apparent dependence on trail length (up to length 71; by comparison, success abruptly dropped for standard GA at around length 55, and was nearly zero by length 60; standard GA is barely capable of overcoming the difficult element between cells 59 and 60). SSC-ants were also robust to changes in population size (results (not shown) were qualitatively similar for computations with 5 and 25 times larger populations). Our main conclusion is that the MGE-technique in the case of the SSC implementation does not decrease the search-time drastically, but mainly helps to prevent stagnation at local optima. Hence, both ant's implementations (FSA and SSC) show that the MGE-approach makes evolutionary search more effective.

4.3 Analysis of Transposon Population Dynamics

Transposon population dynamics were studied for a host population of 100 ants, with experiments limited to 500 generations. These limitations did not affect the overall results and conclusions of the experiments, but they allowed for graphical representation of the results, since the number of new abundant (more than 10 individuals) transposon forms in every run did not exceed 200. Every generation of the population produces new transposon offspring, typically tens of new abundant mutant forms. In the first 500 generations this gave on the order of 100 - 150 new forms.

We compared the dynamics of 25 successful ant populations (found an effective navigation algorithm) with the 25 unsuccessful ant populations (Figure 16). We found:

1. As a rule, only one form of transposon dominates in a population at any given moment; tens of new forms appear in every generation and disappear within, at most, several generations.
2. From time to time, the current dominant form is displaced by another one. Over a large number of generations, we observed several cases where a formerly dominant form reestablished itself.

3. Typically, 2-3 dominant forms replace each other in a population over a 500 generation computation. (Less often, we saw 5-6 dominant forms, and rarely, up to 10.)
4. From time to time, an explosion in the abundance of subdominant forms occurs, lasting 5-10 generations. Overall, these subdominant forms can give 2-3 population explosions. These explosions can lead to the conversion of a subdominant form into a dominant one.

We observed these dynamics in both stagnant and in highly-evolvable successful ant populations. Inspection of the path-learning dynamics showed apparent “coincidences” of the moment of change in dominant transposon form with the moment of finding the first effective navigation algorithm (Figure 16). Just after these switches, the mean population score rises over several generations to the next plateau. It remains to be proven whether this apparent correlation stems from a functional relationship.

4.4 Horizontal Transmission of Transposons Is Necessary for Their Effective Mutation Effect

Because transposons are transmitted vertically, that is, from ancestors to descendants, transposons belonging to a host with a reproductive advantage would rapidly spread through a population and develop new forms in the process. But this process is insufficient *per se* for effective acceleration of ant learning. The two-place MGE operator, performing horizontal distribution of transposons from one ant to another is necessary for an increase of ant training ability. Here we compared the test presented in Figure 13 with a similar test in which the frequency of applying the two-place MGE operator was reduced by a factor of ten, to 5%. (In all previous experiments presented in this article, this percentage was 50%.) The decrease in ant learning ability with decrease in frequency of two-place operator application was shown in Figure 8 of [74]. This frequency lowers the search efficacy nearly to the level of the control (lacking transposons).

4.5 Intelligent Mutators Have a Search Space Confining Effect

The Muir’s Trail search space has a rugged geometry due to the specific and discrete characteristics of the problem. That is why gradient methods are not effective here [32]. Moreover, this ant navigation problem is classified as a GA hard problem [32], especially if the trail is not designed specifically for ant population training. The efficiency of transposons in the role of intelligent mutators can be measured by their search space domain confining ability. The selection criteria in the MGE operators increases the probability of finding effective navigation algorithms for elements (trails) of high complexity. We propose that quasi-periodic state successions meeting the transposon definition will be picked up by selection and used as a component of a universal navigation algorithm.

Comparing mutation frequencies between experiment (transposons) and control (no transposons) confirms that transposons reduce the number of evaluations to reach a comparable learning rate. The mutation frequencies for control runs (Figure 13) have a crossover rate + mutation rate = 0.0001+0.04 P/bit/generation; MGE1 and MGE2 operators add on average 0.0027 and 0.0075 P/bit/generation respectively. In other words, transposons

on average add 0.012 P/bit/generation to the control value of 0.041. This addition accelerates ant population learning. For example, on Figure 13 at the end of the experiment (4622nd time-step), the control has a max score of 6.47, whereas the test set already attained this value by the 451st time-step, some 10 times faster.

Furthermore, though the ant test is now accepted as a classical one, its implementation in the form of an FSA is far from optimum. Firstly, the 32 states are obviously too many for the classical trails. This redundancy brings an enormous extension of search space dimensionality. Secondly, generating action sequences for the situations when the ant is faced with a black cell is redundant. “Step forward” is clearly the only adequate action here. Finally, the NOP action is useless.

The simplest and most effective method for this type of navigation problem appears to be to use symbol strings, composed of combinations of three possible actions (FWD, RGT, and LFT) for the situation when the ant faces a white cell. Applying the MGE approach to the ant problem, we performed this kind of search space size reduction. In so doing, we deal only with the “white half” of the search space which is formed by all sequences of actions, executed while the ant has white cells in front of it. Furthermore, we escape internal loops, and exclude the NOP action. We believe this mechanism explains the MGE approach effect, i.e. the up to ten-fold increase in evolutionary search rate. The additional contributing factor for the MGE technique’s vast increase in efficiency is the inclusion of the transposition procedure (with application frequency substantially higher than that of the other mutation operators).

4.6 Dominant Transposon Forms Are the Components of Effective Navigation Algorithms

The problem of programming an artificial ant to follow the Santa Fe trail has been repeatedly used as a benchmark problem in GP [15, 18, 22, 27, 28, 34, 39]. Particularly Langdon and Poli have shown that the performance of several techniques is not much better than the best performance obtainable using uniform random search [32]. According to these authors, the search space is large and forms a Karst landscape containing many false peaks and many plateaus separated by deep valleys. While such a fitness landscape is difficult for hill climbers and for GA [23], it is likely realistic for many optimization problem spaces and needs study.

