OPEN ACCESS

BIO-complexity

Research Article

Climbing the Steiner Tree—Sources of Active Information in a Genetic Algorithm for Solving the Euclidean Steiner Tree Problem

Winston Ewert,[1]* William Dembski,[2] Robert J. Marks II[1]

[1] Department of Electrical and Computer Engineering, Baylor University, Waco, Texas, USA; [2] Discovery Institute, Seattle, Washington, USA

Abstract

Genetic algorithms are widely cited as demonstrating the power of natural selection to produce biological complexity. In particular, the success of such search algorithms is said to show that intelligent design has no scientific value. Despite their merits, genetic algorithms establish nothing of the sort. Such algorithms succeed not through any intrinsic property of the search algorithm, but rather through incorporating sources of information derived from the programmer's prior knowledge. A genetic algorithm used to defend the efficacy of natural selection is Thomas's Steiner tree algorithm. This paper tracks the various sources of information incorporated into Thomas's algorithm. Rather than creating information from scratch, the algorithm incorporates resident information by restricting the set of solutions considered, introducing selection skew to increase the power of selection, and adopting a structure that facilitates fortuitous crossover. Thomas's algorithm, far from exhibiting the power of natural selection, merely demonstrates that an intelligent agent, in this case a human programmer, possesses the ability to incorporate into such algorithms the information necessary for successful search.

Cite as: Ewert W, Dembski W, Marks II RJ (2012) Climbing the Steiner tree—Sources of active information in a genetic algorithm for solving the Euclidean Steiner tree problem. BIO-Complexity 2012(1):1-14. **doi:**10.5048/BIO-C.2012.1

Editor: Douglas Axe

Received: October 21, 2011; **Accepted:** March 6, 2012; **Published:** April 5, 2012

Notes: A *Critique* of this paper, when available, will be assigned **doi:**10.5048/BIO-C.2012.1.c.

* Email: evoinfo@winstonewert.com

INTRODUCTION

Genetic algorithms and simulations inspired by natural section have been offered as evidence for—or even convincing proof of—the abilities of neo-Darwinian evolution [1-5]. These algorithms are search algorithms. With all such algorithms, a question is posed, such as what is the best antenna design, what is the shortest route through several cities, or what is the most central meeting point. All of these questions admit various approximate answers, but not all answers are equally good. Indeed, some are better than others—for example one route may be shorter than another, or one antenna design may be more efficient. A *search algorithm* seeks the optimal answer (ideally) or at least a reasonably good answer (usually) by evaluating many possible candidate solutions. It can, for instance, try many possible paths and use the results of each path to guide which path it will try next. A *genetic algorithm* is a search algorithm that uses procedures that mimic natural selection and random mutation to determine which candidate solutions to try next. Genetic algorithms have been applied successfully to a wide variety of problems [6-8], thus demonstrating their usefulness as a tool for conducting searches.

Given the obvious success of genetic algorithms that are proposed to mimic natural selection, one might ask how anyone questions the efficacy of Darwinian processes. If computer simulations of evolution can conduct successful searches, surely biological evolution can do so as well, some say. It may be, however, as advocates of intelligent design have argued, that such algorithms succeed by incorporating information about the target [9,10]. This is sometimes misunderstood as the claim that all such search algorithms work by sneaking the exact answer into the algorithm. On this view, when a genetic algorithm locates its target, it is only revealing that which has been cleverly hidden in the algorithm, like a magician pulling a rabbit from a hat. Certainly, some simulations such as Dawkin's METHINKS IT IS LIKE A WEASEL function this way [11,12].

Nevertheless, many simulations do not contain a fully articulated solution hidden behind lines of code that can be readily reconstructed by inspecting the program without running it. Proponents of intelligent design admit this possibility, arguing

that such algorithms include information even when the solution has not been explicitly front-loaded into the simulation. The algorithm need not contain the explicit solution that is later found by running the algorithm. Rather, its code is front-loaded with specialized information for how to find that solution. Instead of the explicit answer, the means by which the eventual product can be found is front-loaded.

Imagine a bird-watcher who knows nothing about birds. He spends his time searching for birds in places that are unlikely to contain them, such as the grocery store, abandoned mines, or the international space station. One might assist the bird-watcher by telling him exactly where a bird can be found. But he can also be assisted in more subtle ways. Informing him that birds live in places like forests and not aboard space stations assists him, even though he is not being provided the exact coordinates of a bird.

The claim about genetic algorithms is not they have produced a hidden answer, like a bird-watcher using a precise set of coordinates to locate a bird. Instead, the claim is that the bird-watcher has been provided with details about the habitat and behavior of birds. This information assists in his search, enabling him to find the birds he wants to watch.

The Darwinist claim is that no such assistance is required. Rather, natural selection is innately capable of solving any biological problem that it faces. Analogously, a genetic algorithm ought to be able to succeed given nothing more than the description of the problem faced. It should not be necessary for an intelligent agent to tune or direct the evolutionary process. Any process so tuned is a teleological process, not a naturalistic one. The argument from genetic algorithms depends on maintaining the ateleological status [13].

Conservation of information theorems, such as the No Free Lunch Theorems [14], place limitations on the abilities of search algorithms. Any search algorithm uses resources to try to find its target. Different algorithms exist, but as long as they avoid searching the same place multiple times, given the same resources they will have the same average performance. As a result, excluding algorithms that visit the same place multiple times, no algorithm can be claimed to be superior to any other algorithm [15-17]. At best, it can be superior over some subset of all fitness functions. In consequence, observing that evolutionary algorithms are search algorithms does nothing to explain their success.

Genetic algorithms typically succeed because programmers incorporate problem-specific knowledge into the search algorithm. Various examples have been published in the literature. Avida [4], a program purported to demonstrate evolution, works by rewarding simpler versions of complex components [18]. Dawkin's "METHINKS IT IS LIKE A WEASEL" simulation [1] works by providing the distance to a target phrase [11]. **Ev** [3], another program purported to demonstrate evolution, works by providing the distance to a target along with a biased genomic representation [12]. Such algorithms do not demonstrate the abilities of undirected processes, but rather the powerful combination of human intelligence and brute force computing power.

The ID critic David Thomas openly disputes this assessment. To defend the creative power of genetic algorithms, he has presented a genetic algorithm for solving the Steiner tree problem, a well-known difficult computer science problem, in the journal *Skeptical Inquirer* [19]. Further details of the genetic algorithm are presented on the Panda's Thumb blog [20]. The actual code for the algorithm is given in the original Fortran [21] as well as in a rewritten version in C++ [22]. The problem is similar to that of building a road network between several cities. Based on his algorithm's success, Thomas claims that,

> … two pillars of ID theory, "irreducible complexity" and "complex specified information" [have been] shown not to be beyond the capabilities of evolution, contrary to official ID dogma. [19]

In the conclusion of his Panda's Thumb blog post, Thomas issued a challenge to ID advocates:

> If you contend that this algorithm works only by sneaking in the answer (the Steiner shape) into the fitness test, please identify the precise code snippet where this frontloading is being performed.

The above quote shows that Thomas is under the misapprehension that intelligent design advocates claim that the actual answer is encoded into the algorithm. This is not in fact what intelligent design advocates claim, as will be shown later. Rather, we say that success is due to prior knowledge being exploited to produce active information in the search algorithm. Although the code does not include the actual Steiner shape, it does include a tuned algorithm for how to find the Steiner shape.

Thomas appears to admit the possibility of such fine-tuning, but denies that he himself has done so [19]:

> Now, I *could* have implemented a "Fitness Function" that was designed to produce the formal Steiner solution only. For example, I could have taken steps to favor "shortest connectivity between all pairs of fixed nodes," or perhaps "junctions between three segments meeting at 120-degree angles." … But that would have required "knowledge of the target" at each and every step, and that was something I purposely wanted to avoid.

While Thomas did not implement functions to favor either of the particular properties mentioned in this quote, he has in fact implemented algorithms that depend on knowledge of the target, as we will show. In addition, in response to Thomas's challenge, we will identify the precise code snippets where the frontloading is being performed.

