Impact of Regulatory Genes on Optimization Behavior.

Daniel Ashlock and Wendy Ashlock

Abstract—In nature, regulatory genes determine which part of an organism’s genome is expressed. In this study a simple regulatory mechanism is used to modify linear representations. The regulatory mechanism substantially enhances exploration at the expense of exploitation. For complex, polymodal fitness landscapes the modification yields a substantial improvement in performance. A negative control example is designed that demonstrates the technique yields a remarkable degradation of performance on a unimodal optimization problem designed to interact poorly with the technique. Analysis shows that the regulatory mechanism creates the potential for insertion and deletion mutations within the linear representation. These mutations have the effect of substantially increasing the number of genomes one mutation away from any given genome. This has the effect of decreasing the diameter of any search space where they regulatory technique is implemented.

Keywords: Evolutionary Optimization, Nature Inspired Algorithms, Representation.

I. INTRODUCTION

The NIH’s online dictionary of medical terms defines a regulator gene as

“Genes which regulate or circumscribe the activity of other genes; specifically, genes which code for proteins or RNAs which have gene expression regulation functions.”

Regulator genes, organized into gene regulatory networks [15], control the timing and level of expression of proteins and regulatory RNA’s that permit correct functioning of the biochemistry of living organisms. A more complex attempt at gene regulation in an evolutionary computation system appears in [18]. In this study we implement a very simple type of gene regulation. Rather than being organized in a network, a regulator gene is added for each position in a linear evolutionary computation representation. This means that the action of the regulatory genes happens only in production of the expressed phenotype.

II. REGULATORY REPRESENTATIONS

The regulatory representation starts with any linear representation for evolutionary computation. This representation is then modified by adding a second binary chromosome of regulatory genes. This binary chromosome can be explicit or implicit in the representation. The length of the chromosome in the original representation and the binary gene are both lengthened by an amount called the regulatory margin.

Instead of using the original linear chromosome an expressed chromosome is derived by using only locations in which the matching position in the regulatory gene has a one. The resulting expressed chromosome may be too short or too long. If it is too short then either a worst-possible fitness is awarded or fitness is assessed on the partial chromosome. The real optimization representation in this study falls into the former category while the SAW problem falls into the latter category. If the expressed chromosome is too long, then only the initial segment of the correct length is used. We now give specific examples of regulatory gene representations and define the problems on which the technique is tested.

A. The SAW task

The self-avoiding walk (SAW) problem [1] is an evolutionary test problem with a rich, complex fitness landscape. The nominal representation for the SAW problem uses a string over the alphabet \( \{U, D, L, R\} \) as its chromosome. The letters correspond to moves on a grid in the directions: up, down, left, and right. The length of a SAW chromosome is at least the number of cells in the grid minus one. Fitness is evaluated by starting in the lower left corner of the grid and then making the moves specified by the chromosome. The sequence of moves made is referred to as the walk. If a move is made that would cause the walk to leave the grid, then that move is ignored. The walk can also revisit cells of the grid. Fitness is equal to the number of squares visited when the walk is completed. The problem is called the self-avoiding walk problem because optimal solutions using the minimum number of moves do not revisit squares; they are self-avoiding walks. An example of a SAW is given in Figure 1. SAW problems are defined by the size of the grid; the example is taken from the \( 5 \times 5 \) SAW problem. The length of a SAW chromosome is one less than the number of squares in the grid. Since the lower left corner starts full, this is exactly the number of moves required to traverse the grid.

The basic SAW problem is modified by incorporating regulatory genes implicitly into the representation. This is done by changing the alphabet to the eight character alphabet \( \{U, D, L, R, u, d, l, r\} \) where the lower case letters are not expressed (skipped) when performing the walk. The length of the gene for an \( X \times Y \) SAW problem is increased by a regulatory margin, but only the first \( XY - 1 \) expressed locations are used to evaluate fitness. This leaves the number of expressed moves the same in the standard and regulatory representations with only the connectivity of the fitness landscape induced by crossover and mutation changing. If too many loci are unexpressed then the regulatory representation will be at disadvantage because it cannot make enough moves.

The authors thank the National Science and Engineering Research Council of Canada for its support of this research.