What possible evolutionary processes can lead to the emergence of a more general algorithm for artificial ant pathfinding on trails of high difficulty and arbitrary organisation? The initial section of the trail in Figure 12 is organised as follows: a turn to the right, a turn to the left, and a gap two cells in length. A population of ants can develop an effective algorithm for traversing the first half of the trail within a period of several generations, with rules: if there is no trail ahead, the ant turns to the right. If the trail is still not found, the ant makes two more turns to the right. If after this it still does not find the trail, the ant takes one step forward and “sniffs around” the adjacent cells. If it still does not find the trail, the ant takes one more step forward in the direction of the interrupted path and repeats the previous procedure. This is the RLLRF (or LRRLF) loop (See section 3.4.1).

The ant may, however, encounter situations of a higher order of difficulty. For instance, the next cell of the trail may be four to five cells away diagonally from the last, as is the case of the gap between cells #64 and #65 on our trail in Figure 12. This becomes a “bottleneck” for the evolutionary process. It is clear that a population can learn step-by-step on a trail of gradually increasing complexity. But an approach where the population of ants generates a more general pathfinding algorithm on the basis of particular rules developed earlier (namely, RLLRF or LRRLF loops), is of much greater interest for us. This “intellectual” work is a result of active internal processes in an ant's genotype that lead to navigation rules spontaneously being made redundant.

In the case of an encounter with a trail element of high difficulty that the population was not trained to handle in an earlier section of the trail, the following course of events may occur. As the ant population is unable to traverse the “bottleneck”, no ant in the population will be able to amass a higher score than others, and as such, no ant will have any evolutionary advantage over another. Thus, there will be no selection pressure in the population to overcome the obstacle. However, the internal population of transposons have their own selection pressure - i.e. to increase in size. This selection pressure will result in the simple algorithms mentioned earlier (namely, RLLRF and LRRLF loops) being replaced by an increasing number of bulkier programs (see Table S1 – S3) through the mechanisms of transposon growth and transposon mutagenesis (See sections 3.4.2 and 3.4.3).

These bulkier programs are able to solve the initial navigation problem. However, they do this with obvious redundancy. Specifically, new programs appear when the ant is sniffing around the surroundings not directly ahead in the direction of the lost trail, but moving zigzag in the same direction (Figure 17A). Redundant programs like this can be applied to more complex navigation. If the population is still in a state of stagnancy, then eventually navigation programs will appear with high redundancy due to recursion of the bulkier algorithm. This can happen because of the periodicity of transposon code. Some of these redundant programs, for example, will provide instructions for sniffing around the surroundings in the direction of the lost trail, but by a closed contour, which will catch a continuation of the trail five cells to the right and five cells up (Figure 17B). What is remarkable is that FSA ants tend to evolve four-segment closed paths, while SSC-ants tend to develop two-segment closed paths (Tables S1-S3 and Figure 17A,B)

As a result, the evolutionary process will develop a more general algorithm of navigation, as is the case of the examples in Tables S1-S3. It is important that these algorithms provide instructions for searching the vicinity of the last cell on the discovered trail. Ants of this type are able to pass through trails with chaotically scattered difficult elements.

Detailed analysis of the organization of dominant transposon forms in ant populations that succeeded in finding an effective navigation program showed that the transposons themselves became components of the programs. This is especially so for the portion of the navigation program used for effective “sniffing around” in situations where the ant is faced with a wide gap. In other words, we have determined that the transposons in our experiments take an active role in the propagation of their hosts’ building blocks.

5 Future Directions

For future work we aim to test symbolic strings representations on more sophisticated benchmark tests and beginning to apply the approach to real-life problems. We foresee several nearest extensions. First, are problems in the field of animats (which are more sophisticated than ants), particularly FSA animats. This could have prospects for real-life automatic navigation. Second, is in the area of in silico evolution (evolution on the computer) of gene regulatory networks, GRNs (see our publications [76, 83] and our review in this area [67]). A third direction would be extension of the MGE approach itself.

In the area of FSA animats, future work will aim at improvement and generalization of both animats and the MGE-algorithms. This will involve extending our approach to other FSA-animat benchmark tests and to other FSA-implemented problems. Classical animats for the ant-tracker problem have only one, frontal sensor, and don't need a large state memory (the only thing that a well-trained ant must remember is whether or not it is on the path). Using more than one frontal sensor represents a substantial modification of the ant-tracker problem on the way to more realistic navigating animats. We plan to use the FSA approach to design multisensor animats, proposed by Ashlock and Joenks [4] and Ashlock and Freeman [3]. The agent senses its environment, consisting of the nearest eight cells, through 8 sensors. It is promising that Ashlock and Freeman [3] used an FSA implementation of such a multisensor animat and achieved surprisingly good results for the Tartarus problem.

Two problems for virtual robots, the Tartarus and the Bulldozer [3 - 6, 77, 78], are similar to the ant-pathfinder problem and are good benchmarks for evolving agents that need state memory. We will extend our work on these problems in the FSA-implementation. The Ashlock and Freeman's [3] FSA implementation of the 8-sensor animat makes it possible to apply with some modifications the artificial transposons and MGE-algorithms introduced in this paper. Though these two robotic problems are qualitatively more complicated than the ant-animat problem, our transposon approach has an analogue in Tartarus. Namely, Ashlock and Joenks [4] have proposed a way to control an agent's behavior by a variable length sequence of actions, which is repeated over and over until it runs out of moves. In this manner, an agent executes a dance, similar to the 'sniffing' strategy of an ant.

For GRN in silico evolution and evolutionary gene design by computer, we plan to apply the symbolic string representation of a gene [83]. This perfectly suits our transposon technique. (In fact, it is very similar to the SSC ant considered in this publication.) Symbolic string representation of gene regulatory regions provides a natural way to implement (artificial) transposons, using the techniques implemented in this paper. In essence, short biological words of the DNA alphabet (A, T, G, C), known as the transcription factor binding sites, TFBS (often highly specific sequences) are encoded by unique symbol(s) from another alphabet (the alphabet for TFBS; say, the first letter of the factor name). Transcription factors bind specifically to the binding sites, activating or repressing the target gene expression. TFBS tend to cluster in cis-regulatory modules (CRM). In our approach, each CRM is a sequence of the symbols (a word from the TFBS alphabet). In this representation, mutations are changes, additions and eliminations of the symbols (TFBS) in the CRM sequences. Transposons can be short periodic sequences with some fixed and some variable

elements. Our preliminary tests have confirmed the applicability of the approach to systems biology problems.