Not all critics of intelligent design subscribe to the strawman that ID advocates think all search algorithms include their answer, but instead provide more sophisticated arguments [5]. Discussing their arguments is outside the scope of this paper. Nevertheless, our prior work [11, 12, 18] as well as this current paper demonstrate by example that success is due to the active information in the search algorithm.

Background on the Steiner tree problem

The Steiner tree problem can be visualized as cities that need to be connected by a highway system (Fig. 1). The problem is to construct the road as inexpensively as possible. The most straightforward approach would be to build highways between the cities as depicted in Figure 1B. However, it is often possible to produce a shorter road system by having highway interchanges outside of the cities. Figure 1C shows a highway system where this is the case. For the purposes of this problem, we ignore the actual travel time on the highway. We are only interested in the cost of building the roads, which we assume to have a fixed per kilometer construction cost. The goal of the Steiner tree problem is to find the highway network that costs the least amount to build.

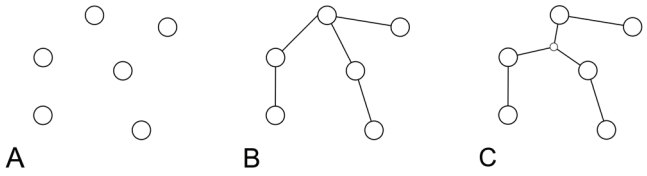

Figure 1: The Steiner tree problem. Shown are (A) cities that need to be connected by a highway system, (B) cities connected directly to each other, and (C) cities connected through an interchange. **doi:**10.5048/BIO-C.2012.1.f1

Properly, this is the *geometric* or *Euclidean* Steiner tree problem. It is an NP-complete problem; however, a polynomial time approximation algorithm exists [23]. This means that although no efficient algorithm to find the optimal solution is believed to exist, an efficient algorithm to find a good solution does exist.

Thomas is not the only one to solve Steiner tree problems using genetic algorithms [24-26]. The problem has also been attacked in the more general form of the graphical Steiner tree problem [27,28], as well as in the rectilinear Steiner tree problem [29,30]. The various versions of the Steiner tree problem differ in how they assign distances between the added interchanges and the cities.

All of these algorithms make use of the properties of the Steiner tree problem in order to make it easier to solve. For example, if the locations for the interchanges are known, figuring out how to connect those interchanges in the cheapest way is exactly the same as the *minimum spanning tree problem* which can solved quickly by using Prim's algorithm [31]. The number of possible locations for the interchanges can also be reduced, so that the algorithm does not have to consider as many interchanges. In the case of the rectilinear Steiner tree problem the number of possible interchanges can actually be reduced to be $O(n)$ where n is the number of cities [30]. Each of these algorithms makes use of the specific nature of the Steiner tree problem in order to effectively solve it.

Jesus *et al.* [24] writes:

> … the no-free-lunch theorems (NFL) are addressed to the evolutionary algorithms, ensuring the hopelessness in [sic] the existence of a magic algorithm to solve any problem, or in one that outperforms all others. Unless there exists [sic] some suitable operators that are correlated to the features of the problem, there is no reason to believe the algorithm will do better than a random search. Putting in other terms, it is necessary to 'tailor' our genetic algorithm to the [Euclidean Steiner Tree Problem], to became [sic] useful in this context.

In other words, these algorithms are tuned to the Steiner tree problem far more than just by describing the nature of the problem in a fitness function. Rather they all make use of critical knowledge about good Steiner tree problem solutions in order to find better solutions.

These other algorithms are solving Steiner tree problem orders of magnitude more difficult than the ones considered by Thomas. Thomas solves problems of five or six cities, while other genetic algorithms handle 100 cities, and in one case as many as 1000 cities. The difficulty of the problem scales exponentially, making the size of the problem solved by the other algorithms very impressive when compared to the small version solved by Thomas's genetic algorithm.

Definitions

Thomas's algorithm. The "genome" used for the genetic algorithm consists of three parts, shown in Figure 2, and described below.

1. The number of interchanges:
   This is an integer in the set {0,1,2,3,4}.
2. The coordinates of the interchanges:
   Each interchange has an integral X and Y coordinate in the set {0,1,2,…,999}. This part of the genome specifies where the interchanges will be located. The coordinates function like longitude and latitude to specify the exact location of the interchange.
3. The connections between the cities and interchanges:
   For each possible connection there is a Boolean value indicating whether or not a connection should be made. It is necessary to indicate which cities have highway connections to which other cities and interchanges. For each pair of cities or interchanges, there is a single value, either 'T' or 'F,' that indicates whether or not a road is built between those pairs of cities.

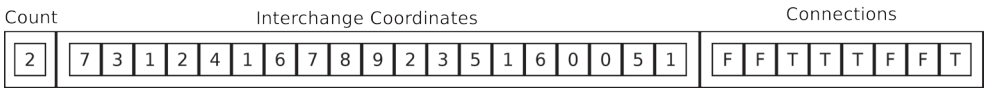

Figure 2: A depiction of the structure of the genome for Thomas's genetic algorithm, split among the different components. Sizes of the different sections are not to scale. The count is actually two symbols long, the coordinates are 24 symbols long, and the connections are 36 symbols long. **doi:**10.5048/BIO-C.2012.1.f2

This genome is of a fixed length, always containing the coordinates and connections for the maximum number of interchanges. This simplifies the genetic algorithm because it not necessary to implement deletion or insertion mutations. When the number of interchanges is less than four, the additional data is ignored as "junk DNA."

The cost function for a solution is

$$C(x) = \begin{cases} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} D(i,j) \;\; if\ all\ cities\ connected \\ 100000\ otherwise \end{cases} \quad (1)$$

where n is the number of nodes, cities plus interchanges, and

$$D(i,j) = \begin{cases} \sqrt{(X_i - X_j)^2 + (Y_i - Y_j)^2} \;\; if\ R_{i,j} = 1 \\ 0\ otherwise \end{cases} \quad (2)$$

where X_i is the x-coordinate of node i and Y_i is the y-coordinate of node i. $R_{i,j}$ = 1 if and only if there is a road between nodes i and j.

If the roads are not connected such that it is impossible to drive from one of the cities to any of the other cities, that solution is deemed incorrect. A very high cost, 100,000, is given to the invalid solution to make sure it is selected against. As long as the solution is connected properly, the cost of the solution is simply the sum of the distance between the connected cities and interchanges. This distance is calculated using the standard Euclidean distance.

The search algorithm itself is a genetic algorithm. Each generation consists of 2000 members. A variant of roulette-wheel selection is used with a skew factor to determine which members of a population will become parents of the next generation. The next generation is produced using single-point crossover between two parents. One tenth of the population has three point mutations applied to the genome. Elitism is also used, preserving the best of one generation to the next.

Figure 3A shows the specific instance of the Steiner tree problem on which Thomas's genetic algorithm was demonstrated. All experiments in this paper and in the original presentation of the algorithm are done using this problem. Thomas would eventually demonstrate his algorithm on a six cities problem. However, because much more detail is given in his description of solving the five cities problem, that is where we are focusing.

Defining Performance. Every search algorithm terminates with a solution that has a specific cost. The optimal solution has the lowest cost; most solutions are not optimal. An algorithm with good performance should produce a low cost solution with a high probability. Performance is therefore presented as a graph of the probability of a particular solution being the outcome of a search process *versus* the cost of that solution. An algorithm that performs well should have larger spikes at lower cost.

Defining Success. What does it mean for a search algorithm to succeed? The actual question of what is deemed a success is arbitrary. Given the entire set Ω of possible solutions, some subset of those is defined to be the target T. A search algorithm is deemed a success if it produces a solution ∈ T. T can be defined in any way suitable for the problem under consideration.

