On the Need for Random Baseline Comparisons in Metaheuristic Search

Daniel S. Soper
Information Systems and Decision Sciences Department
California State University, Fullerton
dsoper@fullerton.edu

Abstract

A wide variety of organizations now regularly rely on established metaheuristic search algorithms in order to find solutions to otherwise intractable optimization problems. Unfortunately, neither the developers of these algorithms nor the organizations that rely on them typically assess the algorithms' performance against a baseline random search strategy, opting instead to compare a specific algorithm's performance against that of other metaheuristic search algorithms. This paper reveals the folly of such behavior, and shows by means of an optimization case study that simple random or nearly random search algorithms can, in certain circumstances, substantially outperform several of the most widely used metaheuristic search algorithms in finding solutions to optimization problems. The implications of the observed results for both organizations and researchers are presented and discussed.

1. Introduction

For the past several decades many organizations, acting in a wide variety of industries, have relied on heuristic search algorithms in order to find near-optimal solutions to problems that would otherwise be infeasible using current technologies [1]. The infeasibility of such problems emerges from computational or temporal limitations that prevent an organization from examining every possible solution in a large decision space. Consider, for example, the case of companies such as United Parcel Service (UPS), DHL Express, and FedEx, each of which must deliver hundreds of thousands or even millions of packages every day. Ideally, each of these companies would like to minimize the total costs associated with delivering its packages with a view toward maximizing corporate profits. Unfortunately, identifying optimal routes for their legions of delivery drivers that simultaneously consider factors such as fuel costs, total time, distance travelled, traffic flows, weather conditions, and promised delivery windows makes this an extremely complicated, NP-hard optimization problem [2]. The intractability of this problem on a large scale is, in fact, so well-known that it has been given its own name – the Vehicle Routing Problem – and has been studied in the optimization literature for nearly 60 years [3].

When an organization is faced with an unavoidable NP-hard optimization problem (such as the Vehicle Routing Problem), it has no choice but to acknowledge the intractable nature of the situation, and turn its attention to algorithms that seek to provide near-optimal solutions to the problem within a timeframe that is acceptable in light of the organization's objectives and constraints. Among the wide variety of algorithms that have been designed to address NP-hard optimization problems, many of the most effective and widely used fall under the umbrella of what are known as metaheuristic search algorithms; i.e., algorithms that use a rule-based, iterative process to explore a decision space with the goal of efficiently finding a near-optimal solution to the underlying problem [4]. Some of the most popular and long-lived heuristic search algorithms include Tabu search [5] and Backtracking [6], with more recent entrants including Ant Colony Optimization [7] and Cuckoo Search [8], among many others.

Implicit in organizational adoption and use of metaheuristic search algorithms such as those noted above is the assumption that the algorithms will yield solutions that offer the organization at least some degree of improvement over what could be expected from entirely random or nearly random search methods. For example, imagine a decision space that contains one million possible solutions, each of which requires one second of computational time to evaluate. If an organization's constraints are such that it can wait for only 1,000 seconds, then it will be computationally possible for the organization to explore only 1% of the overall decision space before it must settle on a solution and proceed with its other tasks. In a scenario such as this, the organization might reasonably adopt a metaheuristic search algorithm in order to find the

URI: http://hdl.handle.net/10125/50045
ISBN: 978-0-9981331-1-9

best solution possible given its time constraints. By doing so, however, the organization is implicitly assuming that its chosen metaheuristic search algorithm will, on average, identify better solutions than could otherwise be obtained by means of a random or nearly random search strategy.

The key problem with this assumption is that it is almost never actually tested in practice. Clearly, the performance of a random search strategy provides a natural and fundamental baseline against which any metaheuristic search algorithm can be evaluated, but if the optimization literature serves as any sort of guide, then it must be concluded that the performance of proposed metaheuristic search algorithms is almost never compared against that of a random baseline. As evidence of this phenomenon, consider that none of the authors of the foundational papers or books in which any of the most popular metaheuristic search algorithms[1] were first described took the time to compare the performance of their newly proposed algorithms to a random search strategy [5, 9-18]. Instead, the common practice among researchers who are proposing a new metaheuristic search algorithm is simply to compare the performance of their proposed algorithm against that of other metaheuristic search algorithms.

Incorporating elements of randomness into metaheuristic search algorithms has been acknowledged by several authors to impart desirable characteristics upon those algorithms. For example, it has been noted that adding a random restart feature to certain metaheuristic search algorithms can help those algorithms to escape from local optima, thereby improving the algorithms' chances of finding a better solution [18]. It has also been shown that using a multi-start approach that incorporates random elements can improve overall metaheuristic search performance for certain combinatorial optimization problems [19]. Although such approaches reveal some of the advantages that randomness can impart, incorporating random elements into a metaheuristic search algorithm is nevertheless fundamentally different from considering the performance of a fully random or nearly random approach. The general indifference and lack of attention in the metaheuristic optimization literature toward the comparative performance of random search strategies serves as the primary motivation for this paper, and raises the following general research question:

Can simple random search strategies outperform well-established metaheuristic search strategies?

