Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

Evolutionary Synthesis of Nand Logic: Dissecting a Digital Organism

Winston Ewert
Department of
Computer Science,
Baylor University,
Waco, Texas

William A. Dembski
Professor of Philosophy,
Southwestern Baptist
Theological Seminary
Fort Worth, Texas

Robert J. Marks II
Distinguished Professor of
Electrical & Computer Engineering,
Baylor University,
Waco, Texas

***Abstract*—According to *conservation of information* theorems, performance of an arbitrarily chosen search, on average, does no better than blind search. Domain expertise and prior knowledge about search space structure or target location is therefore essential in crafting the search algorithm. The effectiveness of a given algorithm can be measured by the *active information* introduced to the search. We illustrate this by identifying sources of active information in Avida, a software program designed to search for logic functions using nand gates. Avida uses *stair step active information* by rewarding logic functions using a smaller number of nands to construct functions requiring more. Removing stair steps deteriorates Avida's performance while removing deleterious instructions improves it. Some search algorithms use prior knowledge better than others. For the Avida digital organism, a simple evolutionary strategy generates the Avida target in far fewer instructions using only the prior knowledge available to Avida.**

***Index Terms*—conservation of information, active information, assisted search, evolutionary search, endogenous information, importance sampling, sea of gates, nand logic, no free lunch theorems.**

I. Introduction

Even moderately sized search problems require external assistance to be successful. The contribution of the external source to the search can be measured as *active information* [4], [5], [7]. Sources of active information include (a) an *oracle* that provides fitness information (or scores) for the search, (b) the *initialization*, properly chosen, can improve the probability of success, and (c) knowledge about the *search space structure*. If prior assumptions about the search are incorrect, the search can perform worse than a random search. Since every constructive choice of priors has a corresponding deleterious choice, the average performance over all searches is the same as a random search [3], [4], [5], [8], [15], [19], [20], [21], [22], [23], [27], [28], [29], [30]. This search property is dubbed *conservation of information* [4], [5], [7], [9], [23] as popularized by the *no free lunch theorem* [3], [8], [15], [30].

The open source Avida program [16] performs logic function synthesis using nand gates [11], [13], [25], [26], [32] and evolutionary search [18]. We show that, as a search algorithm, Avida generates active information from a number of knowledge sources provided by the programmer and, with respect to an evolutionary strategy, performs poorly with respect to other search strategies using the same prior knowledge.

A. Information Measures

To assess the performance of a search, we use the following information measures [4], [5], [7].

1) The *endogenous information* is a measure of the difficulty of a search and is given by
$$I_\Omega = -\log_2 p \qquad (1)$$
where p is a reference probability of a successful unassisted random search.
2) Let the probability of success of an assisted search under the same set of constraints be q. Denote the *exogenous information* of a search program as
$$I_S := -\log_2 q.$$
3) The difference between the endogenous and exogenous information is the *active information*.
$$I_+ := I_\Omega - I_S = -\log_2 \frac{p}{q}.$$

If the knowledge about the space is not accurate or is otherwise misleading, the active information can be negative.

For a computer program generated by an alphabet of instructions as is the case with Avida, a resource constraint must be imposed. Otherwise, unlimited time and computer resources allows an exhaustive search. Let $\gimel$ denote the instruction count and $\acute{\gimel}$ the maximum allowable number of instructions before abandoning the search. Under an instruction count constraint, $\Re = \{\gimel | \gimel \le \acute{\gimel}\}$ the *active information per instruction* (AIPI) is
$$I_\oplus := \mathbb{E}\left[\frac{I_+}{\gimel}\right] \qquad (2)$$
where $\mathbb{E}$ denotes expectation [17]. The AIPI can be estimated by averaging the active information per instruction over K trials of $\acute{\gimel}$ instructions or less. For the k^{th} simulation, there are two possibilities.

1) Success is achieved with $\gimel_k \le \acute{\gimel}$ instructions in which case the point estimate of the AIPI is $I_\Omega/\gimel_k$.
2) If a success is not achieved with $\gimel_k = \acute{\gimel}$ instructions, then the point estimates of the active information and the AIPI both have a value of zero.

Thus, with $\varsigma_k = 1$ for a success in $\acute{\gimel}$ or fewer instructions and $\varsigma_k = 0$ and $\gimel_k = \acute{\gimel}$ for a failure, we can estimate the AIPI as the harmonic average over K trials.

$$\begin{aligned} I_{\oplus} &\simeq \frac{1}{K}\sum_{k=1}^{K}\varsigma_k\left(\frac{I_\Omega}{\gimel_k}\right) \\ &= \frac{I_\Omega}{K}\sum_{\text{successes}}\frac{1}{\gimel_k}. \end{aligned} \tag{3}$$

The AIPI needs to be interpreted with the same caution as the average speed of an auto on a road trip. Instantaneous values can be significantly higher or lower than the average. We dub $I_\oplus/I_\Omega$ the normalized AIPI, or the NAIPI.

II. Evolutionary Search Using Nand Logic

The information measures of many search algorithms are beyond direct analytic evaluation and require empirical analysis using, for example, Monte Carlo simulation. One is evolutionary search for synthesis of logic functions.

