Subthreshold-Seeking Behavior and Robust Local Search

Darrell Whitley[1], Keith Bush[1], and Jonathan Rowe[2]

[1] Computer Science, Colorado State University, Fort Collins, CO 80523
[2] Computer Science, University of Birmingham, Birmingham B15 2TT, UK

Abstract. Subthreshold-seeking behavior occurs when the majority of the points that an algorithm samples have an evaluation less than some target threshold. We characterize sets of functions where subthreshold-seeking behavior is possible. Analysis shows that subthreshold-seeking behavior, when possible, can be increased when higher bit precision is used with a bit climber search algorithm and a Gray code representation. However, higher precision also can reduce exploration. A simple modification to a bit-climber can improve its subthreshold-seeking behavior. Experiments show that this modification results in both improved search efficiency and effectiveness on common benchmark problems.

1 Introduction and Background

The goal of an optimization algorithm is to find optimal points of a search space. However, it may sometimes be useful to try to locate points that are sufficiently good (e.g., within some threshold). We might also like to have some assurance that an algorithm is relatively effective on a wide range of problems.

We will say that a search algorithm is *robust* if it is able to beat random search across a wide variety of optimization problems. Christensen and Oppacher [1] have shown that the No Free Lunch theorem does not hold over broad classes of problems that can be described using polynomials of a single variable of bounded complexity. The algorithm that Christensen and Oppacher propose is robust in as much as it is able to out-perform random enumeration on a general class of problems. But the algorithm they propose is not very effective as a search algorithm. Can we do better than this? And what theoretical and practical implications does this question imply?

In this paper we generalize the approach of Christensen and Oppacher. We will say that an algorithm has *subthreshold-seeking behavior* if the algorithm establishes an aspiration threshold, and then spends more than half of its time sampling points that are below threshold. An algorithm with subthreshold-seeking behavior can beat random enumeration and side-step the No Free Lunch result by focusing on a special, but nevertheless general class of functions.

We next ask to what degree does a local search bit climber display robust, subthreshold-seeking behavior. We show that there are a number of conditions where a local search bit climber will display subthreshold-seeking behavior.

We also present several theorems that indicate how and when a local search bit climber can display subthreshold-seeking behavior. These results prove that subthreshold-seeking behavior increases when higher bit precision encodings are used. In addition to the theoretical analysis presented, we empirically show that a local search bit climber with sufficient precision will spend most of its time "subthreshold" on a number of common benchmark problems. We then make a very simple modification to a bit climber with restarts algorithm to allow it to spend more time "subthreshold." The modified algorithm is both more efficient and effective, finding better solutions faster than local search with restarts.

1.1 The SubMedian-Seeker

Suppose we have an objective function $f : [a, b] \to R$, where $[a, b]$ is a closed interval. We discretize this interval by taking N uniformly sampled points, which we label with the set $\mathcal{X} = 0, 1, ..., N - 1$. By abuse of notation, we will consider $f : \mathcal{X} \to R$, such that $f(x)$ takes on the evaluation of the point labeled x. Assume f is bijective as a function of $\mathcal{X}$ and that the median value of f is known and denoted by $med(f)$.

Christensen and Oppacher define a minimization algorithm called *SubMedian-Seeker.* The algorithm presented here is a similar to SubMedian-Seeker but is simpler and easier to understand.[1]

```
EZ-SubMedian-Seeker

1. If less than $\frac{|\mathcal{X}|}{2}$ points have been sampled, then choose a random sample
   point, $x \in \mathcal{X}$. Otherwise terminate.
2. While $f(x) < med(f)$ pick next sample $x = x + 1$. Else goto step 1.
```

Without lose of generality, we assume that x and its successor $x + 1$ are integers. The algorithm exploits the fact that for certain classes of functions, points that are adjacent to submedian points are more often than not also submedian points. The actual performance depends on $M(f)$, which measures the number of submedian values of f that have *successors* with supermedian values. Let M_{crit} be a critical value relative to $M(f)$ such that when $M(f) < M_{crit}$ SubMedian-Seeker (or EZ-SubMedian-Seeker) is better than random search.