More particularly, this paper inquires into the following specific research question:

Under what conditions can random search algorithms outperform well-established metaheuristic search algorithms in a process optimization task?

The balance of this paper seeks to provide some much-needed insights into these questions. In the next section, a multidimensional process optimization problem is described that will serve as the basis of our investigation into the comparative performance of several random and metaheuristic search strategies. Section 3 briefly describes each of the random and metaheuristic search algorithms used in the study, as well as the means by which the performance of the various algorithms were evaluated and compared. Section 4 describes the results of the analysis, and discusses the implications of those results for both managers and researchers. The paper concludes with Section 5, which provides a brief summary, a discussion of the paper's limitations, and a few final remarks.

2. A process optimization problem

It is now well-established that an organization's long-term prospects for success are intimately linked to the organization's ability to monitor, adjust, and optimize its business processes on an ongoing basis [20]. Organizations are typically characterized by a wide variety of interacting processes, each of which can be configured in a number of different ways. Since each process may interact with many other processes, identifying the best way to configure each of these interacting processes such that together they maximally contribute to some higher-level goal (e.g., profit maximization, efficiency, etc.) is a fundamental component of effective business process management [21].

In this paper, we consider a scenario in which an organization is attempting to optimize three interacting processes, each of which can be configured in 100 different ways. Different combinations of these process configuration possibilities either improve or erode the efficiency of the overall process system. Since each of the three processes in this scenario can be configured in 100 different ways, there are a total of $100^3 = 1$ million distinct ways in which the three-process system can be configured. To model this situation, we adopt a geometric framework in which

[1] See [18] for a list of the most popular metaheuristic search algorithms.

each process and its 100 possible means of configuration is represented as a parameter in a three-dimensional Euclidean space. In this way, any of the 1 million possible system configurations can be conceptualized as a distinct location within the three-dimensional space. For example, if the three processes are labeled *X*, *Y*, and *Z*, then the spatial coordinates (25, 50, 75) would indicate that process *X* is using configuration option 25 (out of 100 possible configuration options), process *Y* is using configuration option 50, and process *Z* is using configuration option 75.

To perturb the three-dimensional space, we adopt a model based on Newtonian gravity in which each of the 1 million spatial locations is occupied by an object having a particular mass. By default, each object was initially assigned a unity mass of 1 kilogram. Twenty objects were then chosen at random, each of which was assigned a random integer mass between 2 and 100 kilograms, inclusive[2]. The resulting space thus possessed a quantifiable level of gravitational acceleration at each spatial location, and contained both a single global maximum and a variety of local optima. For purposes of this study, the level of gravitational acceleration at any location within the three-dimensional space was taken to represent the degree of overall system efficiency that the organization would enjoy if it adopted the three process configuration options represented by the spatial coordinates for the location in question.

The nearly 8 trillion calculations that were necessary in order to compute the total level of gravitational acceleration at each of the 1 million points within the three-dimensional space was laboriously performed using the *DIRECT* algorithm created by the N-Body Shop at the University of Washington [22]. For the purpose of these calculations Newton's universal gravitational constant was set equal to 1.0. After completing the calculations, the mean gravitational acceleration (i.e., the average level of efficiency) for the decision space was determined to be 186.401 (stdev = 44.627), with a minimum value of 3.469 (representing the least efficient combination of process configurations) and a maximum value of 277.149 (representing the most efficient combination of process configurations). The spatial coordinates for the optimal combination of process configurations were observed to be ($X = 74$, $Y = 91$, $Z = 65$). Figure 1 below provides two different depictions of the decision space, with the leftmost image showing a random sample of 100,000 (i.e., 10%) of the possible process configurations, and the rightmost image showing the areas of strongest gravitational acceleration (i.e., high efficiency) within the decision space.

