Research Article

Conservation of Information in Coevolutionary Searches

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

[1]Biologic Institute, Redmond, WA, USA
[2]Electrical and Computer Engineering Deparment, Baylor University, Waco, TX, USA

Abstract

A number of papers show that the *No Free Lunch* theorem does not apply to coevolutionary search. This has been interpreted as meaning that, unlike classical full query searches, coevolutionary searches do not require extensive *a priori* knowledge about the domain. Alternately, coevolutionary searches can be viewed as providing incomplete information about fitness and differ from standard evolutionary searches where queries provide full fitness information. Knowing the full value of a fitness is better than knowing partial subjacent fitness information. Consequently, coevolution can be viewed as a degradation of search performance in this sense. We demonstrate this in a number of examples drawn from free lunch proofs in the literature. This observation does not diminish the power or utility of the coevolutionary search. Coevolutionary subjacent queries are often required due to the unavailability or expense of a full query. Nevertheless, coevolution does not allow an escape from the necessity of exploiting prior information in search processes and remains bounded by conservation of information in general and the *No Free Lunch* theorem in particular.

Cite As: Ewert W, Marks II RJ (2017) Conservation of Information in Coevolutionary Searches. Bio-Complexity 2017 (1):1-15. **doi:**10.5048/BIO-C.2017.1.

Editor: Douglas D. Axe

Received: November 3, 2016; **Accepted:** Februrary 27, 2017; **Published:** August 8, 2017

Notes: A *Critique* of this paper, when available, will be assigned **doi:**10.5048/BIO-C.2017.1.c.

*Email: evoinfo@winstonewert.com

INTRODUCTION

Coevolutionary searches are searches where the fitness of a particular solution depends not only on that solution, but also other factors. Such searches have been used in a wide variety of situations. Examples include sorting networks [1], the morphology and performance of competing agents [2], backgammon [3], checkers [4] and chess [5]. While traditional searches require the expertise of penalty function artists to craft a fitness function that guides the algorithm, coevolution is viewed as not requiring this prior expertise. Wolpert and Macready's seminal *No Free Lunch* (NFL) theorem [6,7] was followed by their paper *Coevolutionary Free Lunches* [8] where coevolution was found in certain cases to provide a free lunch. A free lunch is a case where one search algorithm performs better on average than another search algorithm over all possible search problems. The ability of coevolution to violate the NFL theorems have been subsequently echoed. Examples include:

- "coevolutionary algorithms require little a priori knowledge about the domain." [9]
- "...recent work ...[shows] that free lunches do exist for certain solution concepts." [10]. "
- "[For coevolution], a free lunch does exist." [11]

We show that the NFL theorem *does* apply to coevolution when illuminated from the viewpoint of subjacent versus full queries. By subjacent query, we mean a query which provides only part of the information provided by a traditional query. This conclusion does not invalidate or diminish the importance or utility of coevolutionary search nor, in the absence of problem-specific information, the demonstrated superiority of one subjacent query choice over another. While coevolutionary searches are useful, full queries are often not directly available or are prohibitively expensive. What we show is that coevolutionary search, appropriately posited, is still constrained by the NFL theorem.

Conventional search algorithms, including evolutionary search, require prior knowledge because of the *conservation of information* (COI) principles pioneered by Mitchell's analysis of unbiased generalization (1980) [12], Schaffer's *Law of Conservation of Generalization Performance* (1994) [13], and Wolpert & Macready's *No Free Lunch* theorems (1995) [6,7]. Without problem-specific information, no specified algorithm will succeed better on average than any other [14–16]. An algorithm must "pay" for success on one problem with poor performance on another problem.

Mathematical justification for the success of coevolutionary algorithms has been offered in the form of free lunches [8,17–19]. The NFL theorem is said not to apply to the case of coevolutionary algorithms. A free lunch, though, refers to a pair of algorithms which have differing performance even when averaged across all possible search problems. In this sense some coevolutionary algorithms appear to obtain free lunches and are better than other algorithms even when problem-specific information is not exploited. Coevolution thus appears to escape the "curse" of COI and does not require the use of a priori knowledge to solve search problems.

However, for COI results to apply in the case of conventional search but not in the case of coevolution is odd. Coevolution would appear to be at odds with COI results. What is different about these coevolutionary searches that allows them to not require prior information? The answer is that coevolution queries inferior or *subjacent* fitnesses that are combined into estimates of *full fitness* queries. The analysis of coevolution by others demonstrates there is the appearance of a free lunch when the space of subjacent fitness values is analyzed in the same way conventional NFL theorems are derived. This subjacent space, however, contains inferior information with respect to full fitness values. From the viewpoint of the space of full queries in coevolutionary searches, COI theorems still apply. In this sense, the NFL is applicable to coevolution. Nevertheless, coevolution has been shown to be effective in numerous cases and our results should no way be construed to discredit coevolution as a viable and useful approach to optimization. Multiple subjacent queries, for example, can come at a cheaper cost when compared to full queries. Full queries can be available only through repeated subjacent queries.

1. PRIOR WORK

A number of purported free lunch results have been presented. Many introduce some sort of assumption, restricting the set of possible search problems. Examples include low complexity [20,21] and fitness function smoothness [16,22]. Although authors often describe these results as "free lunches," lunch is in truth paid for by the introduction of an assumption or restriction of the set of possible functions. Furthermore, some results suggest limitations on the usefulness of such assumptions [23]. Other free lunches make use of relative performance metrics [7,24]. Yet others demonstrate free lunches by considering algorithms which select a non-optimal solution found despite having gathered enough information to select the optimal solution. For example in multiobjective optimization, memory constraints may prevent storing enough information to choose the best result [19]. But such constraints only decrease the performance of the algorithm. Any of these free lunches may be worthy of future study, but are not our concern here.

Search on non-trivially sized problems has been shown to be hopeless without exploiting problem-specific information. Because a coevolutionary search is necessarily less effective per query than a full query search, coevolutionary search suffers from the same restrictions.

The existence of free lunches is independent of the monotonicity of the solution concept [25]. A search has a monotonic solution concept if it is well behaved: performance does not degrade with additional queries. However, since this is independent of the existence of a free lunch, it suggests that free lunches do not derive from ill-behaved solution concepts.

The *terminating free lunch theorem* [26] considers a special case of coevolutionary algorithms where it is possible to prove that a candidate solution is invalid before having finished evaluating all the queries about that solution. When all algorithms make use of the same criterion for terminating, they will all have the same performance. In effect, all strategies for deciding which candidate solutions to investigate have the same performance. Any difference in performance results from differing strategies in selecting the queries related to those candidate solutions.

2. BACKGROUND

The full query NFL framework models the search process as one which makes a series of full queries to determine the values of a fitness function at various candidate solutions. Each query is a test which determines the exact fitness or cost of a particular candidate solution. After performing the test, all relevant details about the search algorithm are known. In the case of coevolution, rather than having a query provide the exact full fitness, each query only obtains partial information about the fitness of a candidate solution. While the full query NFL assumes that queries are "clear," under coevolution the queries are "muddy." We call the clear queries *full* and the muddy ones *subjacent*. A subjacent query contains less information than a full query.

A generalization of the coevolutionary search is shown in Figure 1. There are a number of candidate solutions each of which is represented by a row of a subjacent fitness matrix. Previous queries of the fitness populate

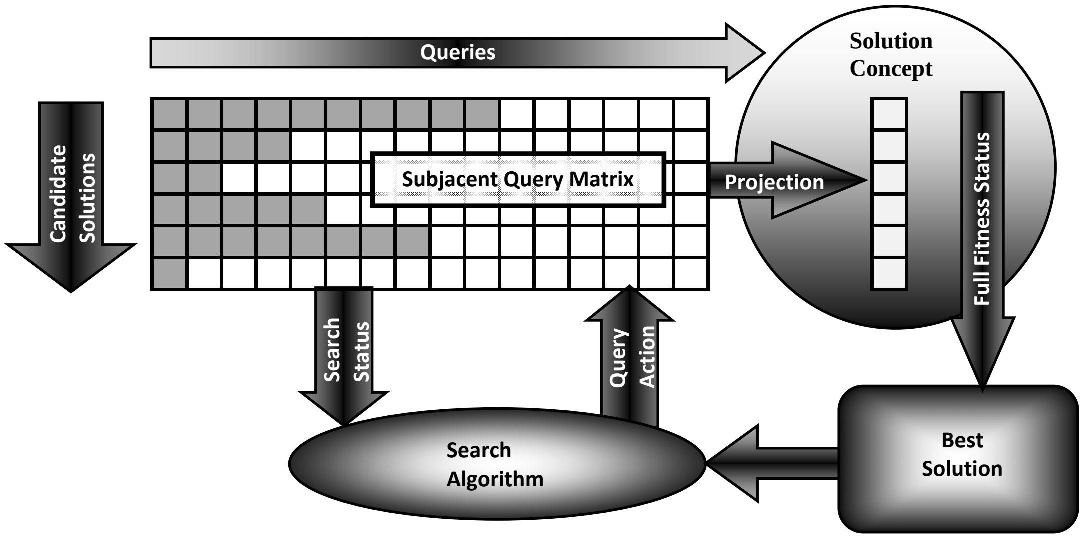


Figure 1: Coevolutionary search. doi:10.5048/BIO-C.2017.1.f1

the partially filled matrix. Results of known subjacent fitness values are projected onto the solution concept space. Projection often consists of aggregation operations including min, average, median and max. When a candidate solution is completely characterized, a row has essentially been successfully characterized by a single full fitness value.