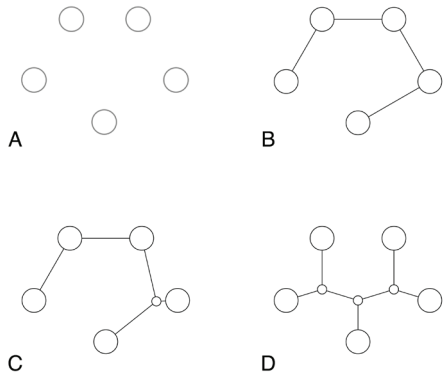

Figure 3: The Steiner tree problem used by Thomas, and various solutions. Shown are (A) the city configuration, (B) the minimum spanning tree or "trivial" solution, (C) the solution most commonly found by Thomas's algorithm with a cost of 1224, and (D) the optimal solution with a cost of 1212. **doi:**10.5048/BIO-C.2012.1.f3

In an attempt to establish the success of his algorithm and the difficulty of the problem being solved, Thomas points out that there are a large number of possible solutions.

> All together, there are about $7*10^{24+10} = 7*10^{34}$ possible organisms in the "hypervolume" - a large number indeed. [20]

Pointing out the large number of possible organisms, however, does not establish that the problem being solved is difficult. The actual number of distinct solutions is lower than the total number of solutions, because the same solution can be expressed in multiple ways. For example, the order of the interchanges in the genome can reordered without changing the cost. Additionally, the distribution of costs can be such that it is easy to obtain a solution with a low cost.

We define three targets, each based on a certain cost threshold. Following the original code and the description of Thomas, all cost values are reported rounded down, keeping only the integer part.

- Adequate

$$T_{adequate} = \{x \in \Omega | 1212 \leq C(x) < 1246\} \quad (3)$$

  Any solution that costs less than 1246, in other words the trivial solution depicted in Figure 3B.

- Good

$$T_{good} = \{x \in \Omega | 1212 \leq C(x) < 1225\} \quad (4)$$

  Any solution at least as good as 1224. This is the solution that the genetic algorithm usually finds, as depicted in Figure 3C.

- Optimal

$$T_{optimal} = \{x \in \Omega | 1212 \leq C(x) < 1213\} \quad (5)$$

Any solution that has a cost of 1212. Thomas indicates that this is the optimal solution [20], and we have no reason to doubt it. An example is depicted in Figure 3D.

In most cases, data for optimal solutions is not shown because in most possible configurations the optimal solution was not found in any of our Monte Carlo experiments (see Results).

Active Information. In order to numerically quantify the assistance that this search algorithm received, we use the concept of active information [9]. Active information is defined to be

$$I_+ = -\log_2 \frac{p}{q} \quad (6)$$

where q is the probability of the success of the search algorithm and p is the probability of the success of a baseline search algorithm. It measures how much more likely a search algorithm is to succeed relative to a baseline search algorithm. Using this measurement the performance of a search algorithm can be quantified. The baseline is necessary so that we can have an idea of the difficulty of the search algorithm according to some base reference and do not regard algorithms which solve easy problems as wildly successful. A baseline search is typically taken to be a single random guess at the answer.

RESULTS

The goal of our research was to determine what, if any, fine-tuning was required in order to produce Thomas's genetic algorithm. Does the genetic algorithm have the ability to solve the problem given only a description of the same? Or does the genetic algorithm bear the marks of an algorithm written to solve a particular problem and making use of prior knowledge about that problem?

In order to accomplish this we implemented a version of the algorithm based on the description given by Thomas [20]. However, the program derived from that description did not reproduce the reported results. Careful reading of the provided Fortran code showed that the description made a number of omissions with respect to the algorithm actually implemented. Thomas's C++ follows the algorithm of the Fortran code in most respects. As our results will show, some of these differences proved to have a significant effect on the performance of the genetic algorithm.

It should be emphasized that fine-tuning genetic algorithms is common practice. There is nothing unreasonable about the practice. It is in fact necessary, and very useful for producing results from genetic algorithms. Problems arise when attempting to draw inference from a fine-tuned simulation to non-fine-tuned biological reality. We will demonstrate that the fine-tuning is necessary to the success of the algorithm; consequently, the results cannot be used to defend the success of search algorithm in the absence of fine-tuning.

Comparing empirical performance

In what follows we will be comparing Thomas's genetic algorithm to that of a random query search. Data on the performance of the search algorithm was obtained by Monte Carlo simulation. In the case of genetic algorithms, we ran 5615 distinct simulations, keeping track of how many times a particular cost was found. In the case of random query searches, we ran 11,228,106,000 random queries to determine the empirical probability distribution for the cost of a random solution. The actual performance of a random query search was calculated by using probability theory and this empirical distribution.

Figure 4 shows the performance of the genetic algorithm presented by Thomas as compared to a random query algorithm. For Thomas's algorithm, each generation is run for 1000 generations giving a total of 2,000,000 queries or cost evaluations performed. That is, the algorithm considers two million distinct solutions and measures the cost of each solution. At the end of that process the best solution found during those queries is produced. The random query algorithm makes 2,000,000 random queries and reports the best of all of those queries. Figure 4 shows that obtaining a low cost solution to the Steiner tree problem is not as hard as it would at first appear.

The random query algorithm produces a cost of 1246 with probability slightly lower than 0.7. This solution corresponds to the minimum spanning tree solution (Fig. 3B), where all highway junctions must be inside cities, and no interchanges can be added. This restriction simplifies the problem; there is a well-known efficient algorithm known as Prim's algorithm for finding the optimal solution [31].

The reason why this solution is so common has to do with the genotype's configuration (Fig. 2). One fifth of all genomes have zero interchanges. For five nodes there are $\binom{5}{2} = 10$ possible connections. This results in 1024 possible ways to connect the nodes, at least one of which corresponds to a minimum spanning tree. This means that at least one in 5120 genomes corresponds to this minimum spanning tree solution. Given two million random queries, *not* finding this trivial solution would be an improbable event. Repeated random queries will quickly find the minimum spanning tree with high probability. As a result, having a search algorithm find it is no great success.

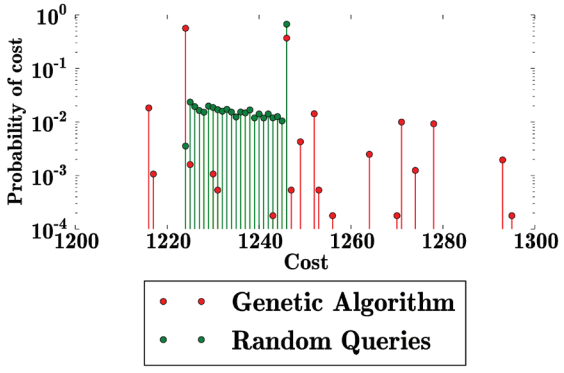

Figure 4: Performance of the genetic algorithm compared to that of random queries. **doi:**10.5048/BIO-C.2012.1.f4

In his discussion, Thomas presents a number of non-Steiner solutions produced by his algorithm. These correspond to some of the bumps in Figure 4. He says,

> I call the non-Steiner solutions "MacGyvers"…. The "MacGyver" solutions [the various non-optimal solutions] are not as elegant and pretty as the formal Steiner solutions, but they get the job done, and often quite efficiently. [20]

One of these solutions mentioned by Thomas [20] is the minimum spanning tree solution presented above; the other two solutions he mentions are more expensive. Such solutions should not be considered as "quite efficient" or "getting the job done" because they are either trivial or worse than the trivial solution. It is not reasonable to deem the search algorithm as successful when it settles on one of these solutions.

Comparing the probability of success

In order to determine the probability of success for the random search algorithm, given the empirical probability distribution for a single query, we use the following technique. Let Q_i for $i \in (1..2000000)$ be a random variable: the cost of a randomly generated solution. Let $Y = \min_{i=1}^{2000000} Q_i$ be the result of choosing the best solution of the 2,000,000 queries. 2,000,000 queries are performed because that is the number of queries performed by a run of the genetic algorithm. The probability distribution function of Y is [32]

$$F_Y(y) = \Pr[Y \le y] = 1 - \Pr[Y > y] = 1 - \Pr[Q_1 > y \text{ and } Q_2 > y \text{ and } \ldots \text{ and } Q_{2000000} > y]. \quad (7)$$

Since the various Q_i are independent and identically distributed,

$$F_Y(y) = 1 - \prod_{i-1}^{2000000} \left(1 - F_Q(y)\right) = 1 - \left(1 - F_Q(y)\right)^{2000000}. \quad (8)$$

Since all costs are rounded down the nearest integer, we define the random variable $Z = \lfloor X$.

$$f_Z(z) = F_Y(z+1) - F_Y(z) \quad (9)$$

Thus we can determine the distribution of Z, the integral cost of the solution produced by a search algorithm making 2,000,000 random queries.