Daniel Ashlock is at the Department of Mathematics and Statistics at the University of Guelph in Guelph, Ontario, Canada, N1G 2W1, email: dashlock@uoguelph.ca

Computer Science and Engineering Department, York University, Toronto, Ontario, Canada, M5J 1P3, email: washlock@cse.yorku.ca

The authors thank the National Science and Engineering Research Council of Canada for its support of this research.
B. Optimization of Real Valued Functions

In order to make a regulatory gene representation for real optimization, an explicit binary gene is added to a vector of real numbers. Only those loci in the real gene that correspond to a 1 in the regulatory gene are used. If insufficient real variables are obtained a worst-possible fitness is awarded, otherwise the first $d$ real values selected are used.

We use the real valued optimization gene regulatory representation to construct a negative control, given in Equation 1. This is a problem where using the regulatory gene representation yields a large degradation in performance in comparison to the unregulated representation. Equation 1 is a radially symmetric unimodal hill with its mode at $(1, 3, 5, 7, \ldots)$. The coordinates of the mode are different for each coordinate axis. The dimension $n$ of the hill can be varied. The action of regulatory genes is to insert and delete positions from the expressed genes. Since there is a distinct, unique correct answer for each coordinate, this function should perform very badly with a regulatory gene representation.

$$f_n(x_1, x_2, \ldots, x_n) = \frac{1}{1 + \sum_{k=1}^{n} (x_k - 2k - 1)^2}$$

The hypothesis under test is that the strength of the gene regulatory representation is its ability to explore. This means that its performance should be enhanced for functions with a large number of modes. Equation 2 is a sum of sinusoids with noncoincident periods with a small linear trend added. There are many optima and the linear trend ensures that the height of the optima varies.

$$g_n(x_1, x_2, \ldots, x_n) = \frac{1}{20n} \sum_{k=0}^{n} x_k + \sum_{k=0}^{n} \sin(\sqrt{k} \cdot x_k)$$

The last real function optimized is a sum of sixteen hills of the form

$$f(x_1, x_2, \ldots, x_n) = \frac{C}{\sum_{k=1}^{n} x_k^2 + 1}$$

The hills are recentered at a series of points at a distance of 0, 3, 6, \ldots, 45 from the origin. The constant $C$ is $i$ where $i$ is the index number $1, 2, \ldots, 16$ of the hill going outward from the origin. The actual location of hilltops is not just at the specified distance but at a random location with 6 units of the top of the next hill in. The location of the hill is otherwise selected uniformly at random among locations that satisfy the distance constraints from the origin and next hill in. The hills thus form an irregular series of steps of length at least 3 but at most 6 outward from the origin.

C. Tartarus with ISAc lists

The Tartarus problem [20], is a standard evolutionary computation test problem. Tartarus takes place on a 6x6 grid surrounded by impenetrable walls. Six boxes are placed away from the walls and with no boxes forming a close group of four boxes. An agent called the bulldozer is also placed on the grid, away from the walls. Examples of starting configurations for Tartarus are shown in Figure 2. During testing on a given board the bulldozer is permitted eighty moves. On each of these moves the bulldozer may turn left, turn right, or advance. The bulldozer may push one box ahead of it but may not push two boxes or push a box through a wall. The bulldozer’s goal is to place box sides against the walls. A box in a corner scores two points while a box against the wall away from a corner scores one point. The maximum possible score on a board is thus 10 points. In standard Tartarus the bulldozer is given knowledge of only the eight adjacent grids to plan its actions.

A good deal of past work on Tartarus exists. In the first publication on the problem [20], calculator-style memories
“store x in y” and “recall from y” were incorporated into standard parse trees. Scores near 5 were achieved with this representation. In [13] an artificial neural net was used as a Tartarus controller and the placement of sensors was permitted to evolve. It was found that placing a sensor two squares ahead of the bulldozer was helpful. In the original work, a parse tree was located that did not use its sensors and which scored relatively well. Since the parse tree must be encoding a fixed dance (no random numbers were available in the terminals and operations used) it follows that this parse tree was performing a fixed sequence of moves. Based on this observation, fixed strings of moves were studied as a baseline in [2]. Directly evolving an 80 character string was found to work poorly with maximum fitnesses around 2. Cycling through shorter genes and including a gene doubling and gene cleaving mutation initially described in [16] resulted in substantially improved performance. Fixed sequences of moves were located with an average fitnesses above 5 on a 5000 board cross validation set.