The nand gate is one of two logic functions from which all other logic can be synthesized [14]. The other is the nor gate. Chips containing a *sea of gates* [6], [11], [13], all nand, are therefore capable of universal logic. This property allows evolutionary development of logic functions using the nand gate as the single logic component [1], [12], [25], [26], [32].

Avida [16], illustrated in Figure 1, performs logic synthesis using only the nand gate. The motivation behind Avida is not engineering design, but is rather to [16]

> "... show how complex [biological] functions can originate by random mutation and natural selection."

Avida generates an output equal to a logic combination of the inputs, X and Y. The logic operation under consideration that requires the most nand gates (five) is the XNOR. This is the ultimate goal of the search. The example of Avida in Figure 1 has performed the XNOR. The first bits X and Y are 1 and 0 and the XNOR of 1 and 0 is 0. This then is the first bit of the output string. The second bits of X and Y are both ones and the XNOR of two ones is 1. This is the second bit in the output. Continuing with subsequent bits, we say the machine in Figure 1 generates an XNOR if the bitwise XNOR's of X and Y is equal to the corresponding output bit. Since the XNOR is 1 if both input bits are the same and 0 if they are not, Avida refers to the XNOR as an EQU [16].

Avida uses a small alphabet of instructions (see Table I) to import the inputs, perform manipulation, and export the output. The instruction tape runs in a continuous loop. The number of entries can change during the search process. Each letter in the loop in Figure 1 corresponds to the lettered instructions in Table I. The highlighted instruction (q) `IO`, for example, outputs the target register, checks to see if any logic function in Table II has been performed, and reads the next input into the target register. Each of the registers in the Avida contains 32 bits although, as we will see, the number of bits in each register does not significantly impact the search. The inputs X and Y are assigned randomly, are read only, and forever remain fixed. There are read-write scratch pad registers in the organism, AX,

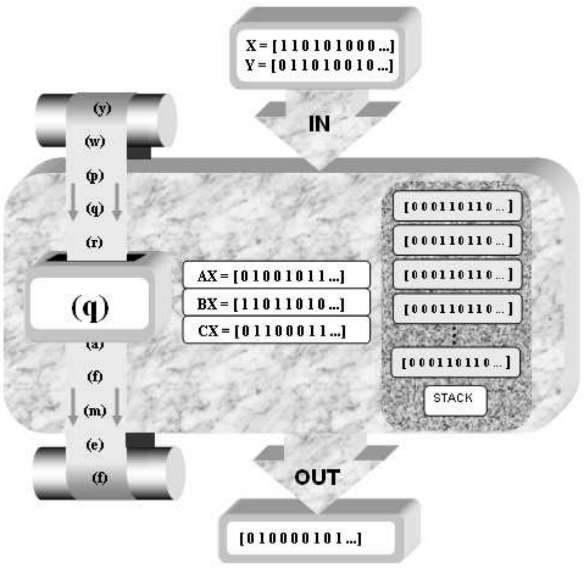

Fig. 1. The digital organism used by Avida, similar to that shown in the original Avida paper [16]. Generating a program from a fixed alphabet of 26 instructions (see Table I) to perform a specified function like XNOR (EQU) is similar to generating a specified word or phrase from the English alphabet somewhere within an alphabet of characters. The Avida simulation actually make use of three inputs, X, Y and Z, in a circular queue. Whenever an input is read by the program, it is removed from the queue and then placed at the back of the queue. As a result, three consecutive reads will read in three different inputs, whereas the fourth will read the first input in again. This property was not addressed in the original paper.

BX, and CX, on which to perform the operations. There is also a stack that can pop a stored 32 bit word into AX, BX and CX. An element from AX, BX and CX can likewise be pushed onto the top of the stack. The goal is to choose instructions from Table I so that the output, written by operation (q) `IO`, is the bitwise XNOR of the input registers X and Y. For most generations of the evolutionary search, Avida uses 3600 of the digital organisms in Figure 1.

A. Problem Difficulty

Like choosing letters from the alphabet to form a sequence of words that will pass a spell check, the goal of Avida is to search for a sequence of instructions that has meaning with respect to the logic functions in Table II. The question is - how difficult is it to generate the logic functions in Table II using the $N = 26$ instructions in Table I in a sequence of instructions?

1) Instruction Count: We want to evaluate the relative performance between Avida and competing algorithms. The most straightforward measure of computational cost is measurement of the actual CPU time of the process. CPU time, however, can differ considerably on different hardware. Other external factors, such as user interaction with the system, may also result in a non-accurate measure of the computational cost.

Another measure of the cost in search algorithms is a query count. In Avida, however, the computational demands of a

TABLE I

$N = 26$ INSTRUCTIONS FOR PERFORMING THE LOGIC FUNCTIONS IN TABLE II. THE DEFAULT TARGET REGISTER IS BX.