The vector in the solution concept space on the right in Figure 1 is itself assessed to give the best solution. If, for example, the row projection in the subjacent fitness matrix is a minimum and the column aggregation in the solution concept space is a maximum, we have a classical maximin (maximum of minimums) optimization problem.

The coevolutionary search algorithm examines the status of query values in the subjacent fitness matrix to decide where the next subjacent query should be taken. The NFL might lead us to expect that without external knowledge about the target being sought, there would be no difference where in the matrix the next sample is taken. In the case of the subjacent query matrix, though, it does matter [8].

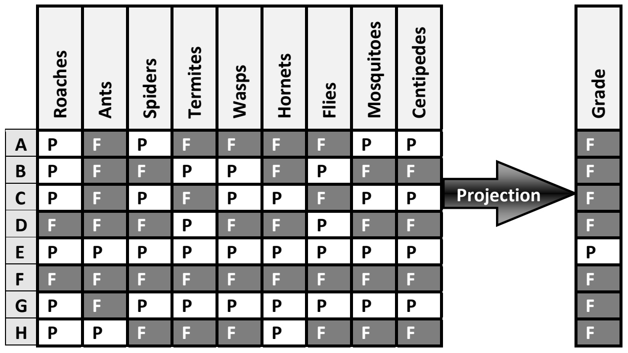
2.1 Examples

2.1.1 Maps

We begin with a high level example of subjacent queries. Imagine searching through a large library for a book which contains a treasure map hidden between two pages of a book. If you quickly flip through the pages in the book, you may miss the map or mistakenly believe that an old shopping list is the map. It takes multiple queries or flips through the book to conclusively determine whether or not it contains the map. There is a trade-off between thoroughly searching a few books and partially searching many books. A full query NFL search assumes that checking a book for the map consumes a single query. A coevolutionary search assumes that it takes several queries to conclusively determine whether or not the map is inside a book. The difference between full query and coevolutionary search is precisely that coevolutionary queries only provide partial information about a candidate solution.

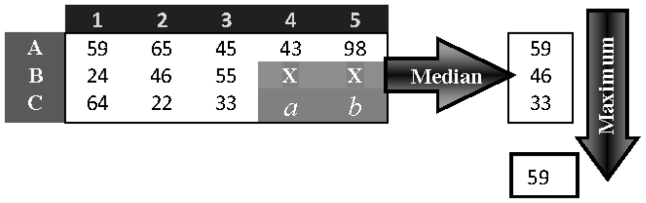
2.1.2 Product Testing

A simple example using the subjacent fitness matrix in Figure 1 is shown in Figure 2. Eight formulas of insecticides labeled A through H are to be tested on the nine different bugs. An insecticide is either effective on eradicating a pest (PASS) or not (FAIL). The insecticide must be effective on all the pests in order to pass the overall test. If all tests are performed, the subjacent fitness matrix would be as shown. The vertical aggregation is a logical AND so one failure disqualifies a candidate insecticide. We test formula A on roaches where it works and then on ants where it doesn't. There is no need to do any more tests on formula A so we begin to test formula B until it also fails on ants. If all eight formulas are to be tested sequentially from left (roaches) to right (centipedes), inspection of the subjacent fitness matrix reveals only 22 of the 72 available subjacent fitness values need to be queried to determine the full query values listed in the rightmost "Grade" column in Figure 2. As the coevolutionary search progresses, not all remaining subjacent queries are equally useful. There is no need to further test an insecticide in a row where a failure has been recorded.



	Roaches	Ants	Spiders	Termites	Wasps	Hornets	Flies	Mosquitoes	Centipedes	Grade
A	P	F	P	F	F	F	F	P	P	F
B	P	F	F	P	P	F	P	F	F	F
C	P	F	P	F	P	P	F	P	P	F
D	F	F	F	P	F	F	P	F	F	F
E	P	P	P	P	P	P	P	P	P	P
F	F	F	F	F	F	F	F	F	F	F
G	P	F	P	P	P	P	P	P	P	F
H	P	P	F	F	F	P	F	F	F	F

Figure 2: A search matrix for product testing. A candidate succeeds if all the test cases for that solution passes. **doi:**10.5048/BIO-C.2017.1.f2



	1	2	3	4	5	
A	59	65	45	43	98	59
B	24	46	55	X	X	46
C	64	22	33	a	b	33

Figure 3: An example where a problem context clearly reveals some subjacent queries will contain more information than others. The subjacent queries for the shaded boxes have not yet been made. The two boxes marked X are useless. They need never be queried because, whatever values they yield, B will never beat A. Solution candidate C will win, on the other hand, if box a exceeds 59, no more queries are required to determine whether A, B or C is the winner. **doi:**10.5048/BIO-C.2017.1.f3

2.1.3 Median Search

Another illustrative example is shown in Figure 3. There are three classrooms, A, B and C and each has five students. Which classroom has the oldest students? For the row projection, we choose the median operation. Every time we perform a query, we pay a fixed fee to the student for their age information. Our study thus far is shown in Figure 3. All five students have been queried in Classroom A and the median age is 59. Classrooms B and C each have three queries to the subjacent fitness matrix resulting in the full query estimates of 46 and 33 respectively. There are four students remaining to be queried. How do we best spend our money? A simple calculation shows that the biggest median possible for Classroom B is 55. This would happen even if the remaining two students were each 100 years old. Thus, any additional query in Classroom B is a waste of money because no outcomes exist that will exceed the median age in Classroom A. Classroom C, on the other hand, could be the winner or tied if both of the remaining students are 59 or older. The path to finding the best solution is now clear. We first query a student in Classroom C. If the age is less than 59, we are done. Classroom A wins no matter how old the fifth student in Classroom C. However if the queried student is 59 or older, the winner (Classroom A or Classroom C) will be decided by the fifth and final query in Classroom C. The two unanswered queries in the boxes marked X in Figure 3 are useless. The queries a and possibly b, on the other hand, will determine the winner. Not all subjacent queries are created equal.

2.2 Observations

In the product testing Example 2.1.2, the full fitness values can be found exactly using only a portion of the subjacent fitness matrix. In the median search Example 2.1.3, only a portion of available subjacent queries are needed to exclude with certainty a candidate solution from further consideration. Often, an estimate of the full fitness value will be made by choosing subjacent queries using probabilistic measures to minimize the uncertainty of the full fitness estimate. In general, the chosen assessment operations impose a structure on the subjacent fitness matrix revealing that some future queries will be probabilistically more useful than others.

From the perspective of full fitnesses in the solution concept space, the NFL still applies. Given a space of median class ages, finding the maximum is same as finding the ace of spades in a well shuffled deck of cards. One search algorithm will perform on average as well as any other.

3. ANALYSIS

3.1 Foundations

A search is a process which selects a solution from a set of potential solutions, Ω. Previous work has considered selecting a champion from a tournament [8], or choosing a solution configuration [18] in either case being equivalent to selecting from the set of possible solutions. Some set $T \subset \Omega$, is consider to be the target. A search succeeds if it selects a solution $\omega \in T$. Each row in the query matrix tables above corresponds to a potential solution. The performance of a search algorithm relative to a baseline search can be measured using *active information* [27].

$$I_+ = -\log_2 \frac{p}{q}$$

where p is the probability of the baseline search being successful and q is the probability of success for the comparative search. Typically, the baseline search is taken to be a single random guess made uniformly over the search space, Ω. This means that $p = \frac{|T|}{|\Omega|}$.

A search which performs better than the random guess requires information sources. Something must account for its ability to locate the target with better than random probability. Commonly, this takes the form of an oracle, which is queried by the search, thereby gaining access to information about the location of the target. A common case is the *needle in the haystack* (NIAH) oracle which indicates only whether or not a given solution is in the target. Using this oracle, it is possible to obtain approximately $\log_2 \hat{Q}$ bits of active information where $\hat{Q}$ is the number of queries performed [27].

For coevolutionary search, we will consider two classes of oracles. A coevolutionary oracle queries a single cell in the query matrix. This is a subjacent query. A row oracle queries an entire row. This is a full query. A full query is equivalent to several subjacent queries. A full oracle may be much more expensive, or even impossible to query in practice. However, we can still use it as a theoretical construct.