Determining sources of active information

Most search algorithms are parameterized. The parameters in genetic algorithm design are nearly always tuned by the programmer using some variation of trial and error. Typically, only the final successful result is reported in the literature, while program failures during the design of a successful genetic algorithm are generally not reported. These failures, however, can provide information about the character of the program's goal as learned by the programmer during the design of the program. Knowledge gained through failures themselves are part of the search and are included in the search by the programmer. Active information is therefore provided to the search. Tell-tale signs of this tuning are often evident in the program. Such is the case with Thomas's code.

Determining the baseline. Before we can measure the active information available from the different search algorithms, we need to establish the underlying difficulty of the problem. If we were to just make a random guess at a solution for the problem under consideration, what is the probability that guess would reach the target? That defines the probability of success, p, for the baseline search algorithm in Equation 6. But that probability depends on which threshold is being considered. We will consider the probabilities of the three target thresholds: optimal, good, and adequate, as defined in the previous section.

Baseline for optimal solutions. How many optimal solutions exist? The method of encoding used by the genetic algorithm results in the same solution being encoded in multiple ways. By calculating the number of duplicates of the optimal solution we can estimate the probability of finding the optimal solution. The optimal solution has three interchanges. This means that the fourth interchange can take any value with 10^6 possible equivalent variations, since the fourth interchange's coordinates are now junk DNA. The fourth interchange would have eight possible connections, all of which are ignored, producing 2^8 equivalent variations. Additionally, the three interchanges can be placed into the representation in any order, but that order does not change the cost of the solution. There are 3! = 6 possible such arrangements.

There are four interchange solutions that do not connect anything to the fourth interchange and thus have equivalent cost. The actual coordinates of the fourth interchange do not matter, which gives 10^6 equivalent variations. The order of interchanges and which interchange is not used do not matter, giving 4! = 24 possible variations.

Other solutions of equal fitness may exist; however, this establishes a lower bound on the probability of guessing the correct solution. Consequently,

$$|T| \ge 6 * 10^6 * 2^8 + 10^6 * 24 \approx 1.56 * 10^9 \quad (10)$$

where T is the set of all optimal solutions. The total number of solutions is

$$|\Omega| = 5 * (1000^2)^4 \, 2^{\binom{9}{2}} \approx 3.44 * 10^{35}. \quad (11)$$

We can thus bound the probability of randomly selecting an optimal value

$$\frac{|T|}{|\Omega|} \ge 4.54 * 10^{-27}. \quad (12)$$

Baseline for good and adequate thresholds. We empirically estimated the probability of both the good and adequate thresholds by running Monte Carlo experiments that tested random solutions. 11,228,106,000 random queries were performed to obtain an empirical estimation of the probability of finding the target. The probabilities for finding the good and adequate targets were 1.7812e-09 and 1.9905e-07, respectively. The empirically determined probability for finding the optimal target was zero, because none of the random queries was successful in finding the target.

Effect of repeated queries. Active information can be obtained by many random queries. In fact we expect about $\log_2 Q$ bits

of active information from performing Q queries [9]. Thus for about two million queries we expect to get approximately 21 bits of active information. The empirically determined bits of active information obtained by the random queries is 20.93 and 20.65 for the good and adequate targets, respectively. This agrees with the theoretical prediction for two million random queries. The empirically determined bits of active information for the optimal target were negative infinity (-∞), because we were unable to find the optimal target by random queries.[1]

Effect of count of interchanges. The first element of the genome is the number of interchanges (Fig. 2). However, the actual number of interchanges used in the solution may be less than the number specified in the genome. Consider the example illustrated in Figure 5 of an interchange unconnected to any of the cities. Because no highways are built to the interchange, no additional cost is contributed to the solution. The solution is indistinguishable from the equivalent solution without that interchange. As a result, we can say that this solution uses no interchanges despite including an interchange in the genome.

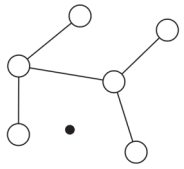

Figure 5: A road network with an additional unused interchange. **doi:**10.5048/BIO-C.2012.1.f5

Including the interchange count in the genome is therefore not strictly necessary. Instead, the interchanges can always be present and simply ignored if they are not connected to anything. The significance of including the interchange count is that it shifts the distribution of solutions. There are five possible interchange counts, {0,1,2,3,4}. With the count element, therefore, one fifth of all possible genomes contain zero interchanges. Without that count element, 2^{-26} of all possible genomes contain no interchanges.[2] Including the count element pushes the distribution towards solutions with fewer interchanges. Not including the count element pushes the distribution towards more interchanges.

The code restricts the minimum number of interchanges to be two. That is, even though the genome theoretically allows for a specification of zero or one interchanges, this is prevented by the initialization and mutation code.

"The precise code snippet where this frontloading is being performed" from Thomas's Fortran version of the program is shown below. It ensures that there are at least two interchanges (Thomas calls them variable points) during the initialization of the population [21] :

```
NPV = INT(RNDVAL*FLOAT(NVMX-1))+2 ↩
 ↪ ! MINIMUM 2 VARIABLE POINTS
```

[1] A value of negative infinity for the active information in the search for the optimal target indicates that success is too rare to determine exactly how rare it is.

[2] Given the initial 5 nodes, there are $\binom{5}{2}$ = 10 possible connections. Given all 9 nodes there are $\binom{9}{2}$ = 36 possible connections. This means that there are 26 connections involving the additional four nodes. In order for none of the additional nodes to exist, all of these connections must be off, giving a probability of 2^{-26}.

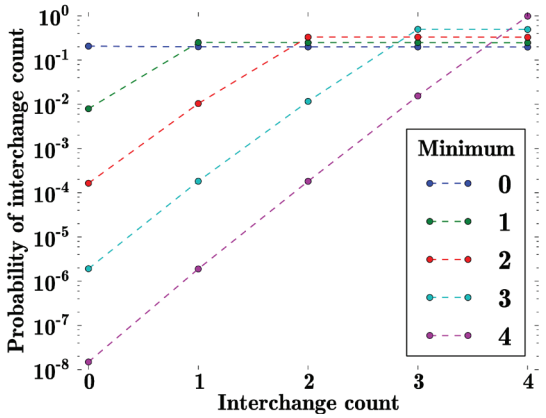

Figure 6: Probability of different interchange counts for a random query versus number of minimum interchanges. Each line represents a different minimum number of interchanges in the genome and gives the distribution of actually used interchanges. The data is discrete, with dashed lines added for clarity of presentation. **doi:**10.5048/BIO-C.2012.1.f6

Other code ensures that mutations maintain at least two interchanges.

```
NEW = INT(URAND(SEED)*FLOAT(NVMX-1))+2 ↩
 ↪ ! MINIMUM 2 VARIABLE POINTS
```

We can analytically determine the distribution of actually-used interchanges given possible choices of a minimum interchange count. Thomas's algorithm requires at least two interchanges in every genome, even though it is possible that some of these are not connected to anything. In the next paragraph we analytically derive the distribution of used interchanges for varying minimum interchange counts.

Figure 6 shows the distribution over different actual interchange counts varying with a minimum interchange count enforced in the simulation. It gives the probability of each number of interchanges given a specific minimum value. Given a minimum of zero, all different interchange counts have almost the same probability. However, given a minimum interchange count of four, which is equivalent to not having the interchange count element, obtaining a solution with less than four interchanges is unlikely. Each choice of a minimum interchange count produces a different distribution. As the

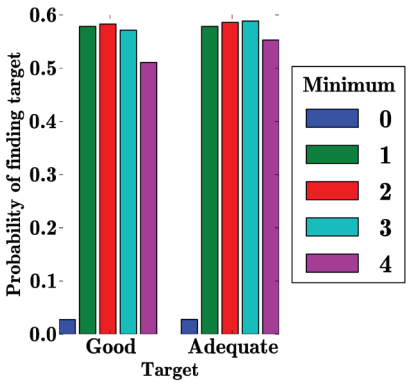

Figure 7: Effect of interchange number on distribution of solution costs. **doi:**10.5048/BIO-C.2012.1.f7

Table 1: Effect of the number of minimum interchanges on active information*

Minimum Interchanges	Optimal	Good	Adequate
0	-∞	23.88	17.08
1	-∞	28.27	21.47
2	-∞	28.29	21.49
3	84.83	28.26	21.50
4	85.62	28.09	21.41

* Active information is given in bits.