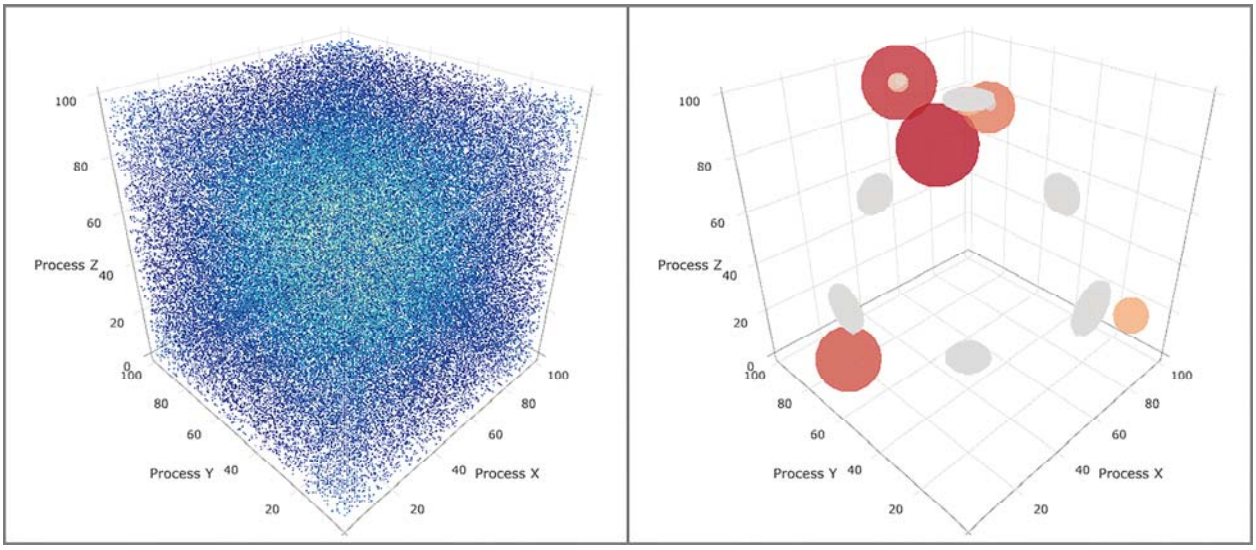

Figure 1. Two views of the process optimization decision space. *Left*: random sample of 100,000 possible process configurations; *Right*: areas of high process configuration efficiency.

[2] The spatial locations and masses of these 20 randomly chosen objects were: (0, 76, 43): 22 kg, (6, 60, 21): 58 kg, (8, 90, 30): 51 kg, (14, 18, 93): 12 kg, (23, 29, 17): 43 kg, (27, 61, 22): 78 kg, (46, 76, 99): 64 kg, (52, 18, 46): 67 kg, (67, 67, 37): 81 kg, (70, 39, 90): 29 kg, (71, 28, 51): 56 kg, (72, 51, 25): 32 kg, (74, 90, 65): 80 kg, (75, 44, 28): 12 kg, (76, 12, 82): 48 kg, (79, 4, 29): 42 kg, (80, 16, 89): 68 kg, (81, 83, 39): 85 kg, (90, 90, 78): 73 kg, and (99, 87, 74): 66 kg.

Having described the geometric framework underlying the three-process optimization problem, we next turn our attention to the random and metaheuristic search algorithms that were used to explore the decision space, as well as the means by which the performance of the algorithms was evaluated and compared. It will serve the reader well to keep this geometric framework in mind, as it will be referred to regularly in the following section.

3. Search algorithms and analytical methods

We begin this section by describing the two random search algorithms and the two metaheuristic search algorithms that were used to search the process optimization decision space described in the previous section. The goal of each of these algorithms was to find the combination of process configurations that would yield the highest level of overall system efficiency after having examined a specified number of possibilities. After describing the four search algorithms, we will then proceed to a detailed description of the analytical methods that were used to evaluate and compare the performance of those algorithms.

3.1. Random search algorithms

Two random or nearly random search algorithms were used in the analysis: random search (RND) and self-avoiding random search (SA-RND).

3.1.1. Random search (RND). The random search algorithm is by far the simplest possible search algorithm to implement, describe, and understand. Recalling that the decision space is characterized by three processes (X, Y, and Z), each of which can be configured in 100 different ways, the random search algorithm works by using a uniform random number generator to select a random integer between 0 and 99 (inclusive) for each of the three processes. Together, these three randomly chosen values represent a unique location in the decision space; e.g., (X = 36, Y = 24, Z = 18). The algorithm then evaluates the overall efficiency of the system at that location, and saves the resulting solution only if it is the best solution that has been thus far observed, overwriting the prior best solution. This process is then repeated until a specified number of possible solutions have been examined, after which the best overall solution identified during the search process is returned.

3.1.2. Self-avoiding random search (SA-RND). After the fully random search algorithm, the self-avoiding random search algorithm is next in rank with respect to its simplicity. The self-avoiding random search algorithm is, in fact, identical to the random search algorithm described in the previous subsection, with one important addition – that of a memory. Whereas in a (fully) random search it is perfectly possible for the algorithm to examine the same location in the decision space multiple times, in a self-avoiding random search, the algorithm keeps a record of all of the locations that it has previously examined. If by random chance the algorithm happens to select a previously visited location for its next inquiry, then a new set of location coordinates is randomly generated until a location is identified that has not already been examined. In this way, the algorithm avoids visiting any given location more than once.

3.2. Metaheuristic search algorithms

We next turn our attention to the two metaheuristic search algorithms that were used in the analysis: Tabu search (TS-100 & TS-1000) [5] and Backtracking search (BT) [6]. These two algorithms were chosen for inclusion in the analysis because they are two of the best-known, well-established, and well-studied metaheuristic search algorithms known to exist (each having existed for several decades), and because they are among the most widely used metaheuristic search algorithms by modern organizations seeking to solve complex optimization problems. Before proceeding to the algorithm descriptions, however, it is first necessary to define three concepts that are central to both of the algorithms: *neighbors*, *moves*, and *neighborhoods*.