1) ISAc lists: The evolvable structure used to control Tartarus agents in this study are ISAc lists. The ISAc lists used here are a generalization of the ones presented in [2], [5], and [1]. The novel feature of these ISAc lists are that they permit multiple Boolean tests to be available rather than a single test and in that they have constants for comparison localized to the individual ISAc nodes. These ISAc lists were also used in [6] to provide agents in an ecological simulation. They were chosen for use as movement controllers in this study because they require few computational resource and, in earlier studies using them, been able to generate fairly sophisticated movement behaviors.

An ISAc list is an array of ISAc nodes. An ISAC node is a hextuple \((a, b, c, t, \text{act}, \text{jmp})\) where \(a\) and \(b\) are indices into the set of inputs available to the controller, \(c\) is a constant in the range \(0 \leq c \leq 3\), \(t\) is the type of the Boolean test used by the node, \(\text{act}\) is an action that the ISAc list may take, and \(\text{jmp}\) is a specification of which position in the list to jump to if the action happens to be a jump action. An ISAc list comes equipped with a set of Boolean tests available to each node. The tests available to the nodes in this study have types 0-3 and are, respectively, \(v[a] < v[b], v[a] < c, v[a] \leq v[b],\) and \(v[a] \leq c\). The array \(v[*]\), called the data vector is the set of inputs available to the Tartarus controller. In the controllers used in this study, this is the eight squares of the grid adjacent to the robot using the encoding empty=0, barrier=1, box=2.

Execution in an ISAc list is controlled by the instruction pointer. The instruction pointer starts at the beginning of the ISAc list, indexing the zeroth node. Using the entries \(a, b, c, t, \text{act},\) and \(\text{jmp}\) of that node, inputs \(v[a]\) and \(v[b]\) are retrieved for the controller’s current state and the Boolean test of type \(t\) is applied to \(v[a]\) and \(v[b]\) or \(v[a]\) and \(c\), as appropriate. If the test is true, then the action in the act field of the node is performed; otherwise, nothing is done and the instruction pointer moves to the next node. If that action is “jump,” the contents of the jmp field are loaded into the instruction pointer. For any other action the instruction pointer advances to the next instruction in the list. Action other than jump actions are executed according to their type: attempt to advance, turn right, or turn left. In addition to jump and the three possible moves relevant to the movement there is a NOP (no operation) action, which does nothing and leaves space for later mutational modification of the ISAc list. The NOP instruction is the source of implicit regulatory representation in the ISAc list representation. An ISAc list continues executing nodes until it generates a simulation-relevant action (left, right, or forward) or times out, a process described below. This action is returned, executed by the simulation software, and the data vector is updated to reflect the results of the action.

ISAc lists are initialized at random, filling in uniformly selected, semantically valid values to the six fields of each node. This means that it is not difficult to create an ISAc list that can run indefinitely without generating an action relevant to the Tartarus task. To prevent such “infinite loop” behaviors, there is a maximum number of ISAc nodes an ISAc list may execute. This limit is called the node limit for the ISAc list. If it exceeds this limit the list has timed out and its fitness evaluation ends immediately. The node limit is set to 2000 in this study. The count of ISAc nodes used is reset for each Tartarus board. This study uses two variation operators. The first is two-point crossover operating on the array of ISAc nodes with the nodes treated as atomic objects. The second is a mutation operator that first selects a node uniformly at random and then modifies one of its six fields to a new, valid value, also selected uniformly at random.

All forms of ISAc list used prior to this study have had implicit regulatory control in the form of the NOP instruction. A NOP can mutate into an active instruction and an active instruction may be removed from the execution of the ISAc list by mutating it into a NOP. In this study we disable the creation of NOP instruction in both initial generation and mutation to test the impact of the lack of regulatory control.

III. EXPERIMENTAL DESIGN

With the exception of the Tartarus experiments, all the evolutionary algorithms operate using size seven single tournament selection in a steady state environment [19]. In this model of evolution, groups of seven members are selected from the population. The two most fit members of each group are copied over the two least fit, replacing them in the population. These copies then undergo crossover and mutation. The combination of tournament selection, replacement, crossover, and mutation is termed a mating event. The Tartarus experiments use size four single tournament selection with a generational algorithm run for 250 generations. This choice matches earlier experiments. The generational version of tournament selection breaks the population into disjoint tournaments and performs a mating event for each to generate a new population. Note that one side effect of using tournament selection of this type is that the algorithms are elitist with the most and second most fit population members guaranteed to survive.
Population size is varied and is specified experiment-by-experiment. The variation operators for the Tartarus experiments are given in the description of the representation. All the other experiments use two point crossover and uniform mutation with the per-loci rate of mutation chosen to yield an expectation of one mutation per new chromosome (the rate is the reciprocal of the number of loci). It is folklore in the evolutionary computation community that this is a good mutation rate. In the Tartarus experiments the number of active loci is not a well defined quantity. A number of mutation for each new Tartarus controller was selected uniformly at random in the range 1-3.