Christensen and Oppacher [1] then prove:

> *If f is a uniformly sampled polynomial of degree at most k and if $M_{crit} > k/2$ then SubMedian-Seeker beats random search.*

[1] The original submedian seeker is able to detect and exploit functions where every other point is below submedian and thus a local optimum. EZ-SubMedian-Seeker will not detect this regularity. However, we generally are not concerned with functions that are maximally multimodal. Also, the Christensen and Oppacher proof still holds for EZ-SubMedian-Seeker.

This result holds because there are at most k solutions to $f(x) = y$ where y can be any particular co-domain value. If y is a threshold, then there are at most k crossings of this threshold over the sampled interval. Half, or $k/2$, of these are crossings from subthreshold to superthreshold values. Thus, $M(f) <= k/2$ for polynomials of degree k. In the case where the median is the threshold, step 1 has equal probability of sampling either a submedian or supermedian value. Therefore, as long as step 2 generates a surplus of submedian points before terminating at a supermedian point, the algorithm beats random search. We can think of step 1 as an exploration phase with balanced cost and step 2 as an exploitation phase that accumulates submedian points. If $M(f) \leq k/2 < M_{crit}$, then SubMedian-Seeker (and EZ-SubMedian-Seeker) will perform better than random enumeration because more time is spent below threshold during step 2. Christensen and Oppacher offer extensions of the proof for certain multivariate polynomials as well. In the next section we characterize a more general Subthreshold-Seeker algorithm.

2 Subthreshold-Seeker

We still assume f is 1-dimensional and bijective and $N = |\mathcal{X}|$. Set a threshold of α between 0 and 1/2. We are interested in spending time in the αN best points of the search space (ordered by f). We refer to these as *subthreshold* points. Addition is *modulo* N and the search space is assumed to wrap around so that points 0 and $N - 1$ are neighbors.

Let $\Theta(f)$ denote a threshold co-domain value such that exactly αN points of $\mathcal{X}$ have evaluations, $f(x)$, less than $\Theta(f)$. Subthreshold-Seeker works as follows:

1. Pick a random element $x \in \mathcal{X}$ that has not been seen before.
2. If $f(x) < \Theta(f)$ let $x = x + 1$ and $y = x - 1$; otherwise goto 1.
3. While $f(x) < \Theta(f)$ pick next sample $x = x + 1$.
4. While $f(y) < \Theta(f)$ pick next sample $y = y - 1$.
5. If Stopping-Condition not true, goto 1.

Once a subthreshold region has been found, this algorithm searches left and right for subthreshold neighbors. This minor variation on Submedian-Seeker means that a well defined region has been fully exploited. This is critical to our quantification of this process. We will address the "Stopping-Condition" later.

For theoretical purposes, we assume that the function $\Theta(f)$ is provided. In practice, we can select $\Theta(f)$ based on an empirical sample.

2.1 Functions with Uniform Quasi-basins

We will define a *quasi-basin* as a set of contiguous points that are below a threshold value. Note this is different from the usual definition of a basin: a quasi-basin may contain multiple local optima. Consider a function f where all subthreshold points are contained in B equally sized quasi-basins of size $\alpha N/B$. We then ask how many superthreshold points are visited before all the subthreshold points are found. Suppose k quasi-basins already have been found and explored. Then there remains $B-k$ quasi-basins to find, each containing $\alpha N/B$ points. There are at most $N - k\alpha N/B$ superthreshold points unvisited. So the probability of hitting a new quasi-basin is (slightly better than)

$$\frac{(B-k)(\alpha N/B)}{N - k\alpha N/B} = \frac{(B-k)\alpha}{B-k\alpha}$$

This calculation is approximate because it assumes that superthreshold points are sampled with replacement. As long as the probability of randomly sampling the same superthreshold point twice is extremely small, the approximation will be accurate. For large search spaces this approximation should be good.

If the probability of "hitting" a quasi-basin is p, the expected number of trials until a "hit" occurs is $1/p$. This implies that the expected number of misses before a successful hit occurs is $1/p-1$. So the expected number of superthreshold points that are sampled before finding a new quasi-basin is approximately (slightly less than)

$$\frac{B-k\alpha}{(B-k)\alpha} - 1 = \frac{B(1-\alpha)}{(B-k)\alpha}$$

This means that the expected number of superthreshold points seen before the algorithm has found all quasi-basins is bounded above by

$$\sum_{k=1}^{B-1} \frac{B(1-\alpha)}{(B-k)\alpha} \quad = \quad \frac{B(1-\alpha)}{\alpha} \sum_{k=1}^{B-1} \frac{1}{(B-k)} \quad = \quad \frac{B(1-\alpha)}{\alpha} H(B-1) \qquad (1)$$

where H is the harmonic function. Note $H(B-1)$ is approximated by $(\log(B-1))$. Throughout this paper log denotes $\log_2$.