Theorem 1. *Given a query matrix where each value is drawn uniformly from any set X, and there are k values in each row, any algorithm using full queries of the row oracle is subject to the NFL theorem*

Proof. The result of querying a row from the query matrix is a tuple consisting of all the cells in the row. Since each cell is drawn from a uniform distribution on X, the row is drawn uniformly from a distribution on X^k. This scenario is equivalent to performing a search on the search space Ω where each fitness value is drawn uniformly from the set X^k. This fits the original formulation of the NFL theorem, and thus it applies. □

Theorem 2. *For any given algorithm, A, using the coevolutionary oracle and making $\hat{Q}$ queries, there exists an algorithm A' also making $\hat{Q}$ full queries that extracts greater or equal active information.*

Proof. Let A' be the same as A except that whenever A would make a query, A' will instead:

- Check whether the row containing the query has previously been queried
- If it has not been queried, query that row
- If it has been queried, query another random row

A' will identify the target whenever A identifies it. Additionally, due to having access to more cells of the table than A, A' may identify the target in cases where A fails to do so. Thus, A' will succeed with greater or equal probability as A. Consequently, A' will have greater or equal active information. □

Theorem 2 shows that any algorithm using a coevolutionary oracle will have equal or less active information to some algorithm using a row oracle. Theorem 1 shows that if we assume uniform distribution over possible search problems, that algorithm using the row oracle has the same performance as all other algorithms using the row oracle. That algorithm will extract approximately $\log_2 \hat{Q}$ bits of active information which means that for any algorithm using a coevolutionary oracle:

$$I_+ \lessapprox \log_2 \hat{Q}. \tag{1}$$

This allows us to bound the performance of coevolutionary search. In fact, per query, coevolutionary search is less efficient than a search using full queries. This does not mean that coevolutionary search is less efficient in practice. Coevolutionary queries are often much cheaper than row queries. Furthermore, previously published free lunch results stand, in that not all coevolutionary algorithms are created equal. Some algorithms have better performance than others.

Previous work on coevolutionary search has focused on whether or not these algorithms have the same performance. It has been amply demonstrated that the algorithms do have varying performance. However, do these differences in performance give coevolutionary search the ability to solve difficult search problems without exploiting problem-specific information? In some cases, cheaper coevolutionary queries will make some problems solvable. However, when the size of the search space dwarfs the number of available queries, finding a solution remains outside the grasp of coevolutionary algorithms which do not exploit problem-specific information.

4. EXAMPLES

To further demonstrate performance of coevolutionary subjacent queries, we consider several example problems contrasting search performance of various algorithms versus a NIAH algorithm using full queries. The examples are derived from those used in various coevolutionary free lunch proofs.

4.1 Algorithms

For each example, we select from the following menu of search algorithms. All of the algorithms, except for the NIAH, make use of subjacent queries.

- *Needle in the Haystack* (NIAH)

  This algorithm uses full queries. Because of the NFL theorem, all algorithms using full queries have the same performance on average. As a result, we only need to consider one such algorithm which makes a number of random queries that evaluate candidate solutions.

- *Horizontal*

  As shown in Figure 4A, the horizontal algorithm emphasizes depth over breadth. A single candidate solution is queried exhaustively. Then it moves onto another candidate solution. The procedure goes through the fitness matrix row by row, exhaustively evaluating all of the cells in a row before

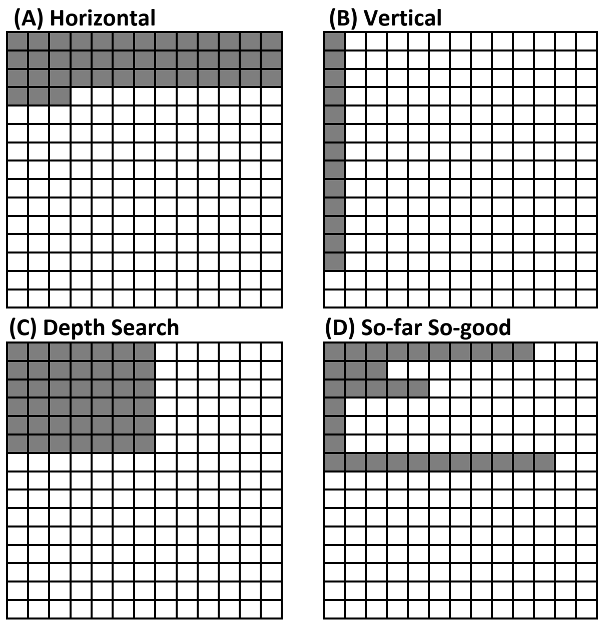


Figure 4: Querying strategies for a subjacent fitness matrix. (A) Queries are made by a horizontal algorithm. All the cells in a row are queried before moving on. (B) Queries made by the vertical algorithm. Only cells in a single column are queried. (C) For the depth search algorithm, a specific number of queries are made into each row. (D) Queries made by the so-far so-good algorithm examines a row until it is impossible for that solution to be in the target. **doi:**10.5048/BIO-C.2017.1.f4

moving to the next one. In effect, it uses the co-evolutionary queries to simulate a full query and thus always exhibits the performance of the NIAH algorithm with fewer queries.

- *Vertical*

  This algorithm emphasizes breadth over depth. See Figure 4B. Each candidate solution is queried once, thereby filling a single column of the subjacent fitness matrix for as many rows as available queries. We assume there are more candidate solutions than queries, and thus the available queries from the first column is never exhausted.

- *Depth Search*

  As seen in Figure 4C, both the horizontal and vertical algorithms can be seen as special cases of a more general depth search algorithm. The depth search algorithm has a parameter, the search depth, d. For each row, it queries d of the cells. If d is the row size then it is equivalent to the horizontal algorithm. If d is one, then it is equivalent to the vertical algorithm.

- *So-far so-good*

  A *so-far so-good* search algorithm follows the horizontal search algorithm except that it ceases querying on a row when it is no longer possible for the solution to be in the target. This is the type of algorithm modeled by Woodward and Neil's *terminating free lunch theorem* [26]. The solution concept defines the target, and depending on the type of projection, it may be possible to prove that a solution cannot be in the target given only partial knowledge. How the so-far so-good algorithm applies depends on the specific case, in some cases it may not apply at all.

4.2 Methods

Each problem has a table of possible coevolutionary queries as depicted in Figure 1. Depending on the specific problem considered, the values will either be true/false or a real number in the range $(0, 1)$. Either way a uniform distribution over the possible values of these queries will be assumed. The difference between the problems is primarily the definition of the solution concept; different rules and projections for the query space to the candidate solution space are defined. The performance of search algorithms will be reported using active information, discussed in Section 3.1. The baseline probability is the probability that a uniformly selected candidate solution will be in the target, according to the rule defined for the problem. The probability of success for the comparative search depends on the algorithm being evaluated as well as the number of queries being performed. For either probability, they are determined analytically when possible with the math being found in the appendix. When analysis of the probability proved intractable, we used Monte Carlo methods to simulate the search algorithm and thus estimate the probabilities.

4.3 Problems

4.3.1 Insecticides

In the example represented in Figure 2, we have a large number of candidate insecticides and k bugs that can be exposed to each insecticide. We want to find a candidate solution which kills all of the bugs. In terms of the general coevolutionary search in Figure 4, each entry in the solution concept vector is projected using the logical AND of the corresponding subjacent matrix row where a P = PASS =1 and F = FAIL = 0. A single fail dooms a candidate to lose.

A number of other search problems can be viewed as equivalent to this problem. It is common for a candidate solution to have to pass a number of tests such as correctly deciding whether a position in a string of DNA is a binding site [28] or an expression that produces the correct output for certain inputs [29] or the correct letter in a phrase [30]. In each of these cases the problem can be clearly divided into a series of tests, all of which must be passed for a complete solution.