We will also extend our work to the generalization of the MGE algorithms. In this first attempt to apply new biologically inspired ideas, we exploited only one of many possible ways to implement MGE into a GA-ant pathfinder. In further work, we plan to implement more complicated rules for MGE-mutagenesis: not only based on the artificial transposon periodicity, but also, more generally, on short schema, with freedom to mutate some predefined elements and protected others. For example, with GRNs it could be schemata like $CD^{**}AB^{**}EF^{**}$, where symbols A – F are fixed, and, say, A&B stand for factors activating the target gene, and C – F are for repressors, while ‘*’ are free to mutate. This transposon sequence is periodic in a sense of obligatory pairs of repressor and activator binding sites. A next natural step would be to give the approach freedom to choose between several sets of rules (possibly on the fly): with, say, different MGE periods, or sets of similar schemata to choose from.

6 Conclusions

Our approach, inspired by biological mobile genetic elements, or MGEs (transposons, “jumping genes”), uses transposon sequence definitions to find transposons in the host genome. Transposon mutagenic action proceeds by MGE-operators changing chromosome elements adjacent to the defined transposon sequence (extending the transposon sequence by so doing). We found that:

- Our approach makes evolutionary searches more effective, using the classic benchmark test of the ant-pathfinder (we tested both FSA and our symbol-string chromosome, SSC, on several versions of trails). Both ant’s implementations (FSA and SSC) show that the MGE-approach makes evolutionary search less prone to premature convergence.

The symbolic string implementation of the ANT navigation program (implicitly for the FSA-ants and explicitly for the SSC-ants) is more similar to the biological prototype than implementations with, say, real-number chromosomes. Symbolic strings are a more natural implementation for studying artificial transposons from a biological perspective, as the population dynamics of genomic “parasites”. The population analysis reveals that:

- As a rule, only one form of transposon dominates in a population at any given moment; many new forms appear in each generation, generally lasting no more than several generations;
- From time to time, the current dominant form is displaced by another one;
- From time to time, an explosion in the abundance of subdominant forms occurs, which can last for several generations. Such an explosion can result in the conversion of a subdominant form into a dominant one;
- These dynamic observations hold for stagnant and for highly-evolvable ant populations which successfully find effective navigation algorithms. The timing of

finding the algorithm typically corresponds with a change in dominant transposon form;

- Detailed analysis of the dominant transposon forms in ant populations that were successful in finding effective navigation programs showed that the transposons themselves become components of the programs.

Supplementary Material

Refer to Web version on PubMed Central for supplementary material.

Acknowledgments

We thank anonymous reviewers for their constructive comments on the manuscript. AVS and DMH have been supported in part by Joint NSF/NIGMS BioMath Program, 1-R01-GM072022, the National Institutes of Health, 2R56GM072022. AVS was also supported in part by the Russian Foundation for Basic Research grant 13-04-02137. VFL has been supported by CRDF Solutions Project RUB1-33054-ST-12. JJM has been supported in part by project ANYSELF TIN2011-28627-C04-02 and Muses <http://musesproject.eu>.

REFERENCES

1. Agrawal A, Eastman QM, Schatz DG. Transposition mediated by RAG1 and RAG2 and its implications for the evolution of the immune system. *Nature*. 1998; 394:744–751. [PubMed: 9723614]
2. Altenberg, L. The evolution of evolvability in genetic programming. In: Kinnear, KE., editor. *Advances in Genetic Programming*. MIT Press; Cambridge: 1994. p. 47-74.
3. Ashlock, D.; Freeman, J. A Pure Finite State Baseline for Tartarus. *Proceedings of the 2000 Congress on Evolutionary Computation*; 2000. p. 1223-1230.
4. Ashlock, D.; Joenks, M. ISAc Lists: A Different Program Induction Method, *Genetic Programming 1998. Proceedings of the Second Annual Conference on Genetic Programming*; 1998. p. 18-26.
5. Ashlock, D.; McRoberts, M. ISU Mathematics Technical Report. 1997. A GP-Automata reprise of Astro Teller's bulldozer experiment; p. AM97-17.
6. Ashlock, D.; Willson, S.; Leahy, N. Coevolution and Tartarus. *Proc. CEC. Congress on Evolutionary Computation*; Portland, Oregon. IEEE Press: 2004. p. 1618-1624.
7. Banzhaf W, Beslon G, Christensen S, Foster JA, Kps F, Lefort V, Miller JF, Radman M, Ramsden JJ. From artificial evolution to computational evolution: A research agenda. *Nature Reviews Genetics*. 2006; 7:729–735.
8. Bolis, E.; Zerbi, C.; Collet, P.; Louchet, J.; Lutton, E. A GP Artificial Ant for Image Processing: Preliminary Experiments with EASEA. In: Gould, JL.; Keeton, WT., editors. *Biological Science*. W. W. Norton & Company; 1996. p. 246-255. EuroGP 2001
9. Brennan, PM. ANT: Simulated Evolution on a PC, in *Carnegie-Mellon University Artificial Intelligence Repository*. 1994. <ftp://ftp.cs.cmu.edu/user/ai/areas/genetic/ga/systems/ant/>
10. Brosius J. Retroposons - Seeds of evolution. *Science*. 1991; 251:753. [PubMed: 1990437]
11. Burke DS, De Jong KA, Grefenstette JJ, Ramsey CL, Wu AS. Putting more genetics into genetic algorithms. *Evolutionary Computation*. 1998; 6:387–410. [PubMed: 10030470]
12. Chan TM, Man KF, Kwong S, Tang KS. A Jumping Gene Paradigm for Evolutionary Multiobjective Optimization' *Evolutionary Computation*. *IEEE Transactions on*. 2008; 12(2):143–159.
13. Chang FC, Huang HC. A refactoring method for cache-efficient swarm intelligence algorithms. *Information Sciences*. 2012; 192(1):39–49.
14. Chen, YH.; Huang, HC. Neural Computing and Applications. Coevolutionary genetic watermarking for owner identification. in press (DOI: 10.1007/s00521-014-1615-z)
15. Chellapilla, K. Evolutionary programming with tree mutations: Evolving computer programs without crossover. In: Koza, John R.; Deb, Kalyanmoy; Dorigo, Marco; Fogel, David B.; Garzon,