Recalling the geometric framework described in the previous section, a *neighbor* is defined as a location (i.e., a possible solution) that is geometrically adjacent to the current location. A *move*, then, is an action in which the focus of attention shifts from the current location to one of the current location's neighbors. The set of neighbors for the current location is referred to as its *neighborhood*. Figure 2 below depicts these concepts in two dimensions.

In the example illustrated in Figure 2, imagine that the focus of a search algorithm is currently on node 13. From this location, the algorithm can move to one of node 13's neighbors (i.e., nodes 7, 8, 9, 12, 14, 17, 18, or 19), which together constitute its neighborhood. As can be readily seen in this illustration, the composition of the current neighborhood changes with every move. The principles of Euclidean geometry allow the concepts of neighbors, moves, and neighborhoods to be readily extended into higher-dimensional spaces,

such as with the current study wherein the metaheuristic search algorithms are operating within a three-dimensional decision space.

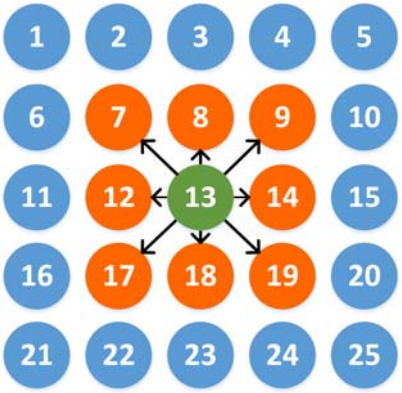

Figure 2. A two-dimensional illustration of moves, neighbors, and neighborhoods in metaheuristic search algorithms.

3.2.1. Tabu search (TS-100 & TS-1000). The Tabu search algorithm begins by examining a specified or randomly chosen location within the decision space, as well as all of the potential solutions in the initial location's neighborhood. The best-performing solution thus far observed is then recorded, after which the algorithm moves to the neighbor that offers the best available solution. All of the previously unexamined neighbors in the new location's neighborhood are then examined, the overall best solution thus far observed is updated if necessary, and the algorithm again moves to the best-performing location in the neighborhood. This process continues until a specified number of locations have been examined, or until some other stopping criterion has been met. Critically, as the algorithm moves from one location to the next it constantly constructs and updates its *Tabu list* – a memory structure that keeps track of recently visited locations within the decision space. Returning to any location on the Tabu list is disallowed (hence the term *Tabu*), and this prevents the algorithm from becoming stuck at a local optimum. The Tabu list is typically of a predefined size (e.g., 1,000 locations), and behaves as a queue, such that when the list is full, the oldest item on the list is dropped in order to make room for the newest item. In its simplest form, the Tabu search algorithm maintains only a short-term Tabu list (such as that described above), but rule-based intermediate-term and long-term memory structures are also possible in more sophisticated versions of the algorithm, the goals of which are typically to diversify the search, or intensify the search in specific areas of the decision space.

The use of a Tabu list introduces several interesting properties into the behavior of the search algorithm. First, the definition of what constitutes the neighborhood for a specific location depends on the composition of the Tabu list. Referring back to Figure 2, after examining node 13 and its neighbors, the algorithm might decide that its next move will be to node 14. Before moving, node 13 and all of its neighbors (except node 14) would be added to the Tabu list. Upon arriving at node 14, the neighborhood would consist only of nodes 10, 15, and 20, since all of node 14's other neighbors are currently marked as Tabu. Second, because the Tabu list is typically of a finite size, it is possible for the algorithm to revisit previously examined locations within the decision space once those locations have expired from the Tabu list. This can encourage the algorithm to explore new paths through the decision space beginning at locations that were initially ignored due to better options being available in the neighborhood at the time of the previous visit.

For the current study, two variants of the Tabu search algorithm were included in the investigation, one of which used a Tabu list whose maximum size was constrained to 100 locations, and the other of which used a Tabu list with a maximum size of 1,000 locations. These variants are later referred to as TS-100 and TS-1000, respectively. Since little guidance exists in the literature regarding the ideal size of a Tabu list, these two variants of the algorithm were included in the current study for purposes of methodological rigor.

3.2.2. Backtracking search (BT). The Backtracking search algorithm is one of the oldest, most well-established, and widely used metaheuristic search algorithms [6]. As with Tabu search, the Backtracking search algorithm begins by examining a specified or randomly chosen location within the decision space, as well as all of the potential solutions in the initial location's neighborhood. The best-performing solution thus far observed is recorded, after which the Backtracking algorithm moves to the neighbor whose solution offers the most improvement over that of the current location. All of the previously unexamined neighbors of the new location are then evaluated, after which the best solution thus far observed is updated if necessary, and the algorithm again moves to the neighbor whose solution offers the most improvement over that of the current location.