A less obviously equivalent problem is a variation on min-max coevolutionary search [8]. This is a coevolutionary search where the fitness of a candidate is the

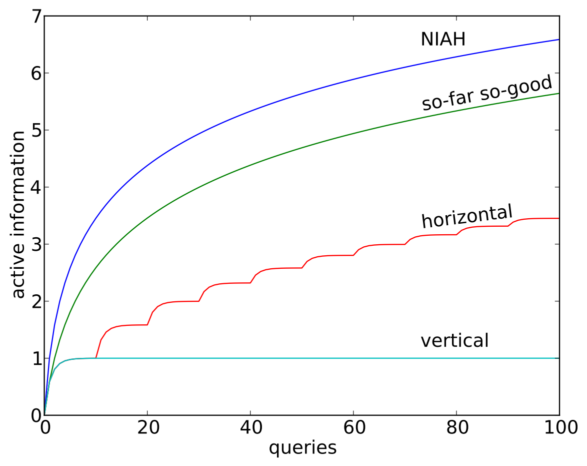


Figure 5: Active information in bits for varying queries across the three algorithms for passing test cases. Data is analytically derived, details are in the Appendix (Section 6.1). There are 10 test cases to pass each with 50% probability. **doi:**10.5048/BIO-C.2017.1.f5

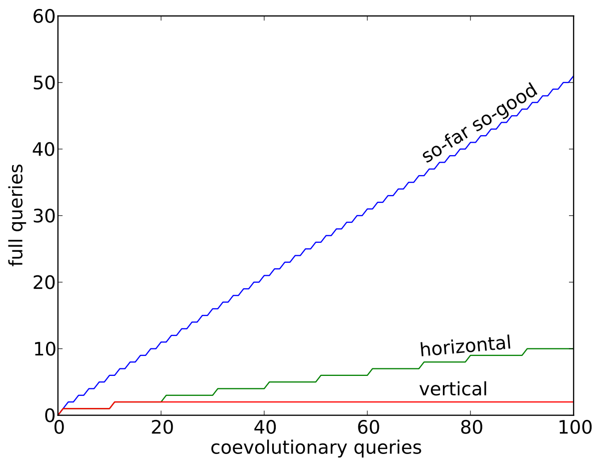


Figure 6: The number of full queries required to gain same performance as coevolutionary queries for the passing tests problem. Data is analytically derived, details are in the Appendix (Section 6.1). There are 10 test cases to pass each with 50% probability. **doi:**10.5048/BIO-C.2017.1.f6

minimum of the values in the row in the subjacent fitness matrix. Rather than looking for the absolute candidate maximum, we define the target as all solutions with fitness above a certain acceptable threshold. As long as a row contains only values above that threshold, then it can be in the target. Effectively, each row element in the subjacent matrix can be viewed as a link in a chain. We are looking for a chain whose weakest link is sufficiently strong. Each cell in the subjacent query matrix is a test for whether the link's strength is above or below an acceptable threshold. Each individual link must be strong enough, or the chain as a whole will be too weak. Thus each coevolutionary query can be considered as a test which must be passed, just as in the insecticide problem.

In order to apply the so-far so-good algorithm to this problem, we must determine when it is mathematically impossible for a partially queried candidate solution to be in the target. A single failure of an insecticide on a pest demonstrates that it does not pass the criterion. We can stop querying a row as soon as a single test has failed. That is, the algorithm should cease querying a row after a single failing query.

Figure 5 shows plots of active information for the four algorithms using varying numbers of queries. We see that there is a performance difference between the three coevolutionary algorithms. However, they are all easily beaten by the full query NIAH algorithm.

Figure 6 shows the number of full queries required in order to obtain the same active information as a varying number of coevolutionary queries. It gives an idea of how many coevolutionary queries are equivalent to a full query. We can view this as a query exchange rate, how many coevolutionary queries it takes to get the same performance as a full query. For example, 100 coevolutionary queries is equivalent to approximately 51 full queries.

The so-far so-good algorithm consistently takes about twice as many queries as the full query search in Figure 6. Each test is a Bernoulli trial with expected time $\frac{1}{p}$ where p is the probability of failure [31]. In this case $p = \frac{1}{2}$, so the expected time is 2. The so-far so-good algorithm thus spends an average of about 2 queries in a row, which is why it takes approximately twice as many queries.

Figure 7 shows the active information for 100 queries for varying values of k, the number of test cases to be passed.[1] The probability of success is set so that the probability of any candidate solution being in the target is 10^{-4}. The graph contrasts having to pass a few hard tests with many easy tests. For small values of k the algorithms have similar performance because at $k = 1$ a coevolutionary query is a full query. However, the performance of the coevolutionary algorithm degrades as k increases. This is intuitive, because when more tests have to be passed each test is less informative.

Figure 8 shows the active information for varying values of k where the number of queries is varied. The coevolutionary algorithms have been given $100k$ queries, whereas the NIAH algorithm's query count is fixed at 100 queries. By performing more queries, the coevolutionary algorithms have better performance than the 100 query NIAH. However, they are are still inferior to the $100k$ query NIAH.

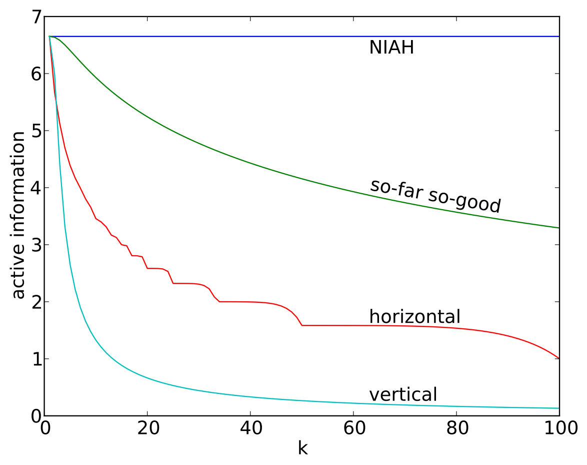


Figure 7: Active information for varying numbers of tests (k). Data is analytically derived. Details are in the Appendix (Section 6.1). Probability of success = $10^{-4/k}$. 100 queries are performed. **doi:**10.5048/BIO-C.2017.1.f7

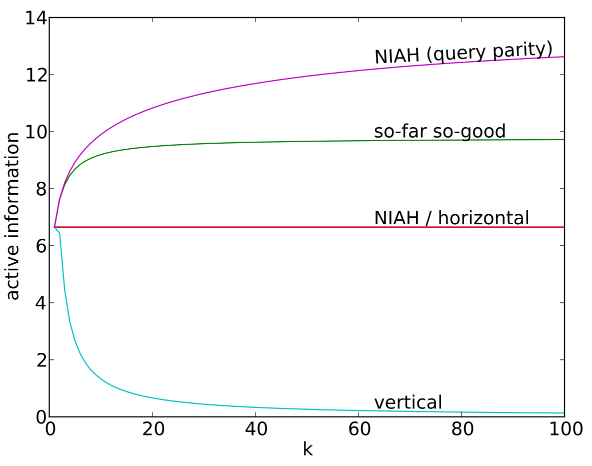


Figure 8: Active information for varying numbers of tests (k) with $100k$ queries for all but NIAH, which has 100 queries. Data is analytically derived, details in the Appendix (Section 6.1). Probability of success = $10^{-4/k}$. **doi:**10.5048/BIO-C.2017.1.f8

Table 1: A fitness matrix for average showing the averages and the target in bold

Features				Average
0.60	0.11	0.53	0.14	0.34
0.26	0.10	0.62	0.71	0.42
0.50	0.61	0.36	0.10	0.39
0.38	0.63	0.30	0.48	0.45
0.69	0.51	0.73	0.29	**0.55**
0.37	0.84	0.28	0.60	**0.52**
0.44	0.06	0.73	0.74	0.49
0.22	0.48	0.56	0.20	0.36
0.37	0.50	0.69	0.40	0.49
0.34	0.49	0.22	0.23	0.32

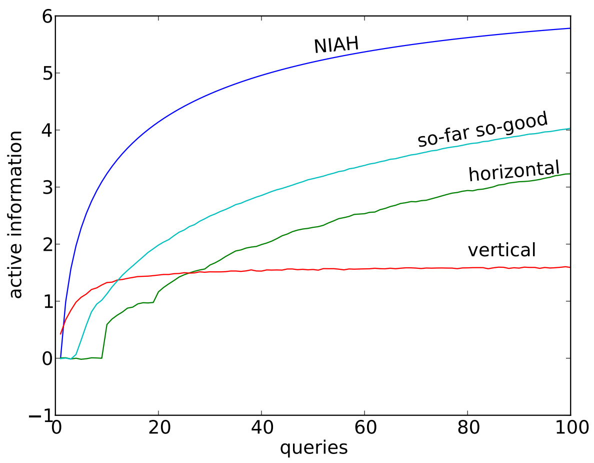


Figure 9: Active information for finding an average above 0.7 comparing the different algorithms and varying numbers of queries. Row size is 10. Data is obtained by Monte Carlo. **doi:**10.5048/BIO-C.2017.1.f9

4.3.2 Average

In this scenario, each candidate solution has ten different features, each represented by a number between 0 and 1. The goal is to maximize the average or sum of these scores in the selected candidate solution. This is depicted in Table 1. We deem a search to have succeeded if it selects a candidate solution with an average score above some threshold. In Table 1 this is 0.5.

The vertical and horizontal algorithms obviously apply. The so-far so-good algorithm can be implemented by observing whether or not the current average is sufficiently low that no possible values in the rest of the row could bring it above the threshold. Suppose that after querying j of k subjacent queries for a given candidate solution we have an average of m. The highest possible average is if the remaining queries are all 1. Thus the maximum achievable average is:

$$\frac{jm + (k - j)}{k}.$$

Queries to a given row will be terminated if the maximum achievable average falls below the desired threshold.