2.2 Functions with Unevenly Sized Quasi-basins

Now suppose that the quasi-basins are not evenly sized. If f is a uniformly sampled polynomial of degree at most k it can have at most $k/2$ quasi-basins and after fixing $\mathcal{X}$, there must be at least 1 subthreshold quasi-basin of size $\alpha N/(k/2)$ or larger. One way Subthreshold-Seeker can be used is to search until one quasi-basin of size at least $\alpha N/(k/2)$ is found. The waiting time to find such a basin is less than $\frac{N}{\alpha N/(k/2)} = \frac{k/2}{\alpha}$. It can be shown that more subthreshold points will be sampled than superthreshold points as long as

$$\alpha \frac{k/2}{\alpha} + \alpha N/(k/2) > (1-\alpha)\frac{k/2}{\alpha} \quad \text{which reduces to} \quad 2\alpha + \frac{\alpha^2 N}{(k/2)^2} > 1$$

and this does not even count smaller subthreshold quasi-basins that are exploited before finding one of size $\alpha N/(k/2)$.

What if we want to find all quasi-basins that contain at least M points, and we suppose there are B such quasi-basins. (If we happen to find some smaller ones, that is a bonus). Suppose we have found k such quasi-basins. The probability of finding another is

$$> \frac{(B-k)M}{(B-k)M + N - BM} = \frac{(B-k)M}{N-kM}$$

So the expected number of superthreshold points visited before this happens is

$$< \frac{N-kM}{(b-k)M} - 1 = \frac{N-BM}{BM-kM}$$

The total number of superthreshold points visited is thus

$$< \sum_{k=1}^{B-1} \frac{N-BM}{BM-kM} = \left(\frac{N-BM}{M}\right) H(B-1)$$

Theorem 1: *Let α define a threshold presenting some fraction of the search space. Suppose there are B quasi-basins each containing at least M points. If $M > \sqrt{\frac{NH(B-1)}{B}}$ then Subthreshold Seeker can find all B basins and will explore more subthreshold points that superthreshold points. For all $\alpha < 1/2$ Subthreshold Seeker beats random search.*

Proof:

$$\text{If } M^2 > \frac{NH(B-1)}{B} \text{ then } BM > \frac{NH(B-1)}{M} > \left(\frac{N}{M} - B\right) H(B-1).$$

BM is the total number of subthreshold points visited. The expected number of superthreshold points visited is given by $\left(\frac{N}{M} - B\right) H(B-1)$. Therefore, for sufficiently large N, when $M > \sqrt{\frac{NH(B-1)}{B}}$ more subthreshold points are visited than superthreshold points. □

Given information (or strong assumptions) about M and B we can restrict α and spend more time exploring the best regions of the search space compared to SubMedian-Seeker.

Table 1 calculates $M = \sqrt{\frac{NH(B-1)}{B}}$ rounded up to the nearest integer. It also computes α. This value is exact when $\alpha > BM/N$; otherwise it underestimates the percentage of the space that is below threshold. Clearly, as the number of quasi-basins goes up, fewer points occur in each quasi-basin. As the search space size increases, there are more points in each quasi-basin, but we can also lower α so that the number of subthreshold points becomes a smaller percent of the search space. The smaller the subthreshold value, the more Subthreshold-Seeker will beat random-search, and the more effective Subthreshold-Seeker becomes as a general search algorithm.

This perspective also provides another insight. Note that we can interpret the change in the size of the search space as a change in precision: the number of quasi-basins generally does not change (for polynomials the number of

Table 1. This table computes how large quasi-basins must be in order for more subthreshold points to be sampled than superthreshold points when there are different numbers of quasi-basins (B) and for different size search spaces (N). The threshold α is also given, assuming exactly B quasi-basins all of size M.

	B = Number of Quasi-Basins							
Size = N	2^5	2^7	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}
10^6	396	234	99	74	55	40	30	22
10^7	1250	740	313	232	172	126	93	68
10^8	3953	2339	988	733	542	399	293	214
10^9	12500	7395	3123	2317	1711	1260	924	676

Sizes of quasi-basins, M, as a function of B and N.