#	Operation	Description
(a)	`nop-A`	No-Operation. May modify the operation of the instruction before it.
(b)	`nop-B`	No-Operation. May modify the operation of the instruction before it.
(c)	`nop-C`	No-Operation. May modify the operation of the instruction before it.
(d)	`if-n-equal`	Compares two values, skips the next instruction if they are equal.
(e)	`if-less`	Compares two values, skips the next instruction if the first is less.
(f)	`push`	Puts a copy of the value in the target register onto the top of the stack.
(g)	`pop`	Removes a value from the top of the stack and places it into the target register.
(h)	`swap-stk`	Switches the stack between two available stacks
(i)	`swap`	Swaps the value of the target register with the next register
(j)	`shift-r`	Shift the bits of the value in the target register to the right. This is the same as dividing by two and rounding down.
(k)	`shift-l`	Shift the bits of the value in the target register to the left. This is the same as multiplying by two.
(l)	`inc`	Increments the value in the target register by one.
(m)	`dec`	Decrements the value in the target register by one.
(n)	`add`	Adds the values of the BX and CX register, placing the result in the target register.
(o)	`sub`	Subtract the value of CX from BX, placing the result in the target register.
(p)	`nand`	Bitwise-nands the values of BX and CX together, placing the result in the target register.
(q)	`IO`	Outputs the target register, checking if for any tasks completed. Reads the next input into the target register.
(r)	`h-alloc`	Allocates additional memory to be used for a daughter organism.
(s)	`h-divide`	Splits the daughter and the parent into seperate organisms.
(t)	`h-copy`	Copies a single instruction from the read head to the write head
(u)	`h-search`	Searches the genome for a pattern of nops, moving the flow-head to their location.
(v)	`mov-head`	Moves one of the other heads to the flow-head.
(w)	`jmp-head`	Causes one of the heads the jump forward by the value in the CX register.
(x)	`get-head`	Copies the position of one of the heads into the CX register.
(y)	`if-label`	Checks to see whether a certain pattern of nops has just been copied. If they have not, skips an instruction.
(z)	`set-flow`	Sets the position of the flow head to the value in target register.

TABLE II

NAND LOGIC ILLUSTRATED IN FIGURES 2 THROUGH 4. THE "&" DENOTES A LOGIC AND, THE "+" IS AN OR AND THE OVERLINE DENOTES COMPLEMENT. IN THE THREE "EITHER-OR" FUNCTIONS, A SUCCESS IS CLAIMED IF EITHER ONE OF THE TWO FUNCTIONS IS PERFORMED. THE NUMBER IN THE "NANDS" COLUMN IS THE LOG$_2$ OF THE "SCORE" COLUMN.

Logic	OUT	NANDS	Score
NOT	either $\overline{X}$ or $\overline{Y}$	1	2
NAND	$\overline{X\&Y}$	1	2
AND	X&Y	2	4
OR_N	either X+$\overline{Y}$ or $\overline{X}$+Y	2	4
OR	X+Y	3	8
AND_N	either X&$\overline{Y}$ or $\overline{X}$&Y	3	8
NOR	$\overline{X+Y}$	4	16
XOR	$X\oplus Y=(X\&\overline{Y})+(\overline{X}\&Y)$	4	16
EQU	$\overline{X\oplus Y}=(X\&Y)+(\overline{X}\&\overline{Y})$	5	32

query can vary widely.

A reasonable accounting metric for Avida is an instruction count. We define an instruction as the execution of any one of the entries in Table I.

2) A Single Avida Organism: Avida's performance can be evaluated using Monte Carlo simulation. Here are some useful definitions.

- *A program* is a list of 100 instructions. 85 are chosen randomly and 15 are specified.[1] The number of instructions in an executed program varies. The same instruction, for example, can be executed more than once.
- *A query* is a sequence of programs executed until either an EQU is found, or until

$$\gimel = 10.8 \text{ billion instructions} \tag{4}$$

are used. This number is roughly the number of instructions used by the Avida default.[2]
- A program or query is *diminished* if instructions (v) `mov-head` and (w) `jmp-head` in Table I are not used. They nearly always cause the failure of an EQU calculation.[3] A program or query including all of the instructions, including (v) `mov-head` and (w) `jmp-head`, is dubbed *full*.
- Let D denote *diminished* and F *full*. P_D and P_F denote diminished and full programs while Q_D and Q_F correspond to diminished and full queries. An S will denote success in finding an EQU, so $P_{S|F}$ and $Q_{S|F}$ denote successful full programs and queries, while $P_{S|D}$ and $Q_{S|D}$ are the diminished equivalents.

[1]These 15 instructions, native to the Avida digital organism, allow the process of replication in the evolutionary search.

[2]$\gimel$ = 100,000 updates × 3600 programs × 300 instructions (on average) per program per update [16].

[3]The instructions (v) `mov-head` and (w) `jmp-head` both manipulate various heads in the Avida CPU. There are three heads which can be manipulated by these instruction, the read-head, write-head, or the instruction pointer. The read and write heads are setup for the copying process at the beginning of an Avida program. If they are moved, the replication process will not work correctly. If the instruction pointer is changed the Avida program will jump to a different position in its program, in all probability causing it to loop in a particular portion of its program never reaching the copy loop.

3) An Estimate of the Endogenous Information of a Full Program: A total of $|Q_D| = 1420$ diminished queries were performed and a total of $|S| = 21$ EQU's resulted. The average number of instructions per diminished program, P_D, was

$$\mathbb{E}[\gimel \text{ per } P_D] \simeq 637.9 \text{ instructions} \tag{5}$$

so the total number of diminished programs simulated is

$$|P_D| = \frac{\acute{\gimel}\,|Q_D|}{E[\gimel \text{ per } P_D]} \simeq 2.40 \times 10^{10}$$

A separate run of full programs[4] gave

$$\mathbb{E}[\gimel \text{ per } P_F] \simeq 1972 \text{ instructions.} \tag{6}$$

The maximum number of instructions allowed by Avida was 2000.