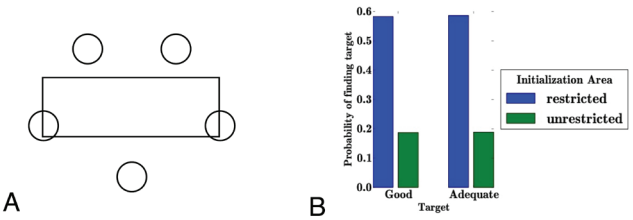

Figure 8: Effect of restricting the initialization area on the distribution of solution costs. The area under restriction is shown in (A), while (B) illustrates the effect on performance of the genetic algorithm with and without the restricted area. **doi:**10.5048/BIO-C.2012.1.f8

minimum is increased, all counts below the minimum become very improbable but never reach zero. Thomas's algorithm uses an interchange count of two, which de-emphasizes solutions with zero or one interchange, while still emphasizing solutions with two or three interchanges.

Figure 7 shows the probability of finding the targets given different minimum interchange counts. We see that a minimum interchange count of two, which is what the algorithm uses, seems to be the best choice. In contrast, placing no restriction on the number of additional interchanges results in a significant decrease of the probability of the finding the targets. Table 1 shows the active information extracted by these different versions of the algorithm. Notice that the active information to find the "Adequate" target using the unrestricted genotype is less than the 20 bits extracted by repeated random queries. It is easier to find an "Adequate" solution by random queries than with the unrestricted form of the genetic algorithm. The algorithm still outperforms the random queries for the "Good" solution. Thomas's original algorithm is restricted to a minimum of two interchanges and extracts over 28 bits of active information. Without the restriction on the number of interchanges, the algorithm extracts less than 24 bits of information. The restriction added four bits of active information to Thomas's algorithm. It is clear that tuning the minimum number of interchanges helped find a better solution.

Effect of restricted initialization. The genome allows the X and Y coordinates to lie anywhere in the range {0,1,2,…,999}. However, when the initial population is generated, the interchanges are restricted to being inside a smaller area. The following code is responsible for that restriction. [21]

```
RNDVAL = URAND(SEED)
XPP = 200+INT(RNDVAL*600.) ! X-LOCATION
XP(J+NFIX) = XPP
RNDVAL = URAND(SEED)
YPP = 400+INT(RNDVAL*200.) ! X-LOCATION
YP(J+NFIX) = YPP
```

The x-coordinate is restricted to be within the range 200-800 whereas the y-coordinate is restricted to be in the range 400-600. Figure 8A shows the area to which the initially selected interchanges are restricted. They are confined to the central area where we would expect useful interchanges to be positioned. Figure 8B shows how the performance of the genetic algorithm is affected by this change. Table 2 shows the active information obtained for each version of the algorithm. As with the previous example, the performance of searches for "Good" and "Adequate" targets have declined. The "Adequate" target is again less likely to be found with this genetic algorithm than a random query algorithm. It is clear that the restriction of the initial population to this area was a helpful decision.

Effect of selection skew. As noted, it is easy to obtain a solution with a score of 1246. The optimal solution has a cost of 1212. Consequently, there is a very small range of possible costs that are of interest, especially when contrasted with the range of possible costs, 0 - 100000. As a result, any typical method of selection is going to have difficulty in differentiating between the different scores.

Thomas normalizes the costs of the solution in the population as follows:

$$A_i = \frac{C_{max} - C_i}{C_{max} - C_{min}} \tag{13}$$

where C_i is the cost of solution i, C_{max} is the solution with the largest cost and C_{min} is the solution with lowest cost. Thomas defines the probability of population member i being selected as a parent for a reproductive event as

$$P_i = \left(\frac{A_i}{\sum_j A_j}\right)^{\frac{1}{s}} \tag{14}$$

where s is the skew parameter. Without the skew parameter there isn't enough selective pressure to effectively separate scores of close value. Figure 9 shows how the skew affects the probability of finding the various targets. With a skew of just 1.0, the probability of success is noticeably decreased. However, a relatively small skew value is sufficient to enable helpful selection. Thomas had to tweak his algorithm to have sufficient selective pressure, so he used $s = 1.5$.

Effect of mutations. It is possible for an algorithm to use prior knowledge in the design of mutations. However, Figure 10 shows Thomas's algorithm not to be very dependent on mutations. Turning off mutations completely has a noticeable but not dramatic effect. However, varying the rate of mutation does not have a large effect either way. Consequently, this algorithm

Table 2: Effect of restricted initiation on active information*

	Optimal	Good	Adequate
restricted	-∞	28.29	21.49
unrestricted	-∞	26.64	19.85

* Active information is given in bits.

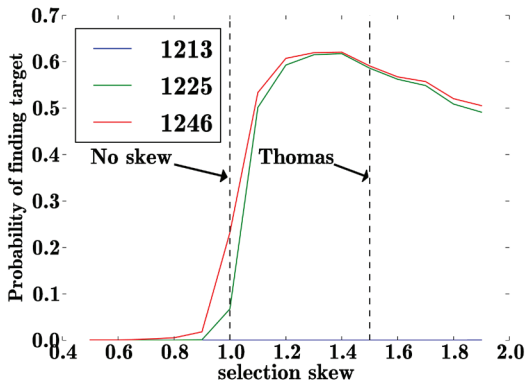

Figure 9: Effect of skew on the probability of finding the targets. **doi:**10.5048/BIO-C.2012.1.f9

does not depend extensively on the mutations as a source of active information.

Effect of crossover. While the algorithm is not very susceptible to mutations, it does depend on crossover. Crossover mixes two different genomes together to form the final genome. We contrast the performance of several methods of crossover:

- Single-point

  This is the method of crossover used in the Thomas's algorithm. A single point in the genome is chosen. The child receives a genome constructed of one parent before that point and another parent after that point (Fig. 11A).

- Two-point

  Two points are selected. The child receives the genome of the first parent except between the two points where the second parent's DNA is taken (Fig. 11B).

- Uniform

  For each letter, the parent from which to copy the letter is chosen randomly. This results in a thorough mixing of the parent's genomes (Fig. 11C).

- None

  No crossover is used; instead one parent is used without modification.

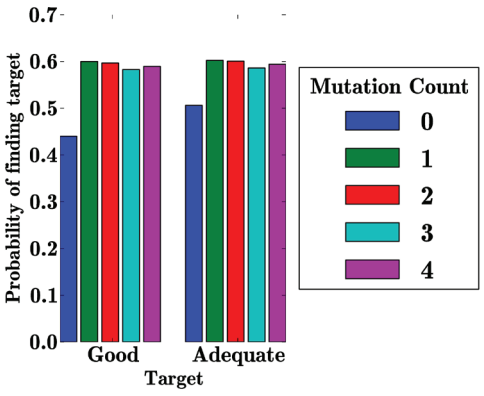

Figure 10: Effect of mutation number on distribution of solution costs. **doi:**10.5048/BIO-C.2012.1.f10

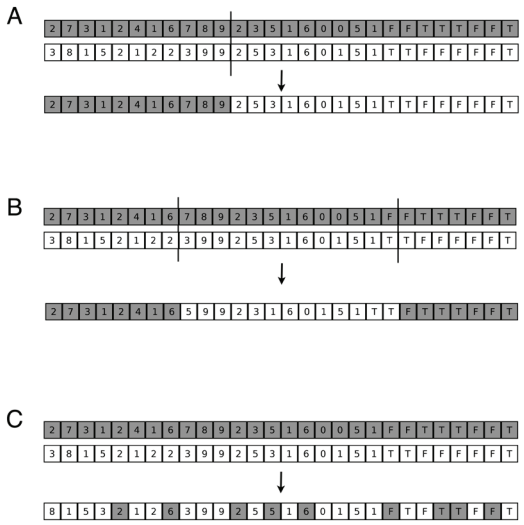

Figure 11: Crossover methods. Illustrated are (A) single-point crossover, (B) two-point crossover, and (C) uniform crossover methods. **doi:**10.5048/BIO-C.2012.1.f11

Table 3 shows the active information extracted by each form of crossover. Figure 12A shows the algorithm performance. It is clear the crossover method affects the algorithm performance. Introducing a single-point crossover is very helpful; however, increasing the amount of crossover does not improve performance further. Rather, performance decreases with increasing crossover. Some mixing of the genomes is helpful, but increased mixing degrades performance.