Size=N	2^5	2^7	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}
10^6	0.013	0.030	0.10	0.150	0.222	0.326	0.479	0.701
10^7	0.004	0.009	0.032	0.047	0.070	0.103	0.151	0.222
10^8	0.001	0.003	0.010	0.015	0.022	0.033	0.048	0.070
10^9	0.0004	0.001	0.003	0.005	0.007	0.010	0.015	0.022

Corresponding values of α

quasi-basins is bounded by $k/2$), but we sample more densely, thus effectively increasing the number of points in the search space. Higher precision allows the search to spend more time subthreshold (or to use a lower threshold). But if the precision is too high, search provides little in the way of exploration when making subthreshold moves.

The subthreshold algorithm, like the original SubMedian-Seeker, isn't really an efficient search algorithm. The goal of search usually is not to examine as many sub-threshold points as possible. However, understanding how such algorithms beat random enumeration can provide practical insights.

Two observations come out of this work. One observation is that precision matters. For any threshold, the relative proportion of points that are subthreshold does not change with a change in precision. However, the number of points that fall within a quasi-basin increases with precision. Assuming a change in precision does not change the boundaries of the quasi-basins, algorithms with subthreshold-seeking behavior will spend more time subthreshold at higher precision. The second observation is that sampling can potentially be used to establish threshold values that can be used to focus the search. In the remainder of the paper, we explore both of these ideas in conjunction with simple, but practical, local search methods.

3 Precision and Subthreshold Local Search

We first look at how precision affects the number of neighbors that exist within a certain distance from some reference point under Gray code.

Lemma 1: *Given a 1-D function of size $N = 2^L$ and a reference point R in the function, under a Gray encoding at most $\lceil \log(Q) \rceil$ bits encode for points that are more than a distance of D points away from R, where $D = \frac{1}{Q}N$.*

Proof: In the 1-dimensional case when the highest order bit is changed under Gray encoding this accesses the only neighbor that is in the opposite half of the search space. (This does not imply that the neighbor is necessarily far away.)

Bits are eliminated to remove the remaining half of the search space which does not contain the reference point. We continue to reduce the search space around the reference point by removing bits until $\lceil \log(Q) \rceil$ bits have been eliminated. The remaining search space is then at most $D = N/Q$ points since

$$\log(N/Q) + \log(Q) = \log(N) \quad \text{and} \quad N(1/2)^{\lceil \log(Q) \rceil} \leq N/Q \qquad \square.$$

As precision increases, the quantity N/Q becomes larger and thus $\log(N/Q)$ increases. However Q and $\log(Q)$ remain constant. Thus, at higher precision, the number of neighbors within a distance of N/Q points increases.

Now assume we are trying to isolate a quasi-basin of size N/Q points.

Theorem 2: *Given a quasi-basin that spans $1/Q$ of a 1-D function of size $N = 2^L$ and and a reference point R inside the quasi-basin, the expected number of neighbors of R that fall inside the quasi-basin under a reflected Gray code is greater than $\lfloor (log(N/Q)) \rfloor - 1$.*

Sketch of Key Ideas:

The full proof for this theorem is long and is available in an expanded version of the current paper (see www.cs.colostate.edu/ genitor/Pubs.html). The proof averages over all possible points in the quasi-basin and all possible placements of reflections points; this accounts for all possible neighborhood configurations that can exist within a quasi-basin.

We define the lower triangle matrix M^x using a recursive definition such that $M^1 = [1]$. Matrix M^x can be decomposed into a 2^{x-1} by 2^{x-1} square matrix whose elements are all the integer x, plus 2 identical lower triangle matrices M^{x-1}. The square matrix occupies the first 2^{x-1} columns of the last 2^{x-1} rows of M^x. The first $2^{x-1} - 1$ rows of M^x correspond to the recursively defined matrix M^{x-1}. Finally, another copy of M^{x-1} is appended to the last $2^{x-1} - 1$ rows of the square matrix.