Of the $N_F = 26^{85} = 1.87 \times 10^{120}$ possible full programs, there are over 10^{108} that compute an EQU. To show this, consider first the number of diminished programs. $N_D = 24^{85} = 2.08 \times 10^{117}$. The probability a diminished program will compute an EQU is about $\Pr[\text{EQU}|P_D] \simeq |S|/|P_D| = 8.73.5 \times 10^{-10}$. The number of diminished programs capable of computing EQU is therefore $|P_{S|D}| = N_D \times \Pr[\text{EQU}|P_D] \simeq 1.82 \times 10^{108}$. If we assume all programs containing (v) `mov-head` or (w) `jmp-head` will fail, this is also the number of full programs that will compute EQU, *i.e.* $|P_{S|F}| \simeq |P_{S|D}|$. The probability of choosing a successful program randomly from the set of full programs is thus

$$p = \Pr[P_{S|F}] \simeq \frac{|P_{S|D}|}{N_F} = 9.69 \times 10^{-13}. \tag{7}$$

The endogenous information of the difficulty of generating an EQU with 85 randomly selected instructions from Table I follows from (1) as

$$I_\Omega \simeq 40 \text{ bits}$$

4) Algorithm $\mathfrak{Q}$, Repeated Diminished Programs: Running a program a repeated number of times generates more information than a single run [5]. For the $|Q_D| = 1420$ diminished query simulations each to a maximum of $\acute{\gimel}$ instructions each, the NAIPI, calculated using (3), was

$$I_\oplus / I_\Omega \simeq 5.78 \times 10^{-12} \tag{8}$$

This is the (small) NAIPI obtained from (a) repeated queries and (b) using only diminished programs.

5) Stair Step Active Information in the NAND Search: Knowledge that higher level nand logic can be built on lower level nand logic can be used to introduce *stair step active information* into the search for the EQU. Stair step active information, discussed in detail elsewhere [5], is an important source of active information in Avida.

Examples of synthesizing the nine logic functions in Table II using only nand gates is well known and is illustrated in minimal form[5] in Figures 2 through 4. They are included here to show that the logic can build on itself. For example, Figure 2 logic function #3 shows the AND as simply a cascade connection of the NOT circuit #1 and the NAND function in #2. Higher up the list, the #9 XNOR (EQU) in Figure 4 is a simple cascade connection of the #8 XOR circuit with the #2 NAND. The #8 XOR, in turn, can be synthesized using other lower numbered logic circuits. Not all of the steps need be directly useful to the subsequent target.

In the nand gate synthesis, logic functions with fewer gates are rewarded as possible steps towards achieving the ultimate EQU target at the top of the stairs. The rewarding in Avida is done using the Score column in Table II. The $\log_2$ of the Score is equal to the minimum number of gates required to perform the logic function. If two or more logic functions are generated by a digital organism, the fitness of the organism is the product of the scores. Each logic function is counted only once in this tally. For example, if an organism generates three separate cases of the NOT function, the fitness rule only counts them as one occurrence.

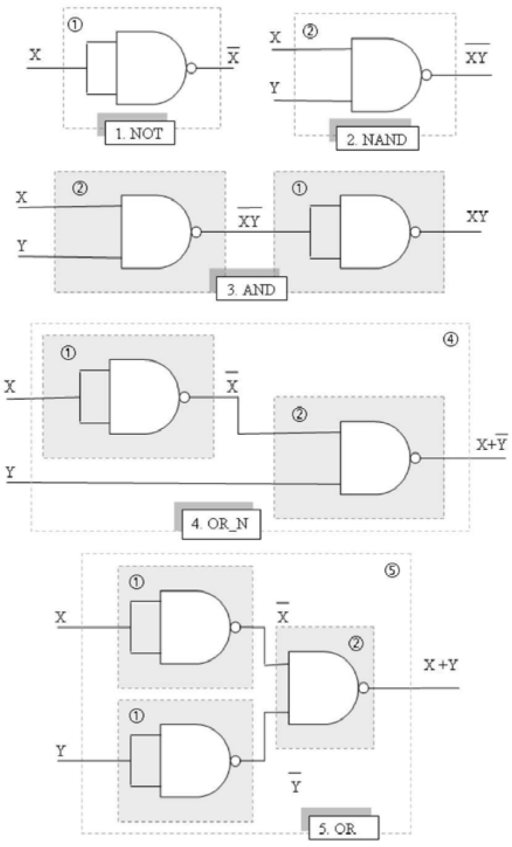

Fig. 2. Minimal NAND logic. Logic function is synthesized using the minimum possible number of NAND gates. Continued in Figure 3.

For all of the subsequent simulations, a resource bound of $\acute{\gimel}$ in (4) is imposed.

1) *Performance of Avida.* Using 3600 digital organisms of

[4] 10,272,633 full programs

[5] Figures 2 through 4 show configurations using the fewest possible number of nand gates. Other configurations are possible.