Except in the experiment that studies changing the statistics of the regulatory loci, regulatory loci are both generated and mutated with a 50% chance each of zero and one. SAW and Tartarus experiments are initialized uniformly at random with semantically valid values. The real-valued optimization are initialized in a cube of size length $2(D + 1)$ centered at the origin for the negative control experiments and a cube of side length 15 centered at the origin for the polymodal function.

A. SAW Experiments

A collection of experiments with the $5 \times 5$ and $6 \times 6$ SAW genes were used to check the impact of changing population size. Population sizes of 10, 50, 250, and 1250 were used. Both the mean time to solution for runs that found a solution and the percentage of runs in which the algorithm failed to find a solution were saved. The best population size was then used to run $7 \times 7$, $8 \times 8$, and $9 \times 9$ SAW experiments. All experiments were run with both the standard and gene regulation representations. All experiments consist of 400 replicates run with independent random number seeds for 1,000,000 mating events.

An ongoing concern when using the gene regulatory technique is that the expressed gene may be too short. If the regulatory margin is small then, if mutations to the regulatory gene are 50/50 use/don’t use then it becomes very easy for a SAW gene to mutate to a state where it is too short. This, in turn, places a substantial burden on evolution to conserve the ‘use’ allele of regulatory genes. A set of experiments were done for the $5 \times 5$ SAW with regulatory margins of 4, 8, 12, 16, 20, 24, 28, and 32. For each regulatory margin the probability of having a ‘use’ was either 50/50 or set to make the expected number of ‘use’ alleles equal to the required expressed length of the gene. Correction of the probability a regulatory loci would be a 1 (‘use’) was done both for generation of the initial population and when mutation was applied. These experiments explore the degree to which a bias in the probability of generating a “use” allele is useful. These experiments also use 400 replicates.

B. Negative Control Experiments

Experiments are performed for the negative control function given in Equation 1 in $d = 2, 3, 4, 5, 6$, and 7 dimensions using both the standard and gene regulatory representations with population size 10. Mean time to solution and the number of times the algorithm failed to find a solution were saved. Each experiment with the negative control uses 400 replicates lasting 1,000,000 mating events. Mutation of loci in the negative controls consisted of adding a Gaussian random variable with a mean of zero an standard deviation of $\sigma = 0.2$.

C. The Highly Multimodal Function Experiments

Experiments were run in $d = 2, 3, 4, 5, 6, 7$ dimensions using the standard and gene regulatory representations with population size 10. Two collections of experiments using 100,000 and 1,000,000 were performed to assess the impact of giving additional evolutionary time to the algorithm. The average value of the best optima found over 400 replicates of the algorithm were saved along with the best optima located in any replicate. Mutation of loci consisted of adding a Gaussian random variable with a mean of zero an standard deviation of $\sigma = 0.2$.

D. Stepwise Hill Function

Experiments were run in $d = 2, 3, 4, 5, 6, 7$ dimensions using the standard and gene regulatory representations with population size 10. Two collections of experiments using 100,000 mating events were performed. The average value of the best optima found over 400 replicates of the algorithm were saved along with the best optima located in any replicate. Mutation of loci in the negative controls consisted of adding a Gaussian random variable with a mean of zero an standard deviation of $\sigma = 1.0$.

E. Tartarus Experiments

Tartarus experiments used populations of 120 length 60 ISAc lists. One experiment used NOP instructions in the usual fashion while the other used code modified not to use NOP instructions at all. In each replicate, the most fit member of the final population was saved and its fitness was re-evaluated on a fixed set of 5000 Tartrars boards to provide an accurate common-currency fitness for comparison.

IV. Results

Summary: the experiment with the negative control function was highly successful in demonstrating degradation of performance for the gene regulatory representation. All other experiments that compared regulatory and non-regulatory representations showed a substantial positive impact of using the gene regulatory representation.