The following illustration represents a quasi-basin over 7 points (left). Each row represents the number of neighbors for each point, such that those neighbors falls inside the quasi-basin under a Gray code, where the main reflection point of the Gray code is at the location marked by the | symbol. The lower triangle matrix on the right is the matrix M^3 for a search space of size $2^3 - 1 = 7$.

```
2 3 2 3 3 3 2 |
  2 2 3 3 2 3 | 1                  1
    1 3 2 3 3 | 2 2                2 2
      2 3 3 3 | 2 3 2              2 2 1
        2 3 2 | 3 3 3 2            3 3 3 3
```

```
2 2 | 3 3 2 3 1            3 3 3 3 1
  1 | 3 2 3 3 2 2          3 3 3 3 2 2
    | 2 3 3 3 2 3 2        3 3 3 3 2 2 1
```

Let $F(x)$ compute the average value over the elements of matrix M^x. By induction one can show:

$$x - 1 < F(x) = \frac{2^x x}{2^x - 1} - 1 < x$$

The full proof shows via two constructive subproofs that as the size of the quasi-basin grows from $2^x - 1$ points to $2^{x+1} - 2$ points the expected number of neighbors is always greater than $F(x) > x - 1$ where $x = \lfloor (log(N/Q)) \rfloor$.

Corollary: *Given a quasi-basin that spans $1/Q$ of the search space and a reference point R that falls in the quasi-basin, the majority of the neighbors of R under a reflected Gray code representation of a search space of size N will be subthreshold in expectation when $\lfloor (log(N/Q)) \rfloor - 1 > \log(Q) + 1$.*

A local search algorithm currently at a subthreshold point can only move to an equal or better point which must also be subthreshold. And as precision increases, the number of subthreshold neighbors also increases, since $\lfloor (log(N/Q)) \rfloor - 1$ increases while Q remains constant. This assumes the quasibasin is not divided by increasing the precision. The above analysis would need to hold for each dimension of a multidimensional search space, but these results suggest there are very general conditions where a bit-climbing algorithm using a Gray code representation can display subthreshold-seeking behavior. This also assumes the search algorithm can absorb the start-up costs of locating a subthreshold starting point.

4 Algorithms

Under favorable conditions a bit-climbing algorithm using a Gray code representation can display subthreshold seeking behavior, but do they display subthreshold seeking behavior on common benchmarks? In this section, we compared two versions of bit-climbers. Both algorithms use steepest ascent Local Search (LS) which evaluates all neighbors before moving. One algorithm, denoted LS-Rand, uses random restarts. Another algorithm, denoted LS-SubT, uses sampling to start search the bit climbing process at a subthreshold point.

LS-SubT first samples 1000 random points, and then climbs from the 100 best of these points. In this way, LS-SubT estimates a threshold value and attempts to stay in the best 10 percent of the search space.

LS-Rand does 100+y random restarts. LS-Rand was given y additional random starts to compensate for the 1000 sample evaluations used by the LS-SubT algorithm. To calculate y we looked at the size of the bit encoding and the average number of moves needed to reach a local optimum.

4.1 Experiments and Results

Both LS-Rand and LS-SubT were tested on benchmarks taken from Whitley et al. [2] who also provide function definitions. The test function included Rastrigin (F6) and Schwefel (F7) which are both separable. The other functions include Rosenbrock (De Jong's F2), F101 and Rana functions as well as a *spike* function similar to one defined by Ackley [3] where:

$$F(x,y) = -20e^{-0.2\sqrt{(x^2+y^2)/2}} - e^{(cos2\pi x + cos2\pi y)/2} + 22.7, \quad x_i \in [-32.768, 32.768]$$

All problems were posed as 2-dimensional search problems. Experiments were performed at 10 and 20 bits of resolution per parameter. A *descent* corresponds to one iteration of local search, which will locate one local optimum. A *trial* corresponds to one run of the respective algorithm, composed of 100 descents for LS-SubT and $100 + y$ descents for LS-Rand. An *experiment* corresponds to 30 trials. Each experiment is a configuration of search algorithm, test function and parameter resolution. Statistics are computed over each experiment. All chromosomes were encoded using standard Gray code.

Table 2. Results of steepest ascent search at 10 bit resolution per parameter in 2-dimensional space. LS-Rand (here Rand) used 104 restarts. LS-SubT (here SubT) restarted from best 100 of 1000 random points. 0.0* denotes a value less than 1×10^{-13}. Evals were rounded to the nearest integer. The † denotes a statistically significant difference at the 0.05 level using a t-Test.