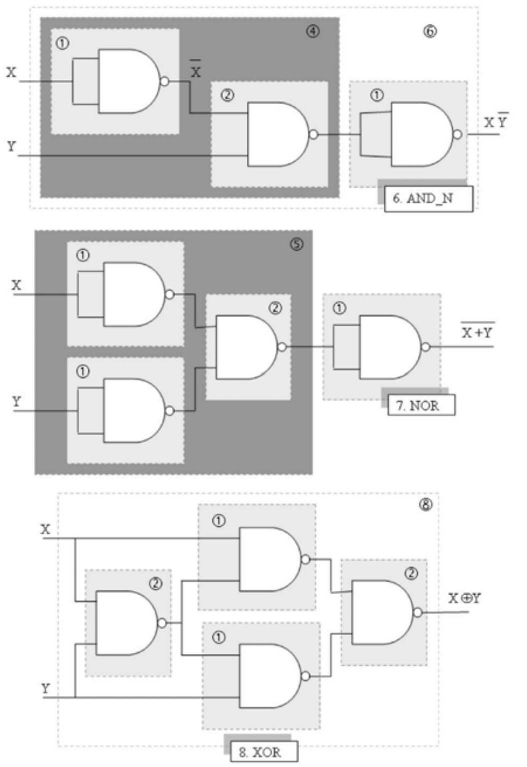

Fig. 3. NAND logic. Continued from Figure 2.

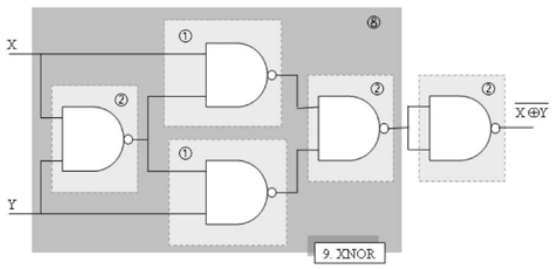

Fig. 4. NAND logic for the XNOR operation. Continued from Figure 3.

the type shown in Figure 1, Avida weighs the fitness of an organism from the Score given in Table II. A detailed description of other operations used in Avida, including mutation rates and replication properties, are available elsewhere [16].

- *Algorithm* $\mathcal{A}_1$. Results using Avida's default values, including all of the stair steps in Table II and all $N = 26$ instructions in Table II, are shown in the first row of Table III. Most of the trials resulted in a success. The NAIPI for $\mathcal{A}_1$ is $I_\oplus/I_\Omega = 1.9 \times 10^{-9}$. This value is due, in part, to active information introduced by the stair steps.
- *Algorithm* $\mathcal{A}_2$. As is shown in the second line in Table III, the NAIPI is reduced by excluding the XOR and NOR steps. Removing these steps therefore makes the search more difficult.[6]
- *Algorithms* $\mathcal{A}_3$ *and* $\mathcal{A}_4$. Taking away additional stair steps worsens the performance even more. In $\mathcal{A}_3$, the additional functions of OR and AND_N are removed resulting in an NAIPI of less than half when compared to using all of the stair steps. If, instead, AND and OR_N are removed, the NAIPI reduction is over two thirds. These stair steps therefore contribute significantly to the active information of the search.

2) *Performance of a ratcheted evolutionary strategy.* Available knowledge for a given search problem can be used with varying efficiency to produce active information. The stair step knowledge available to Avida is a rich source of active information and, in terms of efficient search, is used poorly by Avida in the search for EQU. We consider an alternate ratchet search strategy using only a single digital organism. A generation in the search consists of replacing an instruction in the loop by a randomly chosen instruction. If the fitness of the organism does not decrease, we keep the mutation and repeat the iteration. If the fitness does increase, the mutation is discarded and the process repeated. Different versions of this procedure are shown in Table III and are labeled $\mathcal{R}_1$ through $\mathcal{R}_4$. Each uses the same stair steps as algorithms $\mathcal{A}_1$ through $\mathcal{A}_4$. The same instruction bound and trial number are used in the ratchet simulations. $\mathcal{R}_1$ uses the same resources as $\mathcal{A}_1$ and increases the NAIPI over an order of magnitude. The NAIPI for $\mathcal{R}_2$ through $\mathcal{R}_4$ are likewise bettered significantly with respect to their corresponding Avida implementations in $\mathcal{A}_2$ through $\mathcal{A}_4$. As was the case with the Avida program, removing steps in the ratchet approach decreases algorithm performance as measured by active information.

TABLE III
THE EFFECTS OF REMOVING STAIR STEPS FROM AVIDA ($\mathcal{A}$) AND THE RATCHETED EVOLUTIONARY STRATEGY SEARCHES ($\mathcal{R}$) FOR THE EQU. THE ESTIMATE OF NAIPI IS COMPUTED USING (3). THE COMPARATIVE NUMBERS FOR REPEATED RUNNING OF RANDOMLY GENERATED DIMINISHED AVIDA PROGRAMS, AS DESCRIBED IN SECTION II-A4, IS IN THE LAST ROW. ALL SIMULATIONS, EXCEPT $\mathfrak{Q}$, HAVE 353 RUNS. THE RUN COUNT FOR $\mathfrak{Q}$ IS 1420 QUERIES. THE VALUES IN THE $I_\oplus/I_\Omega$ COLUMN SHOULD BE MULTIPLIED BY 10^{-9}.