A. SAW results.

Table I gives a side-by-side comparison of the gene regulatory and standard representations. These data show that:

1) Failure to find a solution is at a lower level for the regulatory representation, across the board.
2) The populations of size 10 are superior to all others, but the largest population used, with 1250 members, comes in second. This suggests that different search
mechanisms may dominate behaviors at these different population sizes.

3) The mean time to solution is obfuscated by the failure rate. The ratio of mean time to solution for standard:gene regulation is undefined twice and varies from 0.235 to 9.76 in the other cases. These comparisons can occur where the standard algorithm solved the problem as little as 2% or 6% of the time and are based only on runs where the algorithm solved the problem.

4) For the largest problem sizes tested, the standard algorithm failed to solve the problem at all in 400 replicates.

It is clear that the mean time to solution is not a useful statistic when the failure rate is high or when it is very different for the two different representations being tested. These two cases cover all the instances tested. We now turn to the impact of setting the statistics of the regulatory loci to those needed to make the expressed chromosome, on average, the right length.

Figure 3 shows 95% confidence intervals on the probability that a replicated of an experiment performed with the $5 \times 5$ SAW problem will fail. These form a side-by-side comparison of 50/50 statistics for the regulatory genes with corrected statistics. The $5 \times 5$ SAW requires 24 loci. The corrected statistics are helpful only when the regulatory margin is small. With a regulatory margin of 4 loci (17%) there is a significant benefit to using the corrected statistics. When the regulatory margin is 12 or more, using corrected statistics is not significantly helpful and is only marginally helpful at a regulatory margin of 8. This suggests that, unless regulatory margin is very expensive for some reason and so needs to be small, there is no reason to worry about correcting the statistics of the regulatory loci.

B. Negative Control Results

The negative control experiment uses a function constructed so that any mutation of the regulatory genes will cause a large decrease in the fitness of a highly fit gene.

Table II shows that evolution is unable to compensate for this regulatory fiasco with time to solution always much higher for the gene regulatory representation and failure rate zero for the standard representation and increasing with dimension to almost half in seven dimensions.

This experiment demonstrates that the gene regulatory representation should be used with care. It is best in polymodal environments and can fail in a spectacular fashion in unimodal environments. One experiment that might be interesting is to run the experiment again on a single hill with its peak at the origin. In this cases it is plausible that the gene regulatory technique would yield little advantage or penalty in performance.

C. The Polymodal Real Function Results

The results shown in Table III support the following conclusions:
1) The ability of the gene regulatory representation to find new optima is clearly better than that of the standard representation. The best optima located is always better for the regulatory representation and the mean of the best optima located across replicates is significantly better except for the regulatory representation and the mean of the best optima located is always better for the standard representation. Note that the bad results for the negative control also support this conclusion; the regulatory representation explores even when such exploration is counter-productive.

2) When the number of fitness evaluations used during evolution is increased from 100,000 to 1,000,000 the gene regulatory representation experiments always significantly increased the quality of optima they found.

3) There was no significant change in the quality of optima located by the standard representation when they were permitted ten times as many fitness evaluations. This demonstrates the exploratory power of the regulatory representation. Note that the bad results for the negative control also support this conclusion: the regulatory representation explores even when such exploration is counter-productive.

### TABLE II

<table>
<thead>
<tr>
<th>Dim.</th>
<th>Control Time to Solution</th>
<th>Control Fail</th>
<th>Gene Regulation Time to Solution</th>
<th>Gene Regulation Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>31.8 ± 1.8</td>
<td>0%</td>
<td>108 ± 8.2</td>
<td>0%</td>
</tr>
<tr>
<td>3</td>
<td>89.1 ± 3.3</td>
<td>0%</td>
<td>614 ± 90.6</td>
<td>0%</td>
</tr>
<tr>
<td>4</td>
<td>183 ± 6.4</td>
<td>0%</td>
<td>6099 ± 2570</td>
<td>0.25%</td>
</tr>
<tr>
<td>5</td>
<td>312 ± 9.0</td>
<td>0%</td>
<td>46890 ± 12100</td>
<td>4.5%</td>
</tr>
<tr>
<td>6</td>
<td>494 ± 13</td>
<td>0%</td>
<td>121000 ± 21800</td>
<td>16%</td>
</tr>
<tr>
<td>7</td>
<td>699 ± 17</td>
<td>0%</td>
<td>1900000 ± 30407</td>
<td>48.3%</td>
</tr>
</tbody>
</table>