The NIAH algorithm was analyzed as described in the Appendix (Section 6.2), but the horizontal, vertical, and so-far so-good algorithms were evaluated by Monte Carlo simulations. One millions iterations were run for each value.

Figure 9 shows the active information produced by the various algorithms. The needle in the haystack oracle has the best performance. The so-far so-good algorithm is the best coevolutionary algorithm for more than a few queries. The vertical algorithm performs well for a few

[1]In this and subsequent plots, active information is measured in bits.

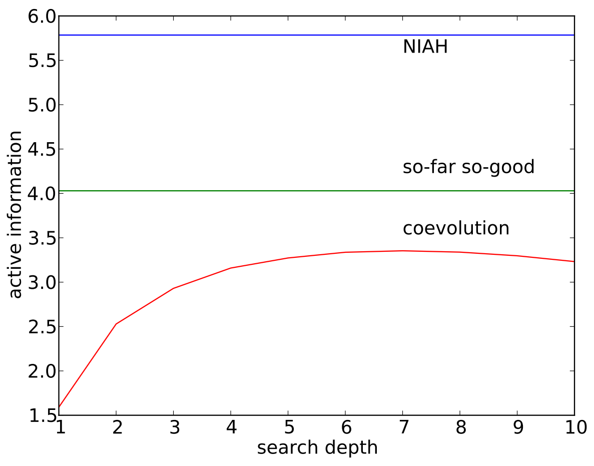


Figure 10: Active information for finding an average above 0.7 for varying search depths and 100 queries. Data is obtained by Monte Carlo. doi:10.5048/BIO-C.2017.1.f10

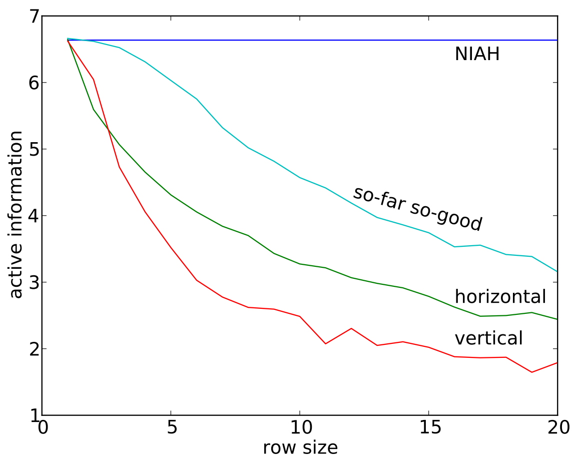


Figure 11: Active information for finding an average above the threshold given varying row sizes and 100 queries. Threshold set so that the probability of a randomly selected row being in the target is 10^{-4}. The data is obtained by Monte Carlo simulation. doi:10.5048/BIO-C.2017.1.f11

queries, but its performance plateaus as more queries are made available. The horizontal algorithms consistently trails the so-far so-good algorithm by about one bit of active information.

Both the horizontal and vertical algorithms can be viewed a special case of a more general algorithm. The algorithm has a parameter, the search depth, which determines how many cells from each row are queried. The horizontal algorithm in the above examples queries 10, whereas the vertical algorithm queries 1. Figure 10 shows the active information for varying search depths. While small search depths have poor performance, all other search depths have similar performance, significantly less then the performance of a NIAH search, and still less than the so-far so-good search.

Figure 11 shows the active information for different row sizes. For small row sizes, both algorithms have the same performance, because for a row size of one a coevolutionary query is a full query. As the row size increases, the active information extracted by the algorithm decreases.

4.3.3 Pareto Coevolution

Another possible scenario is *Pareto coevolution*. In this case each column is a criterion on which the various candidate solutions are judged. For example when purchasing a vehicle, the criterion might be gas mileage, cost, cool factor, etc. A depiction of this scenario is seen in Figure 12.

When evaluating multiple criteria, deciding which candidate solution is best can be a problem. For example, one car might have very good gas mileage, but be very expensive, while another car has terrible gas mileage but is cheap. There is no obvious way to break this tie. However, some solutions are clearly worse than others. A car which costs more, gets worse gas mileage and is less cool than another car cannot be the right choice. This is referred to as one solution dominating another. Figure 12 shows this in the case of the car examples. Several possible cars are dominated because another car is better in both categories.

A Pareto search algorithm keeps track of all of the points which are have not been dominated by another point. This is known as the Pareto front. For the cars in the Pareto front, no car is better in all categories. For the purposes of evaluating algorithm performance, we want to measure the algorithm's success at finding the Pareto front. We can do this by measuring success at finding a dominated point.

The Pareto search algorithm works not by locating the points in the Pareto front but by determining which points are not in the Pareto front. Before any queries are made, based on the available information, no candidate solution dominates any other solution. That is, the initial approximation to the Pareto front includes the entire search space. As queries are made, some points are found to be dominated by other points and are thus removed from the approximation to the Pareto front. Thus we define a search as successful if it finds a dominated point.

We have to change the definition of the baseline search because a single guess no longer makes sense. Instead, the search must guess two points: the dominated point and the point it is dominated by. The search succeeds if the point is so dominated.

To illustrate the so-far so-good algorithm, we make use of the insight of Service et al. [17], who observed that we can prove that two points do not dominate one another without querying all of the cells. If the first car

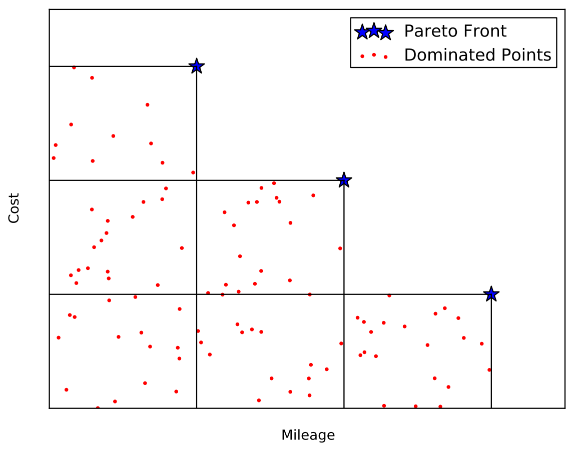


Figure 12: Depiction of Pareto optimization for car example. doi:10.5048/BIO-C.2017.1.f12

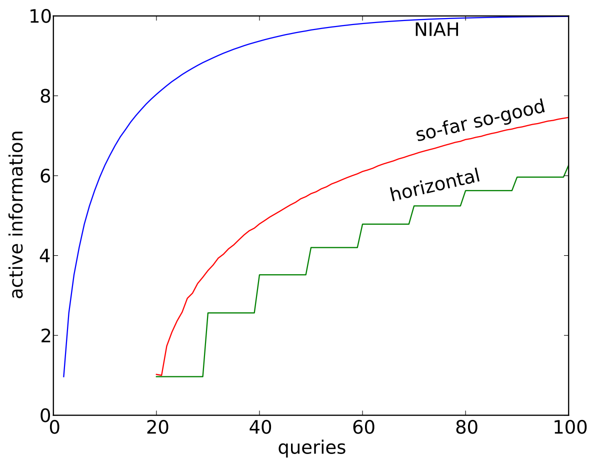


Figure 13: Active information for finding a dominated point given different algorithms and varying numbers of queries. Row size is 10. Data is obtained by Monte Carlo. doi:10.5048/BIO-C.2017.1.f13

beats the second car on price, but the second car beats the first car on mileage, there is no reason to look at the relative cool factor of either car. It is impossible for either car to dominate the other, so the query can be saved. The algorithm proceeds as the horizontal algorithm does except that a solution is only queried enough to demonstrate that it neither dominates nor is dominated by any other solution.

Figure 13 shows the performance of the algorithms for varying numbers of queries. The needle in the haystack oracle clearly significantly outperforms either of the coevolutionary algorithms. The vertical algorithm first succeeds with 20 queries, because it is able to to query the two rows completely and thus succeeds if one of them dominates the others. It proceeds in jumps every 10 queries because partially queried rows do not help the algorithm.

The so-far so-good algorithm proceeds better than the vertical algorithm. It first succeeds at 20 queries because it must fully query the dominated and dominating rows to show they are dominated. This requires 20 queries. But because complete rows need not be queried, there are no jumps in performance.

Figure 14 shows the query exchange rate for coevolutionary and full queries. 100 coevolutionary queries used by the so-far so-good algorithm only equates to 16 full queries.

4.3.4 Best in all Categories

A related scenario would be to find the candidate solution which is the best in every category. This is related to the scenario considered by Service et al. [18]. Such a solution will not always exist, but we can still view it as the target of a search. As in other search scenarios, the target set can end up being empty.