Function	ALG	Mean	Std	Best	Std	Sub	Evals	Std
ackley	Rand	2.72	0.71	0.18	0.0*	62.4	19371	663
	SubT	0.79†	0.32	0.18	0.0*	79.7	16214†	163
f101	Rand	-29.2	0.0*	-29.2	0.0*	71.7	22917	288
	SubT	-29.2	0.0*	-29.2	0.0*	84.0	18540†	456
f2	Rand	0.10	0.01	0.001	0.002	61.4	23504	3052
	SubT	0.10	0.01	0.0004	0.0*	72.0	666†	1398
griewangk	Rand	0.86	0.16	0.010	0.011	59.5	13412	370
	SubT	0.75†	0.11	0.005	0.009	80.1	9692†	125
rana	Rand	-37.8	0.84	-49.65	0.59	49.5	22575	2296
	SubT	-39.7†	0.68	-49.49	0.52	57.6	19453†	1288
rastrigin	Rand	4.05	0.20	0.100	0.30	63.5	18770	495
	SubT	4.00	0.28	0.0	—	75.4	14442†	343
schwefel	Rand	-615.8	11.8	-837.9	0.0*	53.5	17796	318
	SubT	-648.0†	10.1	-837.9	0.0*	68.0	14580†	414

The results of 10 and 20 bit resolution experiments are given in Tables 2 and 3 respectively. *Mean* denotes mean solution over all descents in all trials. (This is also the mean over all local optima found.) *Best* denotes the *best solution per trial* (i.e., the best optimum found over 100 or $100 + y$ descents). *Sub* denotes the percentage of all evaluations that were subthreshold. *Evals* denotes the

Table 3. Results of steepest ascent search at 20 bit resolution per parameter in 2-dimensional space. LS-Rand (here Rand) used 101 restarts. LS-SubT (here SubT) restarted from best 100 of 1000 random points. 0.0* denotes a value less than 1×10^{-7}. Evals were rounded to the nearest integer. The † denotes a statistically significant difference at the 0.05 level using a t-Test.

Function	ALG	Mean	Std	Best	Std	Sub	Evals	Std
ackley	Rand	2.84	0.66	0.0001	0.0*	75.1	77835	1662
	SubT	0.65†	0.28	0.0001	0.0*	89.9	73212†	1194
f101	Rand	-29.2	0.0*	-29.22	0.0*	84.7	84740	1084
	SubT	-29.2	0.0*	-29.22	0.0*	92.3	77244†	1082
f2	Rand	0.0*	0.0*	0.0*	0.0*	86.0	2×10^7	4×10^5
	SubT	0.0*	0.0*	0.0*	0.0*	85.9	2×10^7	3×10^5
griewangk	Rand	0.75	0.20	0.0045	0.003	80.3	66609	1109
	SubT	0.60†	0.09	0.0049	0.003	90.0	59935†	1103
rana	Rand	-40.63	0.93	-49.76	0.47	74.2	3×10^6	8×10^5
	SubT	-42.54†	0.66	-49.83	0.51	85.0	3×10^6	8×10^5
rastrigin	Rand	4.10	0.22	0.033	0.18	81.5	76335	1734
	SubT	3.94†	0.21	0	—	88.5	68019†	1018
schwefel	Rand	-622.7	13.8	-837.97	0.0*	73.5	75285	969
	SubT	-660.4†	13.4	-837.97	0.0*	84.8	69372†	1340

mean number of test function evaluations per trial averaged over all trials in the experiment. *Stddev* denotes the standard deviation of the value given in the adjacent left-hand column.

In general, the results indicate that LS-SubT sometimes produces statistically significant better solution quality compared to LS-Rand. LS-SubT never produces statistically significant worse performance than LS-Rand.

The data suggests two observations about subthreshold-seeking behavior. First, the sampling used by LS-SubT results in a higher proportion of subthreshold points compared to LS-Rand as shown in Tables 2 and 3. Second, a larger proportion of subthreshold neighbors are sampled for searches using higher precision. At 20 bits of precision per parameter, at least 70 percent of the points sampled by LS-Rand were subthreshold, and at least 80 percent of the points samples by LS-SubT were subthreshold. At 10 bits of precision per parameter, LS-SubT sampled subthreshold points 57 to 84 percent of the time.