### TABLE III

<table>
<thead>
<tr>
<th>Dim.</th>
<th>Regulatory Mean fitness</th>
<th>Regulatory Best fitness</th>
<th>Control Mean fitness</th>
<th>Control Best fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2.27 ± 0.02</td>
<td>3.25</td>
<td>2.59 ± 0.02</td>
<td>2.98</td>
</tr>
<tr>
<td>3</td>
<td>3.74 ± 0.01</td>
<td>4.11</td>
<td>3.54 ± 0.02</td>
<td>3.97</td>
</tr>
<tr>
<td>4</td>
<td>4.71 ± 0.01</td>
<td>5.05</td>
<td>4.54 ± 0.01</td>
<td>4.97</td>
</tr>
<tr>
<td>5</td>
<td>5.68 ± 0.02</td>
<td>5.98</td>
<td>5.91 ± 0.01</td>
<td>5.85</td>
</tr>
<tr>
<td>6</td>
<td>6.58 ± 0.02</td>
<td>6.95</td>
<td>6.49 ± 0.01</td>
<td>6.82</td>
</tr>
<tr>
<td>7</td>
<td>7.54 ± 0.04</td>
<td>7.93</td>
<td>7.50 ± 0.01</td>
<td>7.85</td>
</tr>
</tbody>
</table>

### TABLE IV

<table>
<thead>
<tr>
<th>Dim.</th>
<th>Regulatory Mean</th>
<th>Regulatory Best</th>
<th>Control Mean</th>
<th>Control Best</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>11.3 ± 0.293</td>
<td>18.2</td>
<td>6.02 ± 0.233</td>
<td>13.6</td>
</tr>
<tr>
<td>3</td>
<td>7.95 ± 0.286</td>
<td>17.0</td>
<td>7.10 ± 0.228</td>
<td>16.2</td>
</tr>
<tr>
<td>4</td>
<td>7.31 ± 0.247</td>
<td>14.4</td>
<td>7.33 ± 0.266</td>
<td>15.3</td>
</tr>
<tr>
<td>5</td>
<td>7.86 ± 0.228</td>
<td>12.7</td>
<td>7.30 ± 0.217</td>
<td>12.7</td>
</tr>
<tr>
<td>6</td>
<td>8.13 ± 0.214</td>
<td>15.6</td>
<td>8.07 ± 0.224</td>
<td>15.6</td>
</tr>
<tr>
<td>7</td>
<td>8.21 ± 0.222</td>
<td>14.5</td>
<td>8.22 ± 0.249</td>
<td>15.6</td>
</tr>
</tbody>
</table>

### D. Stepwise Hill Function Results

The results shown in Figure IV-E are more complex than those for the other real-optimization experiments. The best optima located is better trades honors back and forth between the regulatory and standard representations. The average performance, however, is significantly better for the regulatory representation in $d = 2, 3$, and 5 dimensions and has no significant difference in $d = 4, 6$, and 7 dimensions. The initialization places the beginning population inside a hypercube about the origin that contains only the first several optima (the number of optima increases as the hypercube of initialization increases in dimension because its diameter is $15\sqrt{d}$). This means that ability to search outside of the domain of initialization will pay benefits as the quality of the optima increase with their distance from the origin.

Since the number of optima for this function is held constant as the dimension increases, the degree to which one hill shadows another will decrease as the dimension increases. This will reduce the advantage of exploration and increase the degree to which climbing hills is an advantage. The results reflect this and confirm the hypothesis that the standard representation is more effecting a hill climbing (exploitation) while the regulatory representation is better at exploration.

### E. Tartarus

The Tartarus experiments were added when the authors belatedly noticed that the NOP instruction, used in analogy with machine language programming, in ISAc lists were a form of implicit gene regulatory control. Unlike the gene regulation used in the other experiments in this study, a NO does not turn an existing gene on or off. It instead eliminates it and replaces it with a gene that does nothing. In spite of this, the use of NOPs is similar to the gene regulatory representation. In addition, a review showed that the experiment of turning off the NOPs had not been done.