In addition, not all crossover points are equal. When the single-point crossover produces a child, it picks a point of division. Everything to the left of that point is taken from the "mother" and everything to the right is taken from the "father." But some points of division are more useful than others. As shown in Figure 2, the genome is divided into three sections that are analogous to genes. The first gene specifies the number of interchanges. The second gene specifies the coordinates of all the interchanges. The third gene specifies which nodes are connected to each other. By restricting the location of the crossover point, we can control which genes are mixed and which are copied intact. To analyze the effects we ran simulations where the crossover point is restricted to different areas. These areas are depicted in Figure 13 and described below:

- Case 1

  The crossover point is in the interchange count or coordinates. This preserves the connections, and produces a mix of the interchange coordinates.

Table 3: Active Information* for Differing Crossover Styles

Method	Optimal	Good	Adequate
Single-point	-∞	28.29	21.49
Two-point	-∞	28.08	21.31
Uniform	-∞	26.93	20.18
None	77.66	26.69	20.16

* Active information is given in bits.

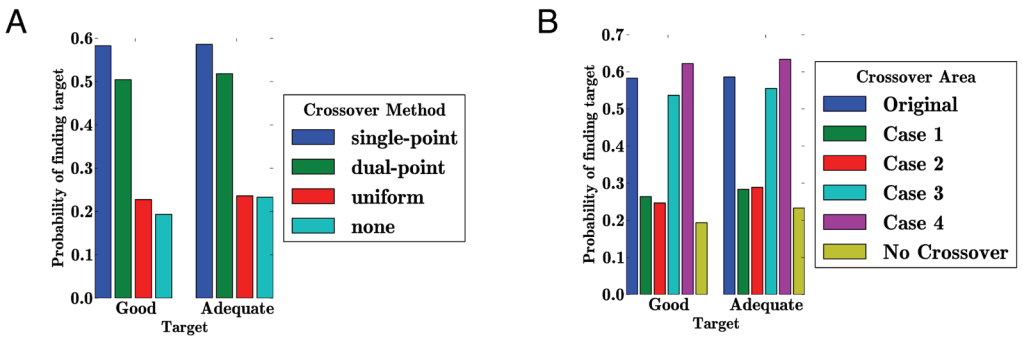

Figure 12: The effect of crossover on the distribution of solution costs. (A) shows the effect of crossover area, and (B) the effect of crossover method. **doi:**10.5048/BIO-C.2012.1.f12

- Case 2

  The crossover point is fixed between the location and the connections. This preserves both the coordinates and connections, but takes each from a different parent.

- Case 3

  The crossover point is in the connections. This preserves the coordinates and produces a mix of the connections.

- Case 4

  This is two point-crossover, with the first point fixed between the coordinates and connections. The second point is somewhere in the connections. Like case 3, this preserves the coordinates and mixes the connections. However, the connections are more thoroughly mixed.

Figure 12B shows the results. Cases 1 and 2, which did not mix the connections, performed relatively poorly. In contrast, cases 3 and 4, which mixed the connections but not the coordinates, performed well. In fact, performing additional mixing provided better performance. This suggests that crossover works well primarily because it combines different connection maps. The one section or gene seems to work very well with crossover while the other sections do not.

Crossover is much more helpful when it results in recombining connection maps rather than recombining interchange coordinates. It does help in the case of being applied to the coordinates, but not nearly to the same degree as it does when applied to the connections. There is something unique about the connection map that makes crossover so successful. We suspect that it exploits some of the same properties used in Prim's algorithm.

Crossover works well because of the way the genome is structured. The genome is built using a structure consisting of a list of locations as well as a connection map indicating which cities and interchanges are connected. However, the ways in which the genome could be structured are limited only by the developer's creativity. A number of other possibilities also exist:

- Every city and interchange could have a list of other cities/interchanges to which it is connected.
- The genome could be a sequence of road segments specified by start and stop points
- The genome could have been represented as a bitmap where each bit indicates whether or not an integral coordinate was on a road.

None of these methods were chosen, and none of them would seem to be useful representations. But they are rejected, not because something is intrinsically wrong with them, but because the developer uses his knowledge of the problem to come up with the best way to represent the solution. The programmer chooses the representation that he believes will be effective and part of that is choosing a representation that works well with crossover.

The crossover and genome structure work together. As such, their selection by the developer constitutes the use of prior knowledge about the problem in order to help identify better solutions.

C++ Algorithm changes

Thomas published a C++ version of his algorithm [22] after posting his original description. Our focus has been on the Fortran version of the algorithm because that it where the most detailed results were presented. For the most part, the C++ algorithm works the same as the Fortran algorithm but some differences should be noted.

- The minimum interchange count system has been modified.

  The initialization restricts the interchange used count to always be at the maximum. [22]

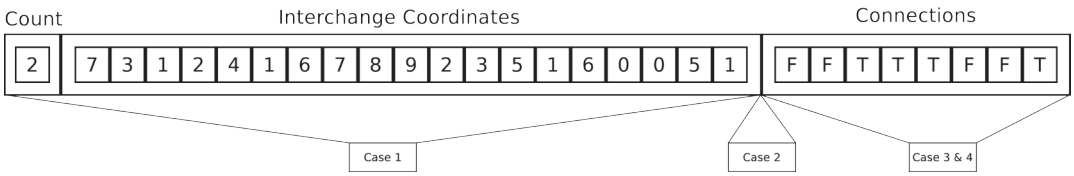

Figure 13: Genome structure depicting the different possible crossover zones. **doi:**10.5048/BIO-C.2012.1.f13

```
x = (double)rand() / (double)RAND_MAX;
num = (int)((double)(m_varbnodes*x);
num = m_varbnodes; // over-ride!!!
```

The claim that no design was involved in the production of this algorithm is very hard to maintain given this section of code. The code picks a random count for the number of interchanges; however, immediately afterwards it throws away the randomly calculated value and replaces it with the maximum possible, in this case, 4. The code is marked with the comment "override!!!," indicating that this was the intent of Thomas. It is the equivalent of saying "go east" and a moment later changing your mind and saying "go west." The most likely occurrence is that Thomas was unhappy with the initial performance of his algorithm and thus had to tweak it.

Additionally, the probability of mutating the interchange count is very low.

```
if (x < 0.0001) ↩
  ↪ // mutate nodes area - Rare...
{
  y = (double)rand() / (double)RAND MAX; ↩
      ↪// uniform random
  num = (int)((double)(m_varbnodes+1)*y);
```

While not outright forbidding genomes with lower interchange counts, this causes the algorithm to prefer interchanges with the maximum count. The only way to decrease this count is to have a relatively rare mutation occur. The result is that the system is biased towards solutions with more interchanges.

- The restricted area initialization was not included.
- The skew remains the same.
- Crossover remains the same.

DISCUSSION

Irreducible complexity

Thomas claims to have debunked irreducible complexity. His claim is based on the idea that, given any of the solutions produced by the algorithm, removing one of the interchanges or one of the highway connections would result in a disconnected map. That is, because the algorithm builds the minimum amount of highway between the cities, there is no redundancy in the system. Removing anything will leave the drivers in at least one of the cities without a road to a destination city. As such, he argues that his genetic algorithm can build irreducible complexity, something that an evolutionary process is not supposed to be able to do.