- Max; Iba, Hitoshi; Riolo, Rick L., editors. Genetic Programming 1997: Proceedings of the Second Annual Conference; Stanford University, CA, USA. Morgan Kaufmann; Jul 13-16. 1997 1997 p. 431-438.
16. Cui F, Sirotni MV, Zhurkin VB. Impact of Alu repeats on the evolution of human p53 binding sites. *Biol Direct*. 2011; 6:2. [PubMed: 21208455]
  17. Fadool JM, Hartl DL, Dowling JE. Transposition of the mariner element from *Drosophila mauritiana* in zebrafish. *Proc. Natl. Acad. Sci. USA*. 1998; 95:5182–5186. [PubMed: 9560250]
  18. Harries, K.; Smith, P. Exploring alternative operators and search strategies in genetic programming. In: Koza, John R.; Deb, Kalyanmoy; Dorigo, Marco; Fogel, David B.; Garzon, Max; Iba, Hitoshi; Riolo, Rick L., editors. Genetic Programming 1997: Proceedings of the Second Annual Conference; Stanford University, CA, USA. Morgan Kaufmann; Jul 13-16. 1997 1997 p. 147-155.
  19. Harvey, I. The Microbial Genetic Algorithm. In: Kamps, G.; Karsai, I.; Szathmáry, E., editors. *Advances in Artificial Life: ECAL 2009*. Springer, Heidelberg; 2011. p. 126-133.p. 978-3642212826.Part II, LNCS 5778ISBN-13
  20. Hillis WD. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D*. 1990; 42:228–234.
  21. Hurst GDD, Werren JH. The role of selfish genetic elements in eukaryotic evolution. *Nature Reviews Genetics*. 2001; 2:597–606.
  22. Ito, T.; Iba, I.; Sato, S. Non-destructive depth-dependent crossover for genetic programming. In: Banzhaf, W.; Poli, R.; Schoenauer, M.; Fogarty, TC., editors. *Proceedings of the First European Workshop on Genetic Programming*; LNCS, Paris. Springer-Verlag; Apr 14-15. 1998 1998
  23. Jefferson, D.; Collins, R.; Cooper, C.; Dyer, M.; Flowers, M.; Korf, R.; Taylor, C.; Wang, A. Evolution as a Theme in Artificial Life: The Genesys/Tracker System. In: *Artificial Life II*. In: Langton, CG.; Taylor, C.; Farmer, JD.; Rasmussen, S., editors. *SFI Studies in the Sciences of Complexity*. Vol. X. Addison-Wesley; 1991.
  24. De Jong, KA.; Potter, MA. Evolving complex structures via cooperative coevolution. *Forth Annual Conference on Evolutionary Computation*; San Diego, CA. Mar 1-3. 1995 1995
  25. De Jong, ED.; Bucci, A. DECA: Dimension extracting coevolutionary algorithm. *Proceedings of the 8th annual conference on Genetic and evolutionary computation*; 2006. p. 313-320.ACM
  26. King CC. Modular transposition and the dynamical structure of eukaryote regulatory evolution. *Genetica*. 1992; 86:127–42. [PubMed: 1334905]
  27. Koza, JR. *Genetic Programming: on the Programming of Computers by Means of Natural Selection*. MIT Press; Cambridge, Mass: 1992.
  28. Kuscu, I. Evolving a generalised behavior: Artificial ant problem revisited. In: William Porto, V., editor. *Seventh Annual Conference on Evolutionary Programming*; San Diego, California, USA. Mission Valley Marriott; Springer-Verlag; Mar 25-27. 1998 1998
  29. Kuyucu, T.; Tanev, I.; Shimohara, K. Incremental Genetic Programming Via Genetic Transpositions For Efficient Coevolution Of Locomotion And Sensing Of Simulated Snake-Like Robot. *Proceedings of the 20th European Conference on Artificial Life (ECAL-2011)*; Paris, France. MIT Press; Aug 8th - 12th. 2011 2011 p. 439-446.
  30. Kuyucu, T.; Tanev, I.; Shimohara, K. Memetic Computing. 2012. Incremental Evolution of Fast Moving and Sensing Simulated Snake-like Robot with Multiobjective GP and Strongly-typed Crossover. DOI: 10.1007/s12293-012-0085-z
  31. Tanev, I.; Kuyucu, T.; Shimohara, K. *Genetic Programming and Evolvable Machines*. Springer; 2013. GP-induced and explicit bloating of seeds in incremental GP improves evolutionary success. DOI: 10.1007/s10710-013-9192-y
  32. Langdon, WB.; Poli, R. Why ants are hard. In: Koza, John R.; Banzhaf, Wolfgang; Chellapilla, Kumar; Deb, Kalyanmoy; Dorigo, Marco; Fogel, David B.; Garzon, Max H.; Goldberg, David E.; Iba, Hitoshi; Riolo, Rick, editors. *Genetic Programming 1998: Proceedings of the Third Annual Conference*; Madison, Wisconsin, USA. Jul 22-25. 1998 1998 University of Wisconsin
  33. Lee, C-Y.; Antonsson, EK. Adaptive Evolvability via Non-Coding Segment Induced Linkage. In: Spector, Lee; Goodman, Erik D.; Wu, Annie; Langdon, WB.; Voigt, Hans-Michael; Gen, Mitsuo; Sen, Sandip; Dorigo, Marco; Pezeshk, Shahram; Garzon, Max H.; Burke, Edmund, editors.

- Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001); San Francisco, California, USA. 7-11 July 2001; 2001. p. 448-453.
34. Lee, JYB.; Wong, PC. The effect of function noise on GP efficiency. In: Yao, X., editor. Progress in Evolutionary Computation, Lecture Notes in Artificial Intelligence. Vol. 956. 1995. p. 1-16. Springer-Verlag, Heidelberg, Germany
  35. Li, K.; Kwong, S.; Man, KF. JGBL paradigm: a novel strategy to enhance the exploration ability of NSGA-II. In: Krasnogor, N., editor. GECCO'11: Proc. 12th Annual Conference on Genetic and Evolutionary Computation; ACM Press; 2011. p. 99-100.
  36. Li K, Kwong S, Wang R, Tang KS, Man KF. Learning paradigm based on jumping genes: A general framework for enhancing exploration in evolutionary multiobjective optimization. *Inf. Sci.* 2013; 226:1–22.
  37. Lones, MA.; Tyrrell, AM. Enzyme genetic programming. Proceedings of the 2001 Congress on Evolutionary Computation, CEC 2001; 2001. p. 1183-1190. IEEE Press
  38. Lozovskaya ER, Hartl DL, Petrov DA. Genomic regulation of transposable elements in *Drosophila*. *Curr Opin genet Dev.* 1995; 5:768–73. [PubMed: 8745075]
  39. Luke, S.; Spector, L. A comparison of crossover and mutation in genetic programming. In: Koza, John R.; Deb, Kalyanmoy; Dorigo, Marco; Fogel, David B.; Garzon, Max; Iba, Hitoshi; Riolo, Rick L., editors. Genetic Programming 1997: Proceedings of the Second Annual Conference. Morgan Kaufmann; Stanford University, CA, USA: Jul 13-16. 1997 1997 p. 240-248.
  40. Luke, S.; Hamahashi, S.; Kitano, H. "Genetic" programming. In: Banzhaf, W., editor. GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference; San Francisco: Morgan Kaufmann; 1999.
  41. Makalowski, W. SINEs as a Genomic Scrap Yard. In: Maraia, Richard J., editor. Chap. 5 in The Impact of Short Interspersed Elements (SINEs) on the Host Genome. R.G. Landes Company; Austin: 1995. 1995
  42. Makalowski W. Genomic scrap yard: how genomes utilize all that junk. *Gene.* 2000; 259:61–67. [PubMed: 11163962]
  43. Masum, H.; Oppacher, F. Regulatory networks and genomic algorithms. In: Callaos, Nagib; Nunes da Silva, Ivan; Molero, Jorge, editors. Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics; 2001. 2001 (SCI 2001/ISAS 2001) volume III. IIIS
  44. McClintock B. The significance of responses of the genome to challenge. *Science.* 1984; 226:792–801. [PubMed: 15739260]
  45. Nawa NE, Furuhashi T, Hashiyama T, Uchikawa Y. A study on the discovery of relevant fuzzy rules using pseudo-bacterial genetic algorithms. *IEEE Transactions on Industrial Electronics.* 1999; 7(5):608–616.
  46. Olsson, B. Optimization using a host-parasite model with variable-size distributed populations. Proceedings of the 1996 IEEE 3rd International Conference on Evolutionary Computation; 1996. p. 295-299. IEEE Press
  47. Olsson, B. Parallel Problem Solving from Nature PPSN VI, Lecture Notes in Computer Science Volume 1917. 2000. NK-landscapes as Test Functions for Evaluation of Host-Parasite Algorithms; p. 487-496.
  48. Olsson, B. Co-evolutionary search in asymmetric spaces. In: Wang, PP., editor. Proceedings of the Fifth Joint Conference on Information Sciences; Association for Intelligent Machinery; 2001. p. 1040-1043.
  49. Pereira, FB.; Machado, P.; Costa, E.; Cardoso, A.; Ochoa-Rodriguez, A.; Santana, R.; Soto, M. Too Busy to Learn. Proceedings of the 2000 Congress on Evolutionary Computation (CEC' 2000); San Diego, USA: 2000. p. 720-727. 16-19 July, 2000
  50. Polak P, Domany E. Alu elements contain many binding sites for transcription factors and may play a role in regulation of developmental processes. *BMC Genomics.* 2006; 7:133. [PubMed: 16740159]
  51. Potter, MA.; De Jong, KA. A Cooperative Coevolutionary Approach to Function Optimization. In: Davidor, Y.; Schwefel, H-P.; Manner, R., editors. Parallel Problem Solving from Nature (PPSN III); Jerusalem, Israel. 1994. p. 249-257.