If at any time while the algorithm is searching the decision space two or more neighbors happen to offer the same degree of maximal improvement over the current location, then one of the neighbors is chosen at random to be the destination of the next move. If none

of the locations in the neighborhood offers any improvement over the current location, then the algorithm steps backwards to the previous location on the path (hence the term *Backtracking*), and then moves to the location that offers the second-best degree of improvement over the current location's level of performance. If all available paths originating from a specific location have been exhausted, then the algorithm steps backwards once again to the previous location on the path, and continues searching. If all possible paths have been exhausted and the algorithm has stepped backwards all the way to its initial point of origin, then a new origin location within the decision space is chosen at random, and the search process begins again from that point. As with Tabu search, the Backtracking search algorithm continues until it has examined a specified number of possible solutions, or until some other stopping criterion has been reached.

3.3. Analytical methods

Having described the two random and two metaheuristic search algorithms that were used in the analysis, we now proceed to describe the means by which the performance of those algorithms was evaluated and compared. The general strategy for analyzing the performance of the various algorithms was to allow each algorithm to repeatedly explore the decision space described in Section 2, with a view toward identifying the most efficient combination of configurations for processes *X*, *Y*, and *Z*. During each trial, the algorithms were allowed to explore a specific number of locations within the decision space, with the number of locations explored being assigned from a set of 13 equidistantly spaced values on a logarithmic scale ranging from 0.01% to 10% of the total number of locations in the decision space[3]. Recalling that the decision space contained a total of 1 million unique locations, the algorithms were thus constrained to examining as few as 100 locations per trial, to as many as 100,000 locations per trial. Using the number of locations explored as a stopping criterion provided an equitable basis of comparison by which the performance of the algorithms could be observed as they were allowed to explore greater and greater proportions of the overall decision space.

Each algorithm was subjected to 500 separate trials for each of the 13 stopping criteria options described above. For each combination of trial and stopping criterion, each algorithm was assigned the same, randomly chosen starting location within the decision space. This approach was taken in order to ensure equitability among the four search algorithms by preventing any particular algorithm from having an advantage in a particular trial due simply to its having been assigned a propitious random starting location. With 500 trials per stopping criterion, 13 different stopping criteria, and five different algorithms or algorithm variants, a total of 32,500 distinct trials were conducted in the study.

The overall performance of each algorithm for each stopping criterion option was measured as the mean of the best solution found by the algorithm during each of the 500 trials. Comparisons between the levels of performance of any pair of algorithms within a particular stopping criterion option was assessed by means of a series of two-tailed, paired t-tests [23]. The relative performance of the set of random search algorithms vs. that of the set of metaheuristic search algorithms was also assessed by computing the overall average level of performance of the random and metaheuristic algorithms within each of the stopping criterion options, and then using a Welch t-test to evaluate the extent to which the two levels of performance were statistically different from one another [24]. The results of all of these analyses are presented and discussed in the following section.

4. Results and discussion

We begin the presentation of our results with Table 1 below, which shows the average level of performance for each algorithm within each stopping criterion, as well as the average level of performance for the set of random algorithms and the set of metaheuristic algorithms. When considering these values, it may be useful to recall from Section 2 that the level of performance among the complete set of 1 million locations in the decision space ranged from 3.469 to 277.149, with the average level of performance being 186.401 (stdev = 44.627).

Many interesting insights can be gained by carefully examining the values reported in Table 1. First, the overall average performance of the self-avoiding random search algorithm (SA-RND) was slightly higher than that of the (fully) random search algorithm (RND). This accords well with what one should intuitively expect, since by avoiding any locations that it has previously visited, the self-avoiding random search should, on average, visit more unique locations during each trial than its fully random

[3] 13 equidistantly spaced values were considered in order to provide more detailed insights into the comparative performance of the search algorithms than could otherwise have been obtained if only the "natural" points of consideration on the logarithmic scale (i.e., 0.01%, 0.10% 1.00%, and 10.00%) had been evaluated.

counterpart (which may, by random chance, visit the same location more than once). Second, among the metaheuristic search algorithms, the Backtracking search algorithm (BT) generally outperformed both variants of the Tabu search algorithm (TS-100 & TS-1000), thereby suggesting that the Backtracking approach may be more suitable for this type of optimization problem than Tabu search. Third, recalling that the two variants of the Tabu search algorithm varied only according to the size of their Tabu lists, the variant with the shorter Tabu list (TS-100) outperformed the variant with the longer Tabu list (TS-1000) when constrained to searching approximately 0.18% to 0.32% of the overall decision space. When the algorithm was allowed to search 1.00% of the overall decision space or more, however, the variant with the longer Tabu list (TS-1000) outperformed the variant with the shorter Tabu list (TS-100). Insights such as these may prove useful to researchers who are attempting to formulate recommendations about the appropriate size of a Tabu list.

Table 1. Comparative performance of random and metaheuristic search algorithms.