Figure 15 shows the performance of the algorithms for various numbers of queries. The vertical and NIAH searches have exactly the same performance. Figure 16 shows the query exchange rate, and there is 1 for 1 exchange rate between full queries and coevolutionary queries.

Why is this the case? The solution concept requires the candidate solution to be the best in every category. Much of the time, no solution will fit the requirements. However, if a solution does fit the requirements, it can be identified by choosing the maximum score for a single criterion.

Figure 17 shows the active information as it varies for different row sizes. The horizontal algorithm is strongly affected by the row size, but NIAH and vertical algorithms maintain the same performance throughout.

As argued, the performance of the coevolutionary algorithms is bounded by the full algorithm. In this case the nature of the solution concept makes it possible for the performance to be equal.

5. CONCLUSIONS

For the same number of queries, an algorithm using full queries provides a superset of the information of the coevolution algorithm and as such cannot be inferior. In fact, by demonstrating a number of examples drawn from free lunch proofs in the literature, we see that the coevolutionary algorithms often perform noticeably worse than the full query algorithms. However, we have also shown that this is not necessarily the case as one of the examples shows a coevolutionary algorithm with the same performance as the full query algorithm. Nevertheless, the examples suggest that coevolutionary algorithms perform worse or equal to the classical full query algorithm.

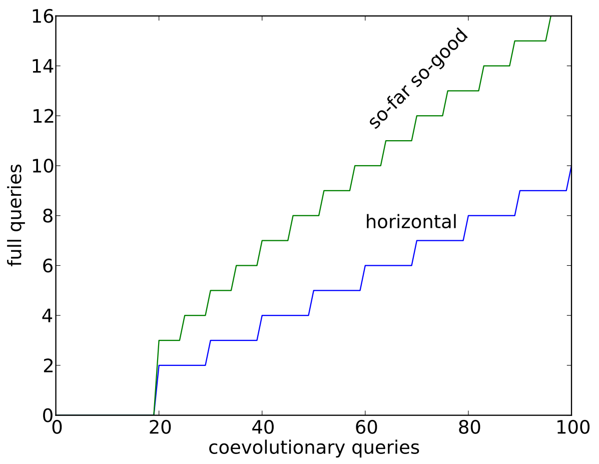


Figure 14: Query exchange rate for coevolutionary and full queries for finding Pareto dominated points. Row size is 10. Data is obtained by Monte Carlo. **doi:**10.5048/BIO-C.2017.1.f14

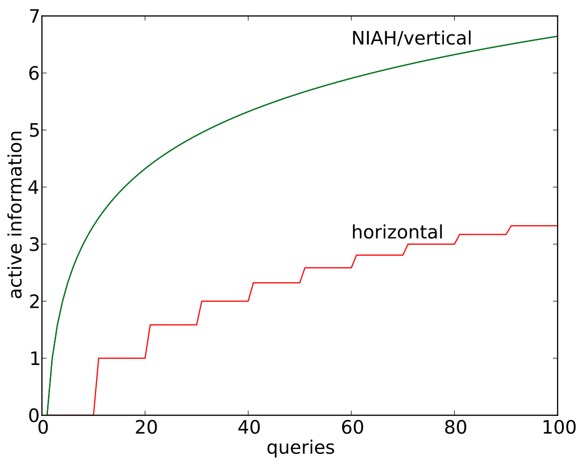


Figure 15: Algorithm performance for finding a candidate solution which is the best in all categories given differing numbers of queries. Row size is 10 and total search space size is 1000. Data is obtained analytically, details in the Appendix (Section 6.4) **doi:**10.5048/BIO-C.2017.1.f15

We conclude that there is nothing special in a coevolutionary algorithm that absolves it from the necessity of making use of prior information to solve non-trivial searches. If the full query algorithm requires prior knowledge, so do the coevolutionary algorithms because they perform worse than the full query algorithm. None of the free lunch proofs provide evidence that less prior knowledge is required for coevolutionary algorithms. If these apparent free lunches do not provide a mathematical justification for success, what does? The success of coevolutionary algorithms like any other algorithms depends on the use of prior knowledge. The algorithms are designed to exploit the characteristics of the problem being solved.

For example, we have the evolution of sorting networks [1], one of the early examples of applying coevolutionary search. But the sorting network was not evolved from scratch, rather

> “because most of the known minimal 16-input networks begin with the same pattern of 32 exchanges, the gene pool is initialized to be homozygous for these exchanges.”

This constitutes an initialization of just over half of the genome. Additionally, the genome was structured taking into account the properties of sorting networks:

> “It may be easier, for example, to produce a short correct network by optimizing a slightly longer correct network than by fixing a bug in a short incorrect network. For this reason, we have taken advantage of the diploid representation of a genotype to allow longer networks to be generated as intermediate solutions.”

Another example is that of checkers [4]. The title of the paper indicates that no human expertise was used in the construction of the strategy. However, the human programmers used a min-max search algorithm as well as adjusting the structure of the neural network to include a piece differential. Both of these are uses of prior knowledge about checkers that human experts have inserted into the search algorithm.

The success of coevolutionary algorithms is not due to escaping the “curse” of the NFL theorem. Rather, the coevolutionary algorithms perform worse than the NFL would predict. Of course, when coevolutionary queries are cheaper or obtaining full queries is not practical, subjacent queries will continue to be worthwhile. Nevertheless, from the perspective of the full query algorithm, the NFL theorem still bounds the performance of the algorithm. Coevolutionary algorithms, like other search algorithms, require the exploitation of problem-specific information.

The principle of conservation of information, in the sense described, still stands strong.

6. APPENDIX

6.1 Test Cases

- Let k be the number of test cases that must be passed.
- Let p be the probability of passing a test case.
- Let Q be the number of queries performed.

The probability of a candidate solution being in the target is p^k. The probability of set of x randomly selected candidate solutions including at least one solution in the target is $1 - (1 - p^k)^x$.

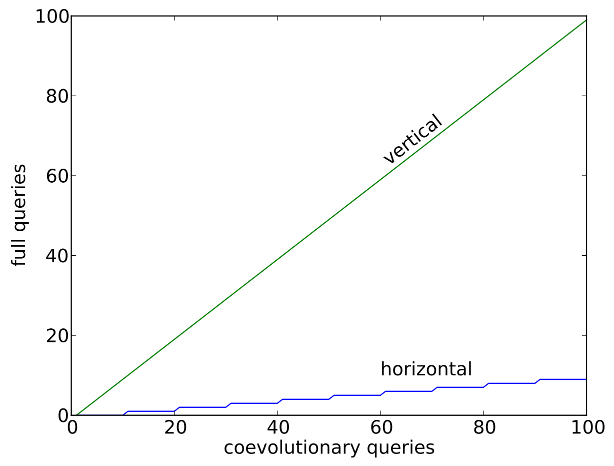


Figure 16: Query exchange rate for best in all categories comparing coevolutionary and full queries. Row size is 15. **doi:**10.5048/BIO-C.2017.1.f16

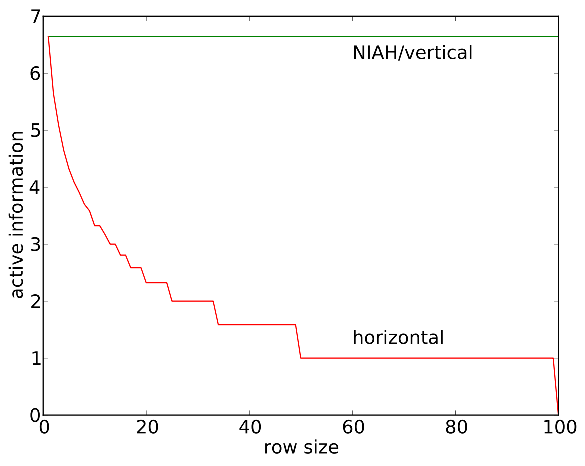


Figure 17: Active information for finding the best in all categories comparing various algorithms and varying row sizes. Number of queries is 100 and size of search space is 1000. **doi:**10.5048/BIO-C.2017.1.f17

6.1.1 NIAH Algorithm

Let F be the event that one of the candidate solutions queried is in the target. The probability of success is by the law of total probability

$$\Pr[S] = \Pr[F]\Pr[S|F] + \Pr[\overline{F}]\Pr[S|\overline{F}]$$

- $\Pr[F] = 1-(1-p^k)^Q$, the probability of set of Q randomly selected candidate solutions containing the target.
- $\Pr[S|F] = 1$, if an acceptable solution is found, this search algorithm will select it.
- $\Pr[\overline{F}] = (1-p^k)^Q$
- $\Pr[S|\overline{F}] = p^k$, if no queried candidates show an acceptable solution, a random candidate solution is chosen, which will pass if all its tests pass.