52. Potter, MA.; De Jong, KA. Evolving neural networks with collaborative species. Proc. of the 1995 Summer Computer Simulation Conference; Ottawa, Ontario, Canada. 1995. 24-26 July 1995
53. Potter, MA.; De Jong, KA.; Grefenstette, JJ. A coevolutionary approach to learning sequential decision rules. In: Forrest, Stephanie, editor. Proceedings of the Sixth International Conference on Genetic Algorithms; San Mateo, CA. July 1995; 1995. p. 366-372. Morgan Kaufman
54. Ripon KSN, Kwong S, Man KF. A real-coding jumping gene genetic algorithm (RJGGA) for multiobjective optimization. *Inf. Sci.* 2007; 177(2):632–654.
55. Ripon, KSN.; Tsang, CH.; Kwong, S. Evolutionary Scheduling:. 2007. An evolutionary approach for solving the multi-objective job-shop scheduling problem; p. 165-195.
56. Shapiro JA. Transposable elements as the key to a 21st century view of evolution. *Genetica.* 1999; 107:171–179. [PubMed: 10952210]
57. Shapiro JA. Repetitive DNA, genome system architecture and genome reorganization. *Res Microbiol.* 2002; 153:447–53. [PubMed: 12405352]
58. Simões, A.; Costa, E. Transposition: A Biologically Inspired Mechanism to Use with Genetic Algorithms. In: Dobnikar, A.; Steele, N.; Pearson, D.; Albrecht, R., editors. Proceedings of the Fourth International Conference on Neural Networks and Genetic Algorithms (ICANNGA 99); Portoroz, Slovenia. 6-9 April, 1999; Springer-Verlag; 1999. p. 178-186.
59. Simões, A.; Costa, E. Transposition: A Biologically Inspired Mechanism to Use with Genetic Algorithms. Proceedings of the Fourth International Conference on Neural Networks and Genetic Algorithms (ICANNGA'99); 1999. p. 612-619. Springer-Verlag
60. Simões, A.; Costa, E. Transposition versus Crossover: An Empirical Study. In: Banzhaf, W.; Daida, J.; Eiben, AE.; Garzon, MH.; Honavar, V.; Jakiela, M.; Smith, RE., editors. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99); 1999. p. 612-619. Orlando, Florida USA, CA: Morgan Kaufmann
61. Simões, A.; Costa, E. Enhancing Transposition Performance. Proceedings of the 1999 Congress on Evolutionary Computation (CEC'99); Piscataway, NJ: IEEE Press; 1999. p. 1434-1441.
62. Simões, A.; Costa, E. Using Genetic Algorithms with Asexual Transposition. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2000); Las Vegas. 2000. 8-12 July
63. Simões, A.; Costa, E. Using Genetic Algorithms with Sexual or Asexual Transposition: a Comparative Study. Proceedings of the Congress on Evolutionary Computation 2000 (CEC'2000); Piscataway, New Jersey. 2000. p. 1196-1203. v 2 July 2000. IEEE Service Center
64. Spirov AV. Self-Assemblage of gene Networks in Evolution via Recruiting of New Netters. Parallel Problem Solving from Nature (PPSN IV), International Conference on Evolutionary Computation. The 4th International Conference on Parallel Problem Solving from Nature, Berlin, Germany, September 22-26, 1996, Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberger, and Hans-Paul Schwefel (Eds), Lecture Notes in Computer Sciences. 1996; 1141:91–100.
65. Spirov, AV. Self-organisation of gene networks in evolution via recruiting of new netters. Proceedings of the First International Conference on Evolutionary Computations and Its Applications; Moscow, Russia. 1996. p. 399-405.
66. Spirov AV, Holloway DM. Retroviral Genetic Algorithms - Implementation with Tags and Validation against Benchmark Functions. *IJCCI (ECTA-FCTA).* 2011; 2011:233–238.
67. Spirov AV, Holloway DM. Using evolutionary computations to understand the design and evolution of gene and cell regulatory networks. *Methods.* 2013; 62(1):39–55. 2013 Jul 15. [PubMed: 23726941]
68. Spirov, AV.; Kadyrov, AS. In: High-Performance Computing and Networking. 1998. Transposon Element Technique Applied to GA-based John Muir's Trail Test; p. 925-928.
69. Spirov, AV.; Kazansky, AB.; Kadyrov, AS. Utilizing of "Parasitic" Mobile Genetic Elements in Genetic Algorithms. *Int. Conf. on Soft Computing and Measurements; St.-Petersburg, Russia.* June; 1998. p. 266-269.
70. Spirov, AV.; Kazansky, AB. Evolutionary Biology and Evolutionary Computations: Parasitic Mobile Genetic Elements in Artificial Evolution. 2nd Int. Conf. on Soft Computing and Measurements, SCM99; St.-Petersburg, Russia. June; 1999.

71. Spirov, AV.; Kazansky, AB. The usage of artificial transposons for the tests with royal road functions. 5th Int. Conf. on Soft Computing and Measurements, SCM2002; St.Petersburg, Russia. June; 2002.
72. Spirov, AV.; Kazansky, AB. Jumping genes-mutators can raise efficacy of evolutionary search. In: Langdon, WB.; Cantu-Paz, E.; Mathias, K., editors. Proc. Genetic and Evolutionary Computation Conference, GECCO2002; New York. July 9-13; San Francisco: Morgan Kaufmann Publishers; 2002. p. 561-568.
73. Spirov, AV.; Kazansky, AB. The usage of artificial transposons for the protection of already found building blocks: the tests with royal road functions. Proc. The 6th World Multiconference on Systemics, Cybernetics and Informatics, SCI2002; Orlando, Florida. July 14-18; 2002. V. IV, pp 75-80, Nagib Callaos, Alexander Pisarchik and Mitsuyoshi Ueda, eds., Int. Inst. Informatics and Systemics
74. Spirov, AV.; Kazansky, AB.; Zamdborg, L.; Merelo Guervós, JJ.; Levchenko, VF. Forced Evolution in Silico by Artificial Transposons and their Genetic Operators: The John Muir Ant Problem. CoRR abs/0910.5542. 2009.
75. Spirov, AV.; Samsonova, MG. Strategy of Co-evolution of Transposons and Host Genome: Application to Evolutionary Computations. In: Proceedings of the Third Nordic Workshop on Genetic Algorithms and their Applications (3NWGA); 20 - 22 August 1997; 1997. p. 71-82.Helsinki, Finland, Ed. Jarmo T. Alander, Finnish Artificial Intelligence Society
76. Spirov AV, Zagriychuk EA, Holloway DM. Evolutionary Design of Gene Networks: Forced Evolution by Genomic Parasites. Parallel Processing Letters. 2014; 24:1440004. [21 pages] DOI: 10.1142/S0129626414400040. [PubMed: 25558118]
77. Teller, A. Advances in Genetic Programming, Complex Adaptive Systems. MIT Press; 1994. The Evolution of Mental Models; p. 199-220.
78. Trenaman, A. Concurrent Genetic Programming and the Use of Explicit State to Evolve Agents in Partially-Known Environments. Genetic Programming 1998: Proceedings of the Third Annual Conference; 22-25 July 1998; Morgan Kaufmann; 1998. p. 391-398.
79. Urbano, P.; Georgiou, L. Improving Grammatical Evolution in Santa Fe Trail using Novelty Search. Proceedings of European Conference of Artificial Life (ECAL) 2013; Taormina, Italy. September; 2013. p. 917-924.MIT Press
80. Vansant G, Reynolds WF. The consensus sequence of a major Alu subfamily contains a functional retinoic acid response element. Proc. Natl. Acad. Sci. USA. 1995; 92(18):8229–8233. [PubMed: 7667273]
81. Yeung, SH.; Man, KF. International Book of Microwave Science and Technology. 2007. A New Jumping Genes Paradigm for an E-Shaped Folded Patch Feed Antenna Design. Volume 2007, Article ID 10672, 10 pages doi:10.1155/2007/
82. Zahadat P, Katebi S. Tartarus and fractal gene regulatory networks with input. Advances in Complex Systems Book (ACS). 2008; 11:803–829.
83. Zagrijchuk EA, Sabirov MA, Holloway DM, Spirov AV. In silico evolution of the hunchback gene indicates redundancy in cis-regulatory organization and spatial gene expression. J. Bioinform. Comput. Biol. 2014; 12:1441009. [20 pages] DOI: 10.1142/S0219720014410091. [PubMed: 24712536]
84. Zongker, D.; Punch, B. lil-gp 1.0. 1995. <http://garage.cse.msu.edu/software/lil-gp/index.html><http://garage.cse.msu.edu/software/lil-gp/index.html>

Highlights

Application of fresh ideas from modern evolutionary biology may improve the performance of evolutionary computations, and jumping mutators (transposons) are good candidates for this breakthrough

We develop a new approach exploiting some key aspects of co-evolution of the hosts and their genomic parasites (transposons)

We add new objects to the standard toolkit of GA - artificial transposons and a collection of operators that operate on them: the artificial transposons implemented are codes served by their own operators of reproduction, mutation and transmission

Our enhancement of Genetic Algorithms by the artificial transposons increases the efficiency of Genetic Algorithms searching

If a transposon finds a good mutation strategy, then both host and parasite will get a chance for reproduction

Figure 1.