Number of Locations Explored (% of Decision Space)	Random Algorithms		Metaheuristic Algorithms			Mean Performance by Algorithm Category	
	RND (R1)	SA-RND (R2)	TS-100 (M1)	TS-1000 (M2)	BT (M3)	Random	Metaheuristic
100 (0.010%)	251.71 $^{M1, M2, M3}$	251.77 $^{M1, M2, M3}$	241.72	241.72	241.67	251.74***	241.71
178 (0.018%)	253.44	253.64	254.28 $^{R1, R2}$	254.28 $^{R1, R2}$	254.28 $^{R1, R2}$	253.54	254.28***
316 (0.032%)	254.73	254.65	256.51 $^{R1, R2}$	256.51 $^{R1, R2}$	256.51 $^{R1, R2}$	254.69	256.51***
562 (0.056%)	255.57	255.46	256.58 $^{R1, R2}$	256.58 $^{R1, R2}$	256.58 $^{R1, R2}$	255.52	256.58***
1,000 (0.100%)	255.96	256.05	256.52 $^{R1, R2}$	256.52 $^{R1, R2}$	256.59 $^{R1, R2}$	256.00	256.54***
1,778 (0.178%)	256.21	256.36	256.47 ^{R1}	256.46 ^{R1}	256.58 ^{R1}	256.28	256.50***
3,162 (0.316%)	256.49	256.63	256.57	256.54	256.73	256.56	256.61
5,623 (0.562%)	256.77 $^{M1, M2}$	256.84 $^{M1, M2}$	256.49	256.49	256.88	256.80*	256.62
10,000 (1.000%)	257.15 ^{M1}	257.44 $^{M1, M2}$	256.51	257.00	257.68 ^{R1}	257.29	257.06
17,783 (1.778%)	258.07 ^{M1}	258.14 ^{M1}	256.51	257.93	258.94 $^{R1, R2}$	258.10	257.79
31,623 (3.162%)	258.98 $^{M1, M2}$	258.81 $^{M1, M2}$	256.46	258.02	259.29	258.90***	257.92
56,234 (5.623%)	260.59 $^{M1, M2}$	260.62 $^{M1, M2}$	256.50	258.15	260.66	260.61***	258.44
100,000 (10.000%)	262.89 $^{M1, M2}$	263.92 $^{M1, M2, M3}$	256.45	257.90	262.50	263.41***	258.95
Overall Mean:	**256.81**	**256.95**	**255.20**	**255.70**	**256.53**	**256.88**	**255.81**

$^{M1, M2, M3}$ indicates that a random algorithm outperformed a metaheuristic algorithm at the $p < 0.05$ level or better (^{M1} = TS-100, ^{M2} = TS-1000, ^{M3} = BT).
$^{R1, R2}$ indicates that a metaheuristic algorithm outperformed a random algorithm at the $p < 0.05$ level or better (^{R1} = RND, ^{R2} = SA-RND).
* indicates that one algorithm category statistically outperformed the other algorithm category (* $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$)

We will next direct our attention toward answering this paper's primary research question; to wit:

Under what conditions can random search algorithms outperform metaheuristic search algorithms in a process optimization task?

The values reported in Table 1 show that there were many situations in which the random search algorithms outperformed the metaheuristic search algorithms, and vice versa. This observation alone casts a shadow of doubt over the assumption of metaheuristic superiority. Further, examining the mean performance of the various algorithms by category reveals that when the area to be searched was less than approximately 0.018% of the total size of the decision space, the random search algorithms statistically outperformed the metaheuristic search algorithms when trying to find a near-optimal solution to the process optimization task. The random search algorithms again showed themselves to be superior when the area to be searched was larger than approximately 0.316% of the total size of the decision space. Put differently, the data show that the metaheuristic search algorithms outperformed their random counterparts only within a very narrow range. In all other situations, the random search algorithms were, on average, observed to consistently outperform the metaheuristic search algorithms. These results are depicted graphically in Figure 3 below.

The implications of Figure 3 for organizational decision-making should not be ignored, particularly if similar patterns are observed to exist among other optimization problems. Put simply, the figure shows

that an organization struggling with this particular optimization problem would do well to rely on metaheuristic search algorithms if – and only if – its constraints were such that it could afford to explore only 0.018% to 0.316% of the decision space before settling on a solution. In all other cases, the figure suggests that the organization would be better off by relying on random search algorithms. The extent to which this pattern of results would hold in the context of other optimization problems is unknown, but until further evidence is collected and analyzed, organizations would be well-advised to evaluate the performance of any metaheuristic search algorithm upon which they may rely against that of a random baseline. Further, it should be the responsibility of any researcher who is proposing a new metaheuristic search algorithm to compare the performance of the proposed algorithm against a similar random baseline.