Model	Steps Removed	Successes	$I_\oplus/I_\Omega$
$\mathcal{A}_1$	None (All Steps Enabled)	346	1.90
$\mathcal{A}_2$	XOR/NOR	319	1.37
$\mathcal{A}_3$	XOR/NOR/OR/AND_N	227	0.62
$\mathcal{A}_4$	XOR/NOR/AND/OR_N	222	0.52
$\mathcal{R}_1$	None (All Steps Enabled)	353	25.30
$\mathcal{R}_2$	XOR/NOR	353	16.26
$\mathcal{R}_3$	XOR/NOR/OR/AND_N	353	6.72
$\mathcal{R}_4$	XOR/NOR/AND/OR_N	353	8.05
$\mathfrak{Q}$	None (Random)	21	5.78×10^{-3}

Will Avida work without the stair step active information

[6]There are searches where removing steps can actually improve performance [4].

TABLE IV

EFFECTS OF INSTRUCTION REMOVAL ON AVIDA ($\mathcal{A}$) AND THE RATCHETED EVOLUTIONARY STRATEGY SEARCHES ($\mathcal{R}$) FOR EQU. THE LETTERS IN THE INSTRUCTIONS COLUMN CORRESPOND TO THE LETTER LABELING OF THE INSTRUCTIONS IN TABLE I. THE ENTRIES FOR $\mathcal{A}_1$ AND $\mathcal{R}_1$ IN TABLE III ARE REPEATED HERE FOR EASE OF COMPARISON. RATCHET ALGORITHMS $\mathcal{R}_5$ AND $\mathcal{R}_6$ USED THE SAME INSTRUCTIONS AS $\mathcal{A}_5$ AND $\mathcal{A}_6$. ALL SIMULATIONS HAVE 353 RUNS. THE VALUES IN THE $I_\oplus/I_\Omega$ COLUMN SHOULD BE MULTIPLIED BY 10^{-9}.

Model	Instructions	Successes	$I_\oplus/I_\Omega$
$\mathcal{A}_1$	`abcdefghijklmnopqrstuvwxyz`	346	1.90
$\mathcal{A}_5$	`abc--fghi--lmnopqrstuv--y-`	353	5.16
$\mathcal{A}_6$	`abc--fg-i------pqrstuv--y-`	353	10.00
$\mathcal{R}_1$	`abcdefghijklmnopqrstuvwxyz`	353	25.30
$\mathcal{R}_5$	`abc--fghi--lmnopqrstuv--y-`	353	197.76
$\mathcal{R}_6$	`abc--fg-i------pqrstuv--y-`	353	593.80

being hard wired into the process? The Avida paper reports that, in all case studies using stair step active information [16],

"... at least one population evolved EQU."

What happens when no stair step active information is applied?

"At the other extreme, 50 populations evolved in an environment where only EQU was rewarded, and no simpler function yielded energy. We expected that EQU would evolve much less often because selection would not preserve the simpler functions that provide foundations to build more complex features. Indeed, none of these populations evolved EQU, a highly significant difference from the fraction that did so in the reward-all environment." [16]

Therefore, hard wired stair step active information is essential in order for Avida to produce results in reasonable time. We were able to do so in Section II-A2 only because the instruction set was diminished and the query count far exceeded the effort reported in the Avida paper [16].

6) Instruction Selection Active Information in the NAND Search: Some of the instructions in Table I are essential to do any of the logic functions in Table II. Performing any logic function without the (p) `nand` instruction, for example, would be difficult. Conversely, there are instructions in Table I that make little or no contributions to the goals of Avida. The instructions (j) `shift-r`, (k) `shift-l`, (l) `inc`, (m) `dec` (n) `add`, and (o) `sub`, for example, are typically deleterious to the performance of a logic operation.

The effects of removing various sets of instructions in Avida and the ratcheted evolutionary strategy searches are shown in Table IV. For both Avida and the ratchet search, removal of deleterious or otherwise nonessential instructions increases the active information significantly. Using only 14 of the 26 instructions in $\mathcal{R}_6$ increases the NAIPI with respect to the vanilla Avida in $\mathcal{A}_1$ by a factor of over 300.

7) Initialization Active Information in the NAND Search: Another potential source of active information in search algorithms is initialization. If a search can be initialized in some sense closer to a solution, then we might expect faster convergence. Like any source of active information, the prior knowledge must be right. If we place the initialization farther from a solution, for example, convergence can take much longer.