Thus

$$\begin{aligned}\Pr[S] &= 1-(1-p^k)^Q + (1-p^k)^Q p^k \\ &= 1-(1-p^k)(1-p^k)^Q \\ &= 1-(1-p^k)^{Q+1}\end{aligned}$$

and

$$I_+ = -\log_2 \frac{1-(1-p^k)^{Q+1}}{p^k}. \tag{2}$$

This is used for the plots of the NIAH algorithm in Figures 5, 6, 7, and 8.

6.1.2 Horizontal Algorithms

Given Q queries, the horizontal coevolution algorithm queries $\lfloor \frac{Q}{k} \rfloor$ complete rows. Let F be the event that an acceptable candidate solution is found in those complete rows. Let $x = Q \mod k$, the number of queries performed on the last row before running out of queries. Let G be the event that none of the x test cases on the last row were failures.

$$\Pr[S] = \Pr[F]\Pr[S|F] + \Pr[\overline{F}]\Pr[S|\overline{F}] \tag{3}$$

- $\Pr[F] = 1-(1-p^k)^{\lfloor \frac{Q}{k} \rfloor}$, the probability of set of $\lfloor \frac{Q}{k} \rfloor$ randomly selected candidate solutions containing the target.
- $\Pr[S|F] = 1$, if an acceptable solution is in the completely queried rows, this search algorithm will select it.
- $\Pr[\overline{F}] = (1-p^k)^{\lfloor \frac{Q}{k} \rfloor}$.
- $\Pr[S|\overline{F}] = \Pr[G|\overline{F}]\Pr[S|G,\overline{F}] + \Pr[\overline{G}|\overline{F}]\Pr[S|\overline{G},\overline{F}]$ by the law of total probability.
- $\Pr[G|\overline{F}] = p^x$, the probability of the x queries on the partially queried row passing.
- $\Pr[S|G,\overline{F}] = p^{k-x}$, the probability the remaining tests pass.
- $\Pr[\overline{G}|\overline{F}] = 1-p^x$.
- $\Pr[S|\overline{G},\overline{F}] = p^k$ if none of the completely queried rows are in the target, and the partially queried row contains a failure, a random row is chosen and success is the probability of all of that row's tests passing.

$$\begin{aligned}\Pr[S] &= 1 - (1-p^k)^{\lfloor \frac{Q}{k} \rfloor} \\ &\quad + (1-p^k)^{\lfloor \frac{Q}{k} \rfloor}(p^x p^{k-x} + (1-p^x)p^k) \\ &= 1 - (1-p^k)^{\lfloor \frac{Q}{k} \rfloor} \\ &\quad + (1-p^k)^{\lfloor \frac{Q}{k} \rfloor}(p^x p^{k-x} + (1-p^x)p^k) \\ &= 1 - (1-p^k)^{\lfloor \frac{Q}{k} \rfloor}(1 - 2p^k + p^x p^k) \qquad (4)\end{aligned}$$

As a sanity check, if $Q \mod k = 0$ then

$$\begin{aligned}\Pr[S] &= 1 - (1-p^k)^{\lfloor \frac{Q}{k} \rfloor}(1 - 2p^k + p^0 p^k) \\ &= 1 - (1-p^k)^{\frac{Q}{k}}(1-p^k) \\ &= 1 - (1-p^k)^{\frac{Q}{k}+1}.\end{aligned}$$

When there are no partially queried rows, the performance is the same as a haystack search with less queries. This is used for the plots of the horizontal algorithm in Figures 5, 6, 7, and 8.

6.1.3 Vertical Algorithm

Let F be the event that at least one of the Q queries reveals a passing test.

$$\Pr[S] = \Pr[F]\Pr[S|F] + \Pr[\overline{F}]\Pr[S|\overline{F}] \qquad (5)$$

- $\Pr[F] = 1 - (1-p)^Q$

  The probability of at least one passing is one minus the probability of all tests failing.
- $\Pr[\overline{F}] = (1-p)^Q$
- $\Pr[S|F] = p^{k-1}$

  If choosing a row with one passing test, the probability of the rest passing is one minus the probability of the remaining $k-1$ elements failing.
- $\Pr[S|\overline{F}] = p^k$

  If choosing a row without any queries, the probability is that of a random test case passing.

$$\Pr[S] = (1-(1-p)^q)(1-p)^{k-1} + (1-p)^Q p^k \qquad (6)$$

This is used for the plots of the vertical algorithm in Figures 5, 6, 7, and 8.

6.1.4 So-Far So-Good

Let $f(p,k,x,Q)$ be the probability of the so-far so-good algorithm being on a row with x uncovered passing tests after Q queries, given the probability of a successful test p and number of tests per solution k.

Before any queries, the probability of being in the initial state is 1.

$$f(p,k,0,0) = 1 \qquad (7)$$

Whenever a failing test is found, the algorithm moves to a new row with 0 uncovered values.

$$\begin{aligned}f(p,k,0,Q) &= \sum_{i=0}^{k-1}(1-p)f(p,k,i,Q-1) \qquad (8) \\ &= (1-p)(1-f(p,k,k,Q-1)) \qquad (9)\end{aligned}$$

Whenever a passing test is found, the algorithm stays on the same row

$$f(p,k,x,Q) = pf(p,k,x-1,Q-1). \qquad (10)$$

If a complete acceptable solution is found, the algorithm remains on that row.

$$f(p,k,k,Q) = pf(p,k,k-1,Q-1) + f(p,k,k,Q-1) \qquad (11)$$

As a sanity check we can sum the probability of being in any row after Q queries. This should be one: the algorithm will always be in exactly one row.

$$\begin{aligned}&(1-p)(1-f(p,k,k,Q-1)) \\ &+ \sum_{i=1}^{k-1} pf(p,k,x-1,Q-1) \\ &+ pf(p,k,k-1,Q-1) + f(p,k,k,Q-1) \\ &= 1 - f(p,k,k,Q-1) - p + pf(p,k,k,Q-1) \\ &+ \sum_{i=1}^{k-1} pf(p,k,x-1,Q-1) \\ &+ pf(p,k,k-1,Q-1) + f(p,k,k,Q-1) \\ &= 1 - p + \sum_{i=1}^{k+1} pf(p,k,x-1,Q-1) \\ &= 1 - p + p \\ &= 1\end{aligned}$$

The search succeeds if all of the tests on the current row, including those not yet uncovered pass.

$$\Pr[S] = \sum_{x=0}^{k} f(p,k,x,Q)p^{k-x} \qquad (12)$$

This is used for the plots of the so-far so-good algorithm in Figures 5, 6, 7, and 8.

6.2 Averages

The Irwin-Hall distribution gives the distribution of the sum of n uniform random variables in the range $(0,1)$[32, 33]. The cumulative density function is

$$F_X(x) = \frac{1}{n!}\sum_{k=0}^{\lfloor x \rfloor}(-1)^k \binom{n}{k}(x-k)^n \qquad (13)$$

where n is the number of variables. Let V_i be the ith value in a row, and T the threshold that must be passed:

$$\begin{aligned}\Pr[S] &= \Pr[Average(V_0, V_1, \ldots V_{n-1}, V_n) > T] &(14)\\ &= \Pr[Sum(V_0, V_1, \ldots V_{n-1}, V_n) > nT] &(15)\\ &= 1 - \Pr[Sum(V_0, V_1, \ldots V_{n-1}, V_n) < nT] &(16)\\ &= 1 - F_X(nT) &(17)\end{aligned}$$

Consider the probability of success for performing Q random queries. This is one minus the probability of failing all Q random queries

$$\Pr[S] = 1 - (1 - F_X(n * t))^Q \qquad (18)$$

The performance of the other algorithms proved intractable by analysis. Instead, one million Monte Carlo trials were run to approximate the average. Figures 9, 10, 11 were created using this math and Monte Carlo.

6.3 Pareto

The probability of success for the baseline search is the probability that given two randomly selected rows, the first is dominated by the second. Let k be the number of variables they are compared upon,

$$\Pr[S] = \Pr[B \prec_k A] \qquad (19)$$

where $B \prec_k A$ is that B dominates A over the first k criteria.

Given two iid random numbers, the probability of the first being higher, assuming that probability of equality is negligible, is one half.

$$\Pr[B \prec_1 A] = \frac{1}{2}$$

$$\begin{aligned}\Pr[B \prec_n A] &= \Pr[B \prec_{n-1} A] \Pr[B \prec_n A | B \prec_{n-1} A]\\ &\quad + \Pr[\overline{B \prec_{n-1} A}] \Pr[B \prec_n A | \overline{B \prec_{n-1} A}]\\ &= \frac{1}{2} \Pr[B \prec_{n-1} A] + 0 \Pr[\overline{B \prec_{n-1} A}]\\ &= \frac{1}{2} \Pr[B \prec_{n-1} A]\end{aligned}$$

which trivially produces

$$\Pr[S] = \Pr[B \prec_k A] = \left(\frac{1}{2}\right)^k. \qquad (20)$$

This equation is used to compute the baseline for active information in Figures 13, 14. All other data in those figures was computed by Monte Carlo.