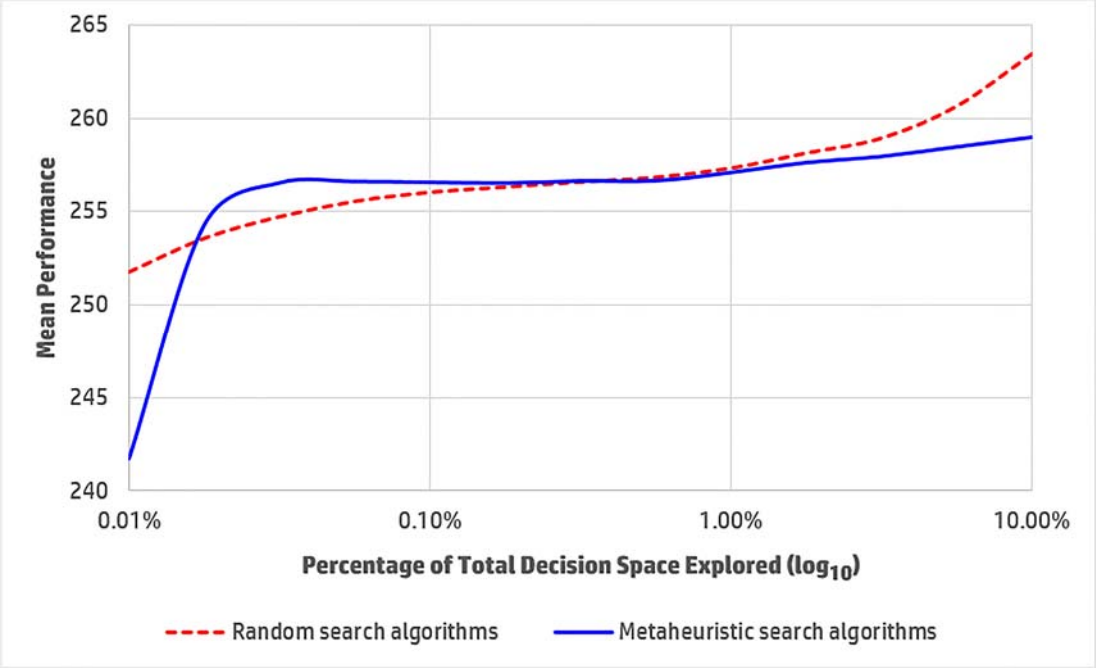

Figure 3. Performance of random and metaheuristic search algorithms on a logarithmic scale.

5. Summary, limitations, and concluding remarks

The current paper reported on a study aimed at investigating whether and under what conditions random search algorithms can outperform metaheuristic search algorithms when attempting to find a near-optimal solution to an optimization problem. Using a process optimization task, the performance of two random search algorithms was compared against that of two popular metaheuristic search algorithms under conditions in which the various algorithms were allowed to search between 0.01% and 10.00% of the overall decision space. The results of the study demonstrate conclusively that under certain conditions, random search algorithms can regularly and consistently outperform several of the most well-known and widely used metaheuristic search algorithms in finding a near-optimal solution to an optimization problem. Specifically, the results showed that the random search algorithms statistically outperformed the metaheuristic algorithms in the process optimization task when the percentage of the decision space explored was either less than approximately 0.018% of the total decision space or greater than approximately 0.316% of the total decision space. Put differently, the metaheuristic search algorithms were only able to consistently outperform the random search algorithms when they were constrained to searching within a relatively narrow range of percentages of the overall size of the decision space.

One possible explanation for the observed results is that due to their uniformly random nature, the random algorithms are likely to do at least *some* exploring in all regions of the decision space, despite doing so with a coarser degree of granularity than their metaheuristic counterparts. This supposition would intuitively hold when the number of locations that the algorithms are allowed to explore is both very small

and comparatively large. The metaheuristic algorithms, by contrast, are designed to spend a great deal of time carefully examining all of the possibilities within smaller regions of the decision space. Although it seems clear that under certain conditions this design characteristic allows the metaheuristic algorithms to outperform the random algorithms, the data show that for the type of optimization problem described in this paper, the benefits to be gained from a fine-toothed examination of specific regions of the decision space do not outweigh the costs of failing to examine a larger geographical area within that decision space, albeit with a coarser degree of granularity.

The findings reported in this paper are critically important to the optimization literature because very few organizations – and even fewer developers of popular metaheuristic search algorithms – ever take the time to actually compare the performance of a chosen or proposed metaheuristic algorithm against the performance of a random baseline. Instead, the implicit assumption has been that metaheuristic search algorithms outperform random search algorithms in optimization tasks. The results presented in this paper strongly challenge the veracity of this assumption of universal metaheuristic superiority, and speak loudly to the fact that sophisticated metaheuristic search algorithms may not perform as well as a simple random search in certain types of optimization tasks.