TABLE V

EFFECTS OF DIFFERENT INITIALIZATIONS ON THE AVIDA ($\mathcal{A}$) FOR EQU. THE DOA DESIGNATION INDICATES THAT NEARLY ALL ORGANISMS DIED BEFORE PRODUCING ANY OFFSPRING. IN $\mathcal{A}_9$, DEATHS WERE CAUSED LARGELY BY INSTRUCTIONS (V) `mov-head` AND (W) `jmp-head`. THESE ENTRIES WERE REMOVED IN $\mathcal{A}_{11}$ AND SUCCESSES WERE ACHIEVED. A RANDOM SELECTION OF `nop-A`, `nop-B`, `nop-C` IN ALGORITHM $\mathcal{A}_{10}$ ALSO RESULTED IN A DOA SCENARIO. [THE REASON IS AS FOLLOWS. AS PART OF THE REPLICATION LOOP, AVIDA SEARCHES FOR PATTERNS OF `nop`'S. IN PARTICULAR THE END OF THE AVIDA PROGRAM IS MARKED BY `nop-A`, `nop-B`. IF THAT PATTERN APPEARS ANYWHERE ELSE IN THE PROGRAM IT WILL BE DETECTED AS THE END OF THE PROGRAM THUS CAUSING THE REPLICATOR TO FAIL. WITH 85 RANDOMLY CHOSEN `nop`'S, THAT PATTERN IS ALMOST CERTAIN TO APPEAR.] THE ENTRY FOR $\mathcal{A}_1$ FROM TABLE III IS REPEATED HERE FOR EASE OF COMPARISON. ALL SIMULATIONS HAVE 353 RUNS. THE VALUES IN THE $I_\oplus/I_\Omega$ COLUMN SHOULD BE MULTIPLIED BY 10^{-9}.

Model	Initialization	Successes	$I_\oplus/I_\Omega$
$\mathcal{A}_1$	`nop-C`	346	1.90
$\mathcal{A}_7$	`nop-B`	318	0.81
$\mathcal{A}_8$	`nop-A`	324	0.85
$\mathcal{A}_9$	Random	DOA	DOA
$\mathcal{A}_{10}$	Random-`nops`	DOA	DOA
$\mathcal{A}_{11}$	Random w/o (u) and (v)	274	0.84

In Avida, the instruction (c) `nop-C` plays an important role. The BX register is used as a default register and is used unless changed by a (a) `nop-A` or or (c) `nop-C` instruction. In the default operation of nand, for example, the nand operation of registers BX and CX is performed and deposited in register BX. Storing intermediate values of Avida's computation in register CX is therefore mandatory in computing the nand and can only be done using the (c) `nop-C` instruction. Apparently aware of this requirement, the designer of Avida set the initial value of all instructions not dedicated to organism replication to `nop-C`. The performance of Avida with this initialization is superior to initialization using nonmandatory `nop-A` or `nop-B` instructions. The performance of these and other initializations are summarized in Table V. In all cases, the NAIPI deteriorates by at least a factor of a half, or as described in the figure caption, is DOA.

III. CONCLUSIONS

A. Active Information

The Avida program uses numerous sources of active information to guide its performance to successful discovery of the EQU logic function. The sources include the following.

- *Stair step active information.* In the initial description of Avida, the authors write [16]

  "Some readers might suggest that we stacked the deck by studying the evolution of a complex feature that could be built on simpler functions that were also useful."

  This, indeed, is what the writers of Avida software do when using stair step active information. The importance of stair step active information is evident from the inability to generate a single EQU in Avida without using it [16].

- *Active information from Avida's initialization.* The initialization in Avida recognizes the essential role of the `nop-C` instruction in finding the EQU. Initializing using all nonessential `nop-A` or `nop-B` instructions results in the a decrease in NAIPI in Avida.
- *Mutation, fitness, and choosing the fittest of a number of mutated offspring* [5] are additional sources of active information in Avida we have not explored in this paper.

B. Disclosure

According to the principle of *conservation of information* [3], [4], [5], [7], [9], [10], [19], [20], [21], [22], [23], [28], [30], all computer search algorithms of moderate to high difficulty require active information.

The conservation of information principle in computer search, as manifest in the No Free Lunch Theorems, are [2]

> "... very useful, especially in light of some of the sometimes outrageous claims that had been made of specific optimization algorithms."

To have integrity, computer simulations of evolutionary search like Avida should make explicit

(1) a measure or assessment of the difficulty, I_Ω, of the problem being solved,
(2) the prior knowledge that gives rise to the active information in the search algorithm, and
(3) a measure or assessment of the active information, *e.g.* either I_+ or $I_\oplus$, introduced by the prior knowledge.

References

[1] Pak K. Chan, **Digital System Design Using Field Programmable Gate Arrays**, Prentice Hall (1994)

[2] S. Christensen and F. Oppacher, "What can we learn from No Free Lunch? A First Attempt to Characterize the Concept of a Searchable," Proceedings of the Genetic and Evolutionary Computation (2001).

[3] William A. Dembski, **No Free Lunch: Why Specified Complexity Cannot Be Purchased without Intelligence**. Rowman & Littlefield Publishers, Inc., 2006.

[4] W.A. Dembski and R.J. Marks II, "The Search for a Search: Measuring the Information Cost of Higher Level Search," International Journal of Information Technology and Intelligent Computing, Vol. 3, No. 4, 2008.

[5] W.A. Dembski and R.J. Marks II, "Conservation of Information in Search: Measuring the Cost of Success," IEEE Transactions on Systems, Man and Cybernetics A, Systems and Humans, in press. Available online at `www.BobMarks.org`

[6] Manoel E. de Lima and David J. Kinniment, "Sea-of-gates architecture," Microelectronics Journal, Volume 26, Issue 5, July 1995, pp. 431-440

[7] W.A. Dembski and R.J. Marks II, "Life's Conservation Law: Why Darwinian Evolution Cannot Create Biological Information," in Bruce Gordon and W.A. Dembski, editors, **The Nature of Nature**, (Wilmington, Del.: ISI Books, 2010).