Model of the role of transposons in the evolution of their hosts.

(A) A transposon transposes itself by making a copy from one of its locations in the host's genes to another location. It can insert itself either in regulatory elements of the host gene or in a coding region. The insert depicts schematically the usual presence of some host gene-regulatory elements in transposon sequences. Two possible scenarios of the transposon insertion into the host genes are illustrated: deleterious silencing of the host gene (B) or less severe damaging of the gene followed by evolutionary acquisition of some new functions for it (C-D).

(B) Typical consequence of the transposon insertion. This deleterious effect manifests itself in silencing the host gene (rightmost cross) either by damaging the gene-regulatory circuits (black crosses) or the product encoded by the gene.

(C) Occasionally, the new insertion is not absolutely deleterious: either a gene exists in more than one copy, or the regulatory circuits are not damaged severely.

(D) A non-lethally damaged gene (C) can, with time, acquire a few mutations, restoring the gene functionality but under new regulatory circuits. These new regulatory loops are the result of recruiting transposon regulatory elements.

Figure 2.

The John Muir Trail. Some "milestone" squares are numbered. The trail itself is a series of black squares on a 32x32 white toroidal (i.e., wraparound) grid. Each black cell is numbered sequentially, from the 1st to the 64th.

Author Manuscript

Author Manuscript

State	Input=0	Input=1
00	FWD/0A	NOP/09
01	RGT/0E	FWD/03
02	NOP/08	NOP/0E
03	FWD/13	RGT/18
04	NOP/17	NOP/0B
05	RGT/17	RGT/0A
06	FWD/04	NOP/09
07	LFT/0A	LFT/17
08	LFT/12	FWD/1E
09	RGT/1E	RGT/16
0A	RGT/16	FWD/06
0B	RGT/13	NOP/16
0C	LFT/03	LFT/0B
0D	LFT/0E	NOP/14
0E	LFT/0C	NOP/12
0F	RGT/15	FWD/1F
10	NOP/0E	LFT/17
11	FWD/12	FWD/0F
12	LFT/11	NOP/0C
13	RGT/02	NOP/1D
14	RGT/0C	LFT/0E
15	FWD/18	FWD/09
16	FWD/01	NOP/08
17	NOP/0B	LFT/1A
18	LFT/13	NOP/11
19	NOP/0D	RGT/01
1A	NOP/1E	LFT/1B
1B	FWD/03	FWD/10
1C	RGT/0A	NOP/00
1D	RGT/06	LFT/0A
1E	RGT/0C	NOP/18
1F	RGT/10	FWD/04

B

Figure 3.

The FSA implementation of the Ant navigation problem.

(A) Flow diagram for the ANT program.

(B) A sample implementation of an ant as a finite state automaton. There are 32 states in all. Each state may lead to one of two possible states, depending on the input signal. The input signal is what an ant sees before him. If the cell before him is black then the input is 1, and if white the input is 0. Furthermore, each state transition is accompanied by one of four possible movements (FORWARD - FWD, RIGHT - RGT, LEFT - LFT, or NOP). (See text for further details.)

(C) An integer string with $4 \times N$ integers (two links and two actions per state) was used to codify the information contained in the transition table (B).

(D) A graph representation of the dynamics of an example ant program, corresponding to transition table (B) in the case of a constant “1” input. Each vertex consists of a state (in circle) and an action (in rectangle).

Figure 4.

A population of ants, based on the variable-length symbol-string chromosomes (SSCs). Each such chromosome is a string of length n , $\text{Min} < n < \text{Max}$, from a three-letter alphabet, F, R and L (Forward, Right and Left: the instructions for ant movement). The series of commands are read by the ant from left to right and are functionally closed in a loop (broken arrows).

Figure 5.

Symbolic string representation of the ANT's navigation algorithm and the artificial transposon definition (FSA-ant implementation). (A)-(B) two possible forms of the symbolic sequences extracted from the ANT's FSA (beginning with the first FSA state and input=0). (C)-(D) examples of the symbolic strings that match the definition of the mature (closed symbolic string) (C) and immature (D) transposons. L is LFT, R is RGT, and F is FWD. The parts of the two symbolic strings corresponding to the transposon definition are underlined.

SEARCH TRANSPOSONS USING THE TRANSPOSON DEFINITION

1.	Chose an ANT chromosome at random
2.	Read its FSA state (reference to new state and action for input=0)
3.	Write the action found in 2 as the first symbol (F, R, L or N) in the allocated symbol string
4.	Read the next FSA state pointed by the previous state, if this points to the first FSA state, then halt
5.	Write the action found in 4 as the next symbol in the allocated symbol string
6.	While new state in 4 does not correspond to any previous FSA states, repeat 4-5
7.	The extracted symbolic string (for a given ant) is checked for correspondence to the transposon's definition and whether the transposon is mature or immature

Figure 6.
The procedures to search transposons, for the FSA-ant implementation.

Author Manuscript
Author Manuscript
Author Manuscript
Author Manuscript

Figure 7.

An example of transposon organization implemented in the case of the symbol-string chromosome ants. For the FSA-ant implementation [74], the transposon is the general periodic sequence $(A_1, A_2, \dots, A_n) * K$. In this case, $n = 4$, $K=3$. The last symbol in the period, A_n , is an arbitrary symbol (x).