The limitations of the current study are, of course, quite clear, but nevertheless deserve specific acknowledgement. First, the performance of the various algorithms was assessed using only one optimization problem. It remains to be seen whether random search algorithms would perform similarly when challenged with other optimization problems. Second, only two random and two metaheuristic search algorithms were included in the analysis. The extent to which the results reported here would hold if different random or metaheuristic search algorithms had been used is unknown. Each of these limitations represents an opportunity to expand the base of knowledge by means of future research, and any conclusions about the comparative performance of random vs. metaheuristic search algorithms under different conditions must be considered within the boundaries of these two limitations.

Finally, it has not been the intention of this paper to disparage metaheuristic search algorithms or those who create or use them. On the contrary, metaheuristic search algorithms have proven themselves over a span of several decades to be valuable tools in the data scientist's toolkit, particularly when used properly in situations involving otherwise intractable optimization problems. It has instead been the intention of this paper to draw both managerial and academic attention to the fact that metaheuristic search algorithms do not always outperform random search algorithms in optimization tasks. To assume otherwise is folly. At a minimum, we believe that every researcher who proposes a new metaheuristic search algorithm should compare and report upon the performance of the proposed algorithm against that of a random baseline. After all, the results reported herein show that under certain conditions, the performance of popular and well-established metaheuristic search algorithms can be surpassed by a simple random search, and that as an inevitable result, there are certainly situations in which randomness can actually be good for organizational decision-making.

6. References

[1] S. Luke, *Essentials of Metaheuristics*, 2nd ed. Raleigh, NC: Lulu Press, 2013.

[2] B. L. Golden, S. Raghavan, and E. A. Wasil, *The Vehicle Routing Problem: Latest Advances and New Challenges*. New York, NY: Springer Science & Business Media, 2008.

[3] G. B. Dantzig and J. H. Ramser, "The Truck Dispatching Problem," *Management Science,* vol. 6, pp. 80-91, 1959.

[4] I. H. Osman and G. Laporte, "Metaheuristics: A Bibliography," *Annals of Operations Research,* vol. 63, pp. 511-623, 1996.

[5] F. Glover, "Future Paths for Integer Programming and Links to Artificial Intelligence," *Computers & Operations Research,* vol. 13, pp. 533-549, 1986.

[6] D. E. Knuth, *The Art of Computer Programming*. Boston, MA: Addison-Wesley, 1968.

[7] A. C. M. D. V. Maniezzo, "Distributed Optimization by Ant Colonies," in *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, 1992, p. 134.

[8] X.-S. Yang and S. Deb, "Cuckoo Search via Lévy Flights," presented at the World Congress on Nature & Biologically Inspired Computing, 2009.

[9] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science,* vol. 220, pp. 671-680, 1983.

[10] J. H. Holland, *Adaptation in Natural and Artificial Systems. An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Ann Arbor, MI: University of Michigan Press, 1975.

[11] R. Storn and K. Price, "Differential Evolution–A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces," *Journal of Global Optimization,* vol. 11, pp. 341-359, 1997.

[12] M. Dorigo and L. M. Gambardella, "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem," *IEEE Transactions on Evolutionary Computation,* vol. 1, pp. 53-66, 1997.

[13] S. Nakrani and C. Tovey, "On Honey Bees and Dynamic Server Allocation in Internet Hosting Centers," *Adaptive Behavior,* vol. 12, pp. 223-240, 2004.
[14] R. Eberhart and J. Kennedy, "A New Optimizer Using Particle Swarm Theory," in *Micro Machine and Human Science*, 1995, pp. 39-43.
[15] Z. W. Geem, J. H. Kim, and G. Loganathan, "A New Heuristic Optimization Algorithm: Harmony Search," *Simulation,* vol. 76, pp. 60-68, 2001.
[16] X.-S. Yang, "Firefly Algorithms for Multimodal Optimization," in *International Symposium on Stochastic Algorithms*, 2009, pp. 169-178.
[17] X.-S. Yang and S. Deb, "Engineering Optimisation by Cuckoo Search," *International Journal of Mathematical Modelling and Numerical Optimisation,* vol. 1, pp. 330-343, 2010.
[18] X.-S. Yang, "Metaheuristic Optimization," *Scholarpedia,* vol. 6, p. 11472, 2011.
[19] T. A. Feo and M. G. Resende, "Greedy randomized adaptive search procedures," *Journal of global optimization,* vol. 6, pp. 109-133, 1995.
[20] P. Trkman, "The Critical Success Factors of Business Process Management," *International Journal of Information Management,* vol. 30, pp. 125-134, 2010.
[21] J. Jeston and J. Nelis, *Business Process Management*. New York, NY: Routledge, 2014.
[22] N-Body Shop, *DIRECT: An O(N^2) Direct Sum Gravity Tool*. Seattle, WA: University of Washington, 2017.
[23] G. Keller, *Statistics for Management and Economics*, 10th ed. Stamford, CT: Cengage Learning, 2014.
[24] B. L. Welch, "The Generalization of Student's Problem when Several Different Population Variances are Involved," *Biometrika,* vol. 34, pp. 28-35, 1947.