The work published in [7] demonstrates that the Tartarus problem, using ISAc lists, is polymodal. Since our results thus far indicate this is the right environment in which to test gene regulatory representations comparing ISAc-based Tartarus with and without NOPs was deemed worthy of inclusion. Common-currency best-of-run fitnesses for 100 replicates with and 100 replicates without NOPs are displayed in Figure IV-E. The results support the hypothesis...
One can make an evolutionary computation problem easier by reducing the number of local optima. We adopt the following definition of optima for the sake of discussion:

**Definition 1:** An **α-optimum** for an evolutionary computation problem is a point in the space of chromosomes where a single application of a variation operator has a probability of α or less of improving fitness.

Since the non-Tartarus experiments in this study used a uniform mutation operation there is a positive, if ridiculously small, probability of immediate mutation to a global optimum. In a strict mathematical sense none of the problems have local optima, just things that are more than 99,999,999,999,999% likely to act like local optima. The definition of α-optima lets us leave this technical quibble aside so that we can look at the impact of gene regulation on the structure of the fitness landscape.

If the **genetic neighborhood** of a chromosome is the set of chromosomes reachable by a single application of a variation operator, then the impact of using the gene regulatory operator is to increase the size of the genetic neighborhood. This has the effect of decreasing the diameter of the fitness landscape as a search space and, potentially, it merges many local optima (at a given α) into other (local or global) optima. When there is a single optima in the space, this has the effect of deflecting the search power of the variation operators in many directions; Function 1 was successfully designed to make those directions counterproductive.

Another issue is that of population size. In [8] it was shown that some evolutionary algorithms function best at very small population sizes. The paper conjectures that a phenomena analogous to genetic drift in biology replaces selection and variation as the primary search mechanism of evolution when the population size is small. Small population sizes are already known to function well on simple, unimodal problems [10]; the work in [8] also demonstrated that small populations are the best choice for very complex polymodal problems such as the parity problem in genetic programming [14]. The results in this study show that the gene regulatory representation favors small population sizes. This is a weak results, as only a few problems have been tested, and we note that large population sizes also turn in a tolerable performance for the SAW problem. This agrees well with the parity results in [8]. We in turn conjecture that a genetic-drift like mechanism may be important in search using the gene regulatory modification of linear representations.

The experiments with the stepwise hill functions cross the boundary between the regions where the regulatory representation and the standard representation exhibit superior performance. The enhanced exploratory ability of the regulatory representation meant that the best results always appeared in a run that used the regulatory representation but the average performance laurels were divided. It is reasonable to conjecture, based on the results of changing the dimension, that the **density** of optima is an important factor with the regulatory representation gaining an advantage when the optima are more densely packed.
VI. Next Steps

This study is a first foray into the use of a very simple form of regulatory gene to modify the behavior of an evolutionary algorithm. There are many linear representations that can be explored with the regulatory representation. Linear grammatical representations for genetic programming [9], linear representations for finite state classifiers [4], as well as a broad variety of other possible real-valued optimization problems. Any generative representation that uses a linear set of commands to specify an object [3], [11] is a potential target for modification—a form of the gene regulatory technique presented here.

The nonlinear responses of the SAW experiments to population size merit an additional look. In this experiment we saw that a large population is good and a tiny one is better. There is room to try still larger populations and, of course, the sensitivity to population size was tested only on one problem. Since the different problems reacted in a very different manner to imposition of the regulatory genes there is no especial reason to imagine sensitivity to population size will stay the same.

So far only very rudimentary parameter tuning of the gene regulatory modification to linear representations has been performed. The correct ratio of mutations that modify the regulatory genes to mutations that modify the underlying representation (SAW moves, real variables) is probably an important parameter and it is also probably problem dependent. Additional exploration of this parameter is a clear avenue for future research.

A. Possible Improvements

The experiments in this study show that the gene regulatory representation explores effectively but exploits ineffectively. This strongly suggests that a hybrid representation, in which the gene regulatory representation is in charge of search but not convergence, is a good area for future research. This could be done in a very simple fashion, by simply compiling the expressed genes in the regulatory representation to a standard population at some point and then continuing evolution with a standard algorithm. More complex approaches are also available such as using the gene regulatory representation as the input to a Baldwinian algorithm [17].

synteny is the phenomena of physical co-localization of genes or loci that affect related systems within a living creature. It results in a lower rate of recombinational disruption of linkage between those loci. The gene regulatory representation could easily be modified, by changing the mutation operator that modified the regulatory genes to either permit or encourage the emergence of synteny. This modification would probably need to be coupled with the use of a variation operator similar to Holland’s inversion operator [12] that permitted loci that work together to regulation, via selection, their rate of crossover based disruption.

References