Behe defined irreducible complexity as:

> a single system composed of several well-matched, interacting parts that contribute to the basic function, wherein the removal of any one of the parts causes the system to effectively cease functioning [33]

We must therefore ask whether or not the cities and interconnecting highways that the algorithm builds can be considered well-matched interacting parts. Behe has in mind a very complex system, one that chance, acting alone without selection, cannot reasonably be supposed to have produced. His argument is directed at the abilities of selection because the assumption is that chance alone is easily ruled out. That is a reasonable assumption for the biological systems that Behe considers.

However, this assumption does not hold in the case of Steiner trees. As has been demonstrated, producing a solution that is connected can be trivially done. This algorithm's success derives from adapting a working solution into a better one. But irreducible complexity is concerned with the difficulty of producing a working solution at all. If producing a working solution by random chance is trivial, then irreducible complexity simply does not apply.

Misunderstanding of prior statements

Thomas's understanding of the intelligent design position is that the product of genetic algorithms has been "snuck" into the algorithm [19]:

> They claim that GAs cannot generate true novelty and that all such "answers" are surreptitiously introduced into the program via the algorithm's fitness testing functions.

Thomas defends this understanding by quoting one of us (Dembski):

> And nevertheless, it remains the case that no genetic algorithm or evolutionary computation has designed a complex, multipart, functionally integrated, irreducibly complex system without stacking the deck by incorporating the very solution that was supposed to be attained from scratch. [34, p. 58]

However, this quotation is not attempting to argue that all genetic algorithms succeed by incorporating the solution. Rather, the claim is specifically restricted to systems which are "complex, multipart, functionally integrated, and irreducibly complex." The above quote makes the argument that genetic algorithms have not developed any systems that are remotely comparable to the complexity that we find in biology, not that all targets must be embedded in the algorithm to be found.

The chapter on genetic algorithms in *No Free Lunch: Why Specified Complexity Cannot Be Purchased without Intelligence* [10] deals with this subject. The discussion is not restricted to genetic algorithms with a pre-specified target. Much of the text discusses such simulations, but only because the most well-known genetic algorithms used to defend Darwinian evolution contain the target in just such a way. Other genetic algorithms, such as one that evolves antennas, or another that evolves strategies for playing checkers, are discussed. Even Elsberry and Shallit, both critics of intelligent design, point this out:

> Dembski considers a number of genetic algorithms: variants on Dawkin's *METHINKS IT IS LIKE A WEASEL* example, an evolution simulation of Thomas

> Schneider, an algorithm of Altshuler & Linden for the design of antennas, and evolutionary programming approach to checkers-playing by Chellapilla & Fogel. [5]

The following quotation from *No Free Lunch: Why Specified Complexity Cannot Be Purchased without Intelligence* [10] makes it clear that not all processes find answers hidden within the algorithm. In many cases the investigator inserts complex specified information (CSI) by the informed choices he makes.

> In no way do SELEX, ribozyme engineering, or similar experimental techniques falsify the Law of Conservation of Information or circumvent the No Free Lunch theorems. In SELEX experiments large pools of randomized RNA molecules are formed by intelligent synthesis and not by chance—there is no natural route to RNA. These molecules are then sifted chemically by various means for catalytic function. What's more, the catalytic function is specified by the investigator. Those molecules showing some activity are isolated and become templates for the next round of selection. And so on, round after round. At every step in both SELEX and ribozyme (catalytic RNA) engineering experiments generally, the investigator is carefully arranging the outcome, even if he or she does not know the specific sequence that will emerge. It is simply irrelevant that the investigator is ignorant of the identity and structure of the evolved ribozyme and must determine it after the experiment is over. The investigator first had to specify a precise catalytic function, next had to specify a fitness measure gauging degree of catalytic function for a given biopolymer, and finally had to run an experiment optimizing the fitness measure. Only then does the investigator obtain a biopolymer exhibiting the catalytic function of interest. In all such experiments the investigator is inserting CSI right and left, most notably in specifying the fitness measure that gauges degree of catalytic function. Once it is clear what to look for, following the information trail in such experiments is straightforward. [10, pp. 220-221]

Dembski went on to apply this logic generally, including to genetic algorithms.

A quotation from Meyer (2004) has also been used by Thomas [19] to support the contention that intelligent design proponents claim that all genetic algorithms reproduce a hidden solution:

> These programs only succeed by the illicit expedient of providing the computer with a "target sequence" and then treating relatively greater proximity to future function (i.e., the target sequence), not actual present function, as a selection criterion. [35]

However, in context "these programs" does not refer to the entire class of genetic algorithms, but rather to two specific examples for which this is a valid critique.

Moreover, the source continues:

> As Berlinski (2000) has argued, genetic algorithms need something akin to a "forward looking memory" in order to succeed. Yet such foresighted selection has no analogue in nature. In biology, where differential survival depends upon maintaining function, selection cannot occur before new functional sequences arise. Natural selection lacks foresight. [35]

The target sequence is merely the method that the algorithms under consideration use to provide foresighted selection. The objection being raised to genetic algorithms is that they simulate foresight, which natural selection does not have. It is not the presence of the target, but rather how it is used that is at issue.

Thomas has misunderstood the intelligent design position. Perhaps it has not been explained with sufficient clarity. We do not argue that every genetic algorithm contains the final output encoded within it. A quick survey of genetic algorithms shows that they routinely produce solutions which cannot have been encoded in them. Thus the argument is a straw man; we do not take that position.

Summary

Thomas has proposed a genetic algorithm used to solve the Steiner tree problem as proof that the Neo-Darwinian account of evolution explains biological complexity and that intelligent design has been roundly refuted. But this claim is based on a misunderstanding of what the intelligent design community has said. The claim is not that the solution found by a genetic algorithm is hidden inside the algorithm, but rather that the algorithm contains the necessary information sources in order to find a low probability target. This is because in order to develop a successful genetic algorithm, the developer makes many decisions based on knowledge of the problem he is attempting to solve. As such, information is derived from this prior knowledge of the search problem. Tuning a genetic algorithm is another source of active information: it is common simply because it works. The reason it works is because it exploits the developer's own knowledge of the problem to be solved.

Various other genetic algorithms that solve Steiner tree problems clearly and openly make use of theoretical insights into the Steiner tree problem to assist in the search. Thomas indicates that he does not make use of such knowledge, but this is not the case. The distribution of the number of interchanges has been tweaked so that the algorithm focuses on solutions with the correct number of interchanges. The initial population is generated such that the interchanges are located where they are more likely to be useful. The strength of selection has been increased by a parameter to counteract the small range of interesting fitness values. The crossover method and genome structure assist in finding the solution. All of these introduce active information; they are exploiting prior knowledge about the problem. Thomas's algorithm does make use of far less prior knowledge than the other Steiner tree algorithms and as a result is only able to solve problems an order of magnitude smaller.

As with our prior work [9,11,12,18], we have shown that the search algorithm proposed as an example of the power of natural selection to generate information from scratch in fact demonstrates the abilities of humans to devise genetic algorithms that draw on existing information. Thomas has failed to demonstrate the abilities of natural selection left to itself. In order to demonstrate the abilities of natural selection, it would be necessary to avoid making any decisions in the development of the genetic algorithm that deliberately assist in finding the solution. Only a teleological process guided by some form of intelligence can function in this way. Insofar as simulations of evolution make use of prior knowledge, they are not simulations of Darwinian evolution in any meaningful sense.