Figure 8.
Examples of artificial transposons.
(A) A mature transposon and the state table corresponding to it. The transposon is a closed five-element cycle of state transitions (0, 17, 13, 21, 9, and again, 0).
(B) An immature transposon and the state table corresponding to it. It differs from the previous one by the presence of a NOP action in the last element and is not a cycle.
(C) Point mutation for transposon code.
(D) An example of transposon growth by substituting the terminating N element with an L, R or F. The parts of the two symbolic strings corresponding to the transposon definition are underlined.

Figure 9.

Mutagenesis caused by transposon growth in a symbol-string chromosome (SSC). Top (T_0 , initial stage) - the SSC for an ant with one of the highest performance scores (i.e. one of the best navigation programs). The chromosome begins with a sequence that corresponds to two periods of the transposon FFRFFF (the significant symbols are underlined). Middle (T_1 , later time) -transposon growth has damaged the excellent navigation algorithm, drastically reducing the host score. Bottom (T_2 , even later time) - continued growth continues to damage the host SSC.

A

```
-Check Ant #79 to pattern. Find pattern with NOP. Pattern is:
state #0...LFT/#4
state #4...RGT/#20
state #20...LFT/#23
state #23...LFT/#21
state #21...NOP/#18
Change NOP to node #0 action
state #21...LFT/#18
```

B

```
-Check Ant #51 to pattern. Find pattern with cycle. Pattern is:
state #0...FWD/#25
state #25...FWD/#1
state #1...FWD/#10
state #10...FWD/#21
state #21...FWD/#7
state #7...RGT/#0
Change last action to node #25 action: state #7...FWD/#0
Change point-cycle to random point: state #7...FWD/#16
```

Figure 10.

Excerpts from output file generated by our ant program in verbose mode, illustrating transposon mutation for immature (A) and mature (B) cases.

TRANSPONON-CAUSED MUTAGENESIS (ONE-PLACE MGE OPERATOR)	
1.	If the found transposon is immature (the last its element is N, NOP), then the N symbol is substituted by the action from the K th element (counted backward from the end) of the transposon symbol sequence
2.	If the found transposon is mature, then the last element of the transposon (which loops the sequence) is substituted by the K th transposon element, and the 'next state' reference is substituted by a random one (outside the states belonging to the transposon)
3.	The states of the FSA are updated after the transposon-caused mutagenesis
THE TRANSPOSITION (TWO-PLACE) MGE OPERATOR	
1.	Chose an ANT chromosome at random and perform the transposon search (using the transposon definition)
2.	If a mature transposon is found, then choose another ANT at random and copy the transposon to the ANT's FSA (overwriting the target ANT's FSA states)

Figure 11.
Key steps for the MGE operators in the FSA-ant implementation.

Figure 12.

Ant trail used in our experiments. The trail itself is a series of squares on a 32x32 white toroidal grid. Each cell is numbered sequentially, from the 1st to the 89th. The first higher-complexity gap is between the 64th and 65th cells, and the second is between the 67th and 68th.

Figure 13.

Numerical experiments, statistically demonstrating the increase of GA efficiency due to the effect of MGE operators. A comparison of the mean and best-of-generation score dynamics in the experiment (MGE operators active) to those in the control group (MGE operators disabled). Score values are averaged over 100 runs in both cases. Population size = 100; number of generations = 5000; the transposon sequence size varies from 5 to 11; crossover rate (P/bit)/generation = 0.0001; mutation rate (P/bit)/generation = 0.04; MGE1 and MGE2 operators rates are 0.0027 and 0.0075 (P/bit/generation), respectively. *Best test* is the best-of-generation scores and *Mean test* are the mean scores for the test runs; *Best control* is the best-of-generation scores and *Mean control* are the mean scores for the control runs.

Figure 14.

Numerical experiments with SSC ants, statistically demonstrating increased GA efficiency due to the transposon operator. A comparison of the mean and best-of-generation score dynamics in the experiment (transposon operator active) to the control group (transposon operator disabled). Score values are averaged over 100 runs in both cases. Size of population = 200; number of generations = 5000; crossover rate (P/bit)/generation = 0.008; mutation rate (P/bit)/generation = 0.024; MGE operator rate (P/bit)/generation = 0.008. *Best test* is the best-of-generation scores and *Mean test* are the mean scores for the test runs; *Best control* is the best-of-generation scores and *Mean control* are the mean scores for the control runs.

Figure 15.

The time complexity of our MGE algorithms compared to standard GA. Average evaluated cases to find a navigation algorithm reaching the n^{th} cell of the trail on Figure S1 plotted against trail length. Success rates of the tests and control are plotted on the secondary axis to the right, 0-100%. (A) FSA-ant tests, (B) SSC-ant tests. Scores averaged over 500 runs in both cases. For the FSA-ant tests the conditions are: size of population = 200; transposon sequence size, from 5 to 11; crossover rate (P/bit)/generation = 0.004; mutation rate (P/bit)/generation = 0.016, MGE1 and MGE2 operators rates are 0.006 and 0.006 (P/bit/generation), respectively. For the SSC-ant tests: population = 2000; crossover rate (P/bit)/generation = 0.008; mutation rate (P/bit)/generation = 0.016; MGE operator rate P/bit/generation=0.008. Maximum number of generations is 500; maximum number of an ant's steps is 420. TST are test data, CTRL – control data, ExpTrnd – exponential trend for the control data.

Figure 16.

An example of the population dynamics of transposons inside the gene pool of the host (ant) population over the first 500 generations. The curve of the maximum score dynamics for the ant population is given at the top of the diagram (marked with a sequence of squares; the secondary, score axis is given on the right of the diagram). Moments in which significantly more effective navigation algorithms emerged, and the corresponding shifts in the population's dominant transposon form are marked with downward-pointing arrows (see text for details). The solid, dashed and dotted lines correspond to different dominant transposons.

Figure 17.

Evolution of the ant's navigation programs under the selection of artificial transposons. (A) An example of a complicated, zigzag path in search of a lost trail. (B) An example of the

closed trajectory produced by four-fold execution of the transposon-bearing code 1 from the Table S1 (illustrated in (A)). (CJ An illustration of how the execution of a navigation program like (A) makes it possible to pass over the gap between cells #64 and #65 on our trail (the program is FRFFFFFRFFRFFL). The ant path is marked by red arrows. The directions of the arrows indicate the orientation of the ant.