[8] Richard O. Duda, Peter E. Hart, and David G. Stork, **Pattern Classification**, Wiley-Interscience; 2 edition (2000).

[9] T.M. English, "Some information theoretic results on evolutionary optimization," Proceedings of the 1999 Congress on Evolutionary Computation, 1999. CEC 99. Volume 1, 6-9 July 1999.

[10] T.M. English, "Evaluation of Evolutionary and Genetic Optimizers: No Free Lunch," in **Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming**, L. J. Fogel, P. J. Angeline, and T Bäck, Eds., pp. 163-169. Cambridge, Mass: MIT Press, 1996.

[11] A. E. Gamal, J. L. Kouloheris, D. How, and M. Morf, "Bi-NMOS: A basic cell for BiCMOS sea-of-gates," in Proc. IEEE Custom Integrated Circuits Conf., 1989, pp. 831-834.

[12] J.D. Golic, " Techniques for Random Masking in Hardware," IEEE Transactions on Circuits and Systems I: Regular Papers, Volume 54, Issue: 2, 2007, pp. 291-300

[13] T. Hanibuchi, K. Higashitani, M. Hatanaka and A. Tada, "A bipolar-PMOS merged basic cell for 0.8 μm BiCMOS sea of gates," IEEE Journal of Solid-State Circuits, Volume 26, Issue 3, Mar 1991, pp.427 - 431.

[14] James L. Hein, **Discrete Structures, Logic, and Computability**, Jones & Bartlett Publishers (2009)

[15] M. Koppen, D.H. Wolpert, W.G. Macready, "Remarks on a recent paper on the 'no free lunch' theorems," IEEE Transactions on Evolutionary Computation, June 2001, Volume: 5 , Issue: 3, pp. 295 - 296.

[16] Richard E. Lenski, Charles Ofria, Robert T. Pennock and Christoph Adami, "The evolutionary origin of complex features," **Nature**, vol 423, 139-144 (8 May 2003).
Supplementary material is available at
`http://myxo.css.msu.edu/papers/nature2003/`.
An "Index of Avida Documentation," is available at
`http://dllab.caltech.edu/avida/v2.0/docs/`.

[17] R.J. Marks II, **Handbook of Fourier Transform and Its Applications**, Oxford University Press (2009).

[18] Cecília Reis and J. A. Tenreiro Machado, "Computational Intelligence in Circuit Synthesis," Journal of Advanced Computational Intelligence and Intelligent Informatics, Vol.11, No.9 pp. 1122-1127, 2007

[19] Cullen Schaffer, "Overfitting Avoidance as Bias (preliminary version)," Proceedings of Machine Learning: IJCAI Workshop on Discovery in Databases, (1991).

[20] Cullen Schaffer, "Deconstructing the Digit Recognition Problem," in Machine Learning: Proceedings of the Ninth International Conference (ML92), D. Sleeman and P. Edwards Editors, Morgan Kaufmann, 1992

[21] Cullen Schaffer, "Sparse Data and the Effect of Overfitting Avoidance in Decision Tree Induction," Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92), 1992.

[22] Cullen Schaffer, "Overfitting Avoidance as Bias," Machine Learning, Volume 10 , Issue 2 (February 1993) Pages: 153 - 178

[23] Cullen Schaffer, "A conservation law for generalization performance," in Proc. Eleventh International Conference on Machine Learning, H. Willian and W. Cohen. San Francisco: Morgan Kaufmann, 1994, pp.295-265.

[24] Rajan Srinivasan, **Importance Sampling**, Springer (2002).

[25] K. Takita and Y. Kakazu, "Automatic agent design based on gate growth-application to wall following problem," Proceedings of the 37th SICE Annual Conference. International Session Papers, 29-31 July 1998, pp. 863 - 868.

[26] K. Takita, Y. Kakazu, "Evolutionary design of autonomous agent based on gate growth," Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems, 1999. IROS '99, Volume 3, 17-21 Oct. 1999 pp. 1555 - 1560 vol.3.

[27] David H. Wolpert, "On overfitting avoidance as bias." Technical Report SFI-TR-92-03-5001, Santa Fe Institute. 1992.

[28] David H. Wolpert, "On the connection between in-sample testing and generalization error." Complex Systems 6: pp.47-94 (1992)

[29] David H. Wolpert, "Stacked generalization." Neural Networks 5:241-259 (1992).

[30] David H. Wolpert, William G. Macready, "No free lunch theorems for optimization," IEEE Trans. Evolutionary Computation 1(1): 67-82 (1997).

[31] David H. Wolpert, and W.G. Macready, "Coevolutionary Free Lunches," IEEE Transactions on Evolutionary Computation, December 2005, Volume 9, Issue 6, pp. 721-735.

[32] M. Yasunaga, T. Nakamura and I. Yoshihara, "Sonar spectrum recognition chip designed by evolutionary algorithm," International Joint Conference on Neural Networks, 1999. IJCNN '99. Volume 5, 10-16 July 1999 pp. 3182 - 3187 vol.5.