APPENDIX: MINIMUM INTERCHANGE MATH

Let C be the number of fixed cities. Let N be the total number of live interchanges in a genome; that is the count specified as the first element of the genome. Let D_f be a discrete random event which occurs when no interchange in the set f has a connection to it and thus contributes no cost to a particular solution. There are $\binom{N}{2}$ possible connections. Excluding connections to the cities $\in f$, we obtain $\binom{N-|f|}{2}$ possible solutions. There are thus $\binom{N}{2} - \binom{N-|f|}{2}$ connections involving the cities in f. In order for these interchanges to have no connections all must be off, denoted by an "F" in the genome rather than on denoted by a "T". This gives us

$$\Pr[D_f] = 2^{-\binom{N}{2}+\binom{N-|f|}{2}}. \tag{A1}$$

Let U be the discrete random variable representing the number of interchanges which are not connected to any city or interchange. There are $\binom{N}{U}$ possible sets of such cities. We can calculate the cumulative density function

$$\begin{aligned} F_U(u) &= \Pr[U < u] = 1 - \Pr[U \geq u] \\ &= 1 - \begin{cases} 0 \text{ if } U > N \\ 1 \text{ if } U < 0 \\ \binom{N}{u} 2^{-\binom{N+C}{2}+\binom{N+C-u}{2}} \text{ otherwise} \end{cases} \end{aligned} \tag{A2}$$

Since this is a discrete random variable we can say that

$$\Pr[U = u] = F_U(u) - F_U(u-1). \tag{A3}$$

Let X be the actual number of used nodes in the genome

$$X = N - U \tag{A4}$$

Let m be the lower bound on N which is enforced by the code. We can thus calculate the probability

$$\Pr[X = x] = \sum_n \Pr[N = n] \Pr[X = x | N = n] \tag{A5}$$

by the law of total probability.

We can easily express the probability distribution of N

$$\Pr[N = n] = \begin{cases} 0 \; if \; n > m \\ \dfrac{1}{5-m} if \; n \leq m \end{cases}. \tag{A6}$$

Using Equation A4 and Equation A3,

$$\begin{aligned} \Pr[X = x | N = n] &= \Pr[U = N - X | N = n] \\ &= F_U(U) - F_U(U-1) \end{aligned} \tag{A7}$$

Acknowledgements

We would like to thank the anonymous reviewers for their great attention to detail and valuable comments. Thanks to their comments, this is a much better paper than the one they reviewed.

1. Dawkins R (1996) The blind watchmaker: Why the evidence of evolution reveals a universe without design. Norton (New York).
2. Marczyk A (2004). Genetic Algorithms and Evolutionary Computation. http://www.talkorigins.org/faqs/genalg/genalg.html. Last accessed September 19, 2011.
3. Schneider TD (2000) Evolution of biological information. Nucleic Acids Res 28: 2794–2799. **doi:**10.1093/nar/28.14.2794.
4. Lenski RE, Ofria C, Pennock RT, Adami C (2003) The evolutionary origin of complex features. Nature 423:139–44. **doi:**10.1038/nature01568.
5. Elsberry W, Shallit J (2011) Information theory, evolutionary computation, and Dembski's "complex specified information." Synthese 178: 237-270. **doi:**10.1007/s11229-009-9542-8
6. Altshuler E, Linden DS (1997) Wire-antenna designs using genetic algorithms. IEEE Antennas and Propagation Magazine 39:33–43. **doi:**10.1109/74.584498.
7. Koza JR, Keane MA, Streeter MJ (2003) Evolving inventions. Sci Am 288:52–59. **doi:**10.1038/scientificamerican0203-52.
8. Thompson A (1997) An evolved circuit, intrinsic in silicon, entwined with physics. In: From Biology to Hardware: Lecture notes in Computer Science, Springer-Verlag (Berlin/HeidelBerg), pp 390-405. **doi:**10.1007/3-540-63173-9_61.
9. Dembski WA, Marks II RJ (2009) Conservation of information in search: Measuring the cost of success. IEEE T Syst Man Cy A 39: 1051-1061. **doi:**10.1109/TSMCA.2009.2025027.
10. Dembski WA (2002) No Free Lunch: Why Specified Complexity Cannot Be Purchased without Intelligence, Rowman & Littlefield (Lanham, MD).
11. Ewert W, Montañez G, Dembski WA, Marks II RJ (2010) Efficient per query information extraction from a Hamming oracle. 42nd Southeast Symp Syste: 290-297. **doi:**10.1109/SSST.2010.5442816.
12. Montañez G, Ewert W, Dembski W A, Marks II R J (2010) A vivisection of the **ev** computer organism: Identifying sources of active information. BIO-Complexity 2010(3):1–6. **doi:**10.5048/BIO-C.2010.3.
13. Shapiro J (2011) Evolution : A View from the 21st Century, FT Press Science (Upper Saddle River, NJ).
14. Wolpert D, Macready W (1997) No free lunch theorems for optimization. IEEE T Evolut Comput 1: 67-82. **doi:**10.1109/4235.585893.
15. Schaffer C (1994) A conservation law for generalization performance. In: Cohen WW and Hirsch H, eds. Proceedings of the Eleventh International Machine Learning Conference. Rutgers University (New Brunswick), pp 259-265.

16. Ho Y-C, Pepyne DL (2001) Simple explanation of the no free lunch theorem of optimization. IEEE Decis Contr P 5: 4409-4414. **doi:**10.1109/.2001.980896.
17. Christensen S, Oppacher F (2001) What can we learn from No Free Lunch? A first attempt to characterize the concept of a searchable function. In: Proceedings of the 2001 Genetic and Evolutionary Computation Conference. Morgan Kaufman (San Mateo). pp 1219–1226.
18. Ewert W, Dembski WA, Marks II RJ (2009) Evolutionary synthesis of nand logic: Dissecting a digital organism. IEEE Sys Man Cybern San Antonio Oct 11-14: 3047-3053. **doi:**10.1109/ICSMC.2009.5345941.
19. Thomas D (2010) War of the Weasels: An Evolutionary Algorithm Beats Intelligent Design. Skeptical Inquirer 43:42–46.
20. Thomas D (2006) Target? TARGET? We don't need no stinkin' Target! http://pandasthumb.org/archives/2006/07/target-target-w-1.html. Last accessed September 19, 2011.
21. Thomas D. FORTRAN for Genetic Algorithm. http://www.nmsr.org/genetic.htm. Last accessed September 19, 2011.
22. Thomas D (2006). Steiner Genetic Algorithm - C++ Code. http://pandasthumb.org/archives/2006/07/steiner-genetic.html. Last accessed September 17, 2011.
23. Crescenzi P, Kann V (1998). A compendium of NP optimization problems. http://www.nada.kth.se/~viggo/wwwcompendium/node78.html. Last accessed March 26, 2012.
24. Jesus M, Jesus S, Márquez A (2004) Steiner Trees Optimization using Genetic Algorithms. Technical report, Centro de Simulação e Cálculo.
25. Barreiros J (2003) An hierarchic genetic algorithm for computing (near) optimal Euclidean Steiner trees. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pp. 56–65.
26. Jones J, Harris Jr FC (1996). A Genetic Algorithm for the Steiner Minimal Tree Problem. In: Proceedings of ISCA's International Conference on Intelligent Systems.
27. Ding S, Ishii N (2000) An online genetic algorithm for dynamic Steiner tree problem. In: IEEE International Conference on Industrial Electronics, Control and Instrumentation 2:812–817. **doi:**10.1109/IECON.2000.972227.
28. Kapsalis A, Rayward-Smith V J, Smith G D (1993) Solving the graphical Steiner tree problem using genetic algorithms. J Oper Res Soc 44:397–406. **doi:**10.1038/sj/jors/0440408.
29. Rabkin M (2002) Efficient Use of Genetic Algorithms for the Minimal Steiner Tree and Arborescence Problems with Applications to VLSI Physical Design. In: Koza JR, ed. Genetic Algorithms and Genetic Programming at Stanford 2002. Stanford Bookstore, (Stanford, CA). pp 195–202.
30. Julstrom B (2002) A scalable genetic algorithm for the rectilinear Steiner problem. IEEE C Evol Computat 2:1169-1173. **doi:**10.1109/CEC.2002.1004408.
31. Cormen T (2001) Introduction to Algorithms. 2nd Ed. MIT Press (Cambridge, Mass).
32. Marks II R J (2009) Handbook of Fourier Analysis & Its Applications. Oxford University Press (Oxford, New York).
33. Behe M (1996) Darwin's Black Box : The Biochemical Challenge to Evolution, Free Press (New York).
34. Dembski W A (2005) Rebuttal to reports by opposing expert witnesses. http://www.designinference.com/documents/2005.09.Expert_Rebuttal_Dembski.pdf. Last accessed 3/26/2012.
35. Meyer S C (2004) The origin of biological information and the higher taxonomic categories. P Biol Soc Wash 117:213–239.