6.4 Best in all Categories

Let n be the size of Ω, the search space. Let k be the size of the rows. Let Q be the number of queries.

Let E be the event that there exists a candidate solution which dominates in all categories. For this to occur, the candidate solution with the largest score in the first category must also have the largest score in the other categories. Given n i.i.d random variables, the probability of any particular one being the largest is $\frac{1}{n}$. Thus

$$\Pr[E] = \left(\frac{1}{n}\right)^{k-1} \qquad (21)$$

6.4.1 NIAH

$$\begin{aligned}\Pr[S] &= \Pr[S|E]\Pr[E] + \Pr[S|\overline{E}]\Pr[\overline{E}] &(22)\\ &= \Pr[S|E]\Pr[E]. &(23)\end{aligned}$$

It is impossible for multiple solutions to dominate all other solutions, so the probability of selecting the solution after Q attempts is

$$\Pr[S|E] = \frac{Q}{n}. \qquad (24)$$

This gives

$$\Pr[S] = \frac{Q}{n}\left(\frac{1}{n}\right)^{k-1} = \frac{Q}{n^k} \qquad (25)$$

which is the basis of the NIAH performance in Figures 15, 16, and 17.

6.4.2 Vertical Algorithm

This algorithm will succeed if it queries the row containing the target. This is exactly the same as for the NIAH algorithm. Thus the performance NIAH and the Vertical algorithm is the same in in Figures 15, 16, and 17.

6.4.3 Horizontal Algorithm

This algorithm will query $\lceil \frac{Q}{k} \rceil$ values in the first column. The value with the highest probability of being in the target will be selected, and if one of those rows is the target it will succeed. This gives

$$\Pr[S] = \frac{\lceil \frac{Q}{k} \rceil}{n}\left(\frac{1}{n}\right)^{k-1} = \frac{\lceil \frac{Q}{k} \rceil}{n^k} \qquad (26)$$

which is the basis of the horizontal algorithm performance in Figures 15, 16, and 17.

1. Hillis WD (1990) Co-evolving Parasites Improve Simulated Evolution as an Optimization Procedure. Physica D: Nonlinear Phenomena 42:228–234. doi:10.1016/0167-2789(90)90076-2
2. Sims K (1994) Evolving 3D Morphology and Behavior by Competition. Artificial Life 1:353–372. doi:10.1162/artl.1994.1.353
3. Darwen PJ (2001) Why Co-Evolution beats Temporal

Difference learning at Backgammon for a linear architecture, but not a non-linear architecture. In Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546), volume 2, 1003–1010. IEEE. doi:10.1109/CEC.2001.934300

4. Chellapilla K, Fogel DB (2001) Evolving an Expert Checkers Playing Program without Using Human Expertise. IEEE Transactions on Evolutionary Computation 5:422–428. doi:10.1109/4235.942536
5. Fogel DB, Hays TJ, Hahn SL, Quon J (2004) A Self-Learning Evolutionary Chess Program. Proceedings of the IEEE 92:1947–1954. doi:10.1109/JPROC.2004.837633
6. Wolpert DH, Macready WG (1995) No free lunch theorems for search. Technical report, Technical Report SFI-TR-95-02-010, Santa Fe Institute
7. Wolpert DH, Macready WG (1997) No Free Lunch Theorems for Optimization. IEEE Transactions on Evolutionary Computation 1:67–82. doi:10.1109/4235.585893
8. Wolpert DH, Macready WG (2005) Coevolutionary Free Lunches. IEEE Transactions on Evolutionary Computation 9:721–735. doi:10.1109/TEVC.2005.856205
9. Ficici SG (2004) Solution Concepts in Coevolutionary Algorithms. Dissertation, Brandeis University
10. Popovici E, Bucci A, Wiegand RP, De Jong ED (2012) Coevolutionary principles. In Handbook of Natural Computing, 987–1033. Springer
11. Yang XS (2012) Free lunch or no free lunch: that is not just a question? International Journal on Artificial Intelligence Tools 21
12. Mitchell TM (1990) The Need for Biases in Learning Generalizations. In J Shavlik, T Dietterich, editors, Readings in Machine Learning, Morgan Kauffmann Series in Machine Learning, 184–190 (Originally published as a Rutgers Technical Report, May 1980)
13. Schaffer C (1994) A Conservation Law for Generalization Performance. In W Cohen, H Willian, editors, Proceedings of the Eleventh International Machine Learning Conference, 259 – 265
14. Duda RO, Hart PE, Stork DG (2000) Pattern Classification. Wiley-Interscience, second edition
15. Pepyne DL, Ho YC (2001) Simple Explanation of the No Free Lunch Theorem of Optimization. In Proceedings of the 40th IEEE Conference on Decision and Control (Cat. No.01CH37228), volume 5, 4409–4414. IEEE. doi:10.1109/.2001.980896
16. Christensen S, Oppacher F (2001) What can we learn from No Free Lunch? A First Attempt to Characterize the Concept of a SearchableFunction. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)
17. Service TC, Tauritz DR (2009) Free Lunches in Pareto Coevolution. Proceedings of the 11th Annual conference on Genetic and evolutionary computation - GECCO '09 1721–1727. doi:10.1145/1569901.1570132
18. Service TC, Tauritz DR (2008) A No-Free-Lunch Framework for Coevolution. In Proceedings of the 10th annual conference on Genetic and evolutionary computation - GECCO '08, 371–378. ACM Press, New York, New York, USA. doi:10.1145/1389095.1389163
19. Corne DW, Knowles JD (2003) Some multiobjective optimizers are better than others. In The 2003 Congress on Evolutionary Computation, 2003. CEC '03., volume 4, 2506–2512. IEEE. doi:10.1109/CEC.2003.1299403
20. Droste S, Jansen T, Wegener I (1998) Perhaps Not A Free Lunch But At Least A Free Appetizer. Technical Report September, University of Dortmund, Dortmund, Germany
21. Streeter MJ (2003) Two Broad Classes of Functions for which a No Free Lunch Result Does Not Hold. In Genetic and Evolution Computation, 210–210. Springer. doi:10.1007/3-540-45110-2_15
22. Whitley D (1999) A Free Lunch Proof for Gray versus Binary Encodings. In W Banzhaf, J Daida, AE Eiben, MH Garzon, V Honavar, M Jakiela, RE Smith, editors, Proceedings of the Genetic and Evolutionary Computation Conference, volume 1, 726–733. Morgan Kaufmann
23. Droste S, Jansen T, Wegener I (2002) Optimization with Randomized Search Heuristics—The (A)NFL theorem, Realistic Scenarios, and Difficult Functions. Theoretical Computer Science 287:131–144. doi:10.1016/S0304-3975(02)00094-4
24. Corne DW, Knowles JD (2003) No Free Lunch and Free Leftovers Theorems for Multiobjective Optimisation Problems. In Evolutionary Multi-Criterion Optimization (EMO 2003) Second International Conference, 327–341. Springer LNCS
25. Service TC (2009) Unbiased Coevolutionary Solution Concepts. Proceedings of the tenth ACM SIGEVO workshop on Foundations of genetic algorithms - FOGA '09 121. doi:10.1145/1527125.1527142
26. Woodward JR, Neil JR (2003) No Free Lunch, Program Induction and Combinatorial Problems. Genetic Programming Proceedings of Euro 2610:475–484. doi:10.1007/3-540-36599-0_45
27. Dembski WA, Marks II RJ (2009) Conservation of Information in Search: Measuring the Cost of Success. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans 39:1051–1061. doi:10.1109/TSMCA.2009.2025027
28. Schneider TD (2000) Evolution of Biological Information. Nucleic Acids Research 28:2794–2799. doi:10.1093/nar/28.14.2794
29. Spector L, Clark DM, Lindsay I, Barr B, Klein J (2008) Genetic Programming for Finite Algebras. In Proceedings of the 10th annual conference on Genetic and evolutionary computation - GECCO '08. ACM Press, New York, New York, USA. doi:10.1145/1389095.1389343
30. Dawkins R (1996) The Blind Watchmaker: Why The Evidence Of Evolution Reveals A Universe Without Design. Norton, New York
31. Marks II RJ (2009) Handbook of Fourier Analysis & Its Applications. Oxford University Press, Oxford; New York
32. Hall P (1927) The Distribution of Means for Samples of Size N Drawn From a Population in which the Variate Takes Values Between 0 and 1, All Such Values Being Equally Probable. Biometrika 19:240–245
33. Irwin JO (1927) On the Frequency Distribution of the Means of Samples from a Population Having any Law of Frequency with Finite Moments, with Special Reference to Pearson's Type II. Biometrika 19:225–239