At 10 bits of precision, LS-SubT also did fewer evaluations, meaning that it reached local optima faster than LS-Rand. This makes sense in as much as it starts at points with better evaluations. Sometimes the difference was dramatic. Thus, the majority of the time LS-SubT also produced solutions as good or better than LS-Rand, and it did so with less effort.

At 20 bits of precision, there is less difference between LS-Rand and LS-SubT. This follows from our theory, since higher precision implies that both algorithms spend more time subthreshold after a subthreshold point is found, but this does not necessarily result in faster search.

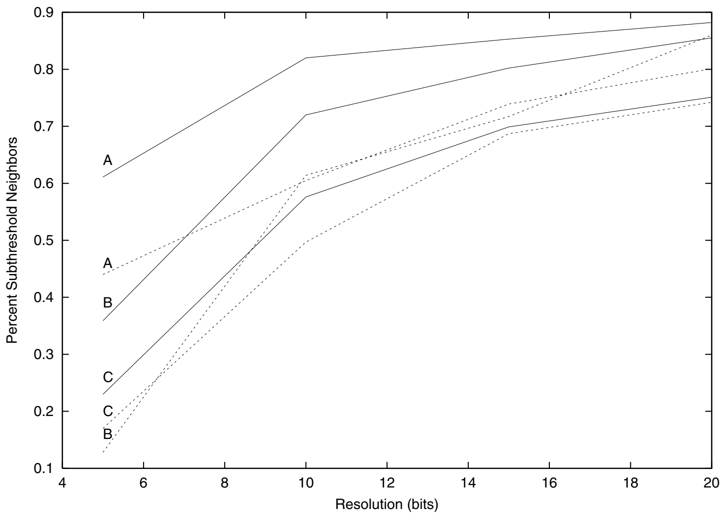

Fig. 1. The graph shows the number of subthreshold points for LS-SubT (the solid line) and LS-Rand (the dashed line) as a function of precision. The functions are A=Griewangk, B=F2, and C=Rana.

To further explore the impact of bit precision, we performed local search using LS-Rand and LS-SubT at 5, 10, 15 and 20 bits of resolution. The test functions used were Griewangk, F2, and Rana functions. Figure 1 shows the percentage of points that were subthreshold at different resolutions. Across all three functions, higher resolution produced a greater percentage of subthreshold sampling. At 5 bit precision less than half of all neighbors are subthreshold for LS-Rand on all functions. With too few bits, both neighborhood sampling and random sampling misses quasi-basins. But with too many bits, search is slowed down because the stepsize can become too small.

There is still much the heuristic search community does not understand about the impact of using different bit precisions. Unexpected results were encountered on two functions. The number of evaluations that were executed on the Rana and F2 functions at 20-bit resolution is huge. Examination of the search space shows that both of these functions contain "ridges" that run at almost 45 degrees relative to the (x, y) coordinates. In this context, the steepest ascent bit-climber is forced to creep along the ridge in very small, incremental steps. Higher precision exaggerates this problem, which is hardly noticeable at 10 bits of precision. This is a serious problem we are continuing to research.

5 Conclusions

The No Free Lunch theorem formalizes the idea that all blackbox search algorithms have identical behavior over the set of all possible discrete functions [4]

[5] [6]. Schumacher et al. [7] review different variants of the No Free Lunch theorem; they show that No Free Lunch holds over a finite set if and only that set is closed under permutation.

The Subthreshold-Seeker algorithm can focus search in the better regions of the search space. Conditions are outlined that allow a subthreshold seeker to beat random enumeration on problems of bounded complexity.

A simple sampling mechanism can be used to initialize local search at subthreshold points, thereby increasing the potential for subthreshold seeking behavior. Of course this strategy also has its own failure modes. Assume that an "important" basin of attraction, or a quasi-basin, is very large above threshold, yet small below threshold; then it is possible that random restarts could have an advantage over subthreshold restarts if success were measured in terms of finding and exploiting this "important" region. Of course the problem with random restarts is that the search can converge to local optima that are superthreshold.

Acknowledgments. This work was supported by National Science Foundation grant IIS-0117209.

References

[1] Christensen, S., (2001), F.O.: What can we learn from No Free Lunch? In: GECCO-01, Morgan Kaufmann (2001) 1219–1226

[2] Whitley, D., Mathias, K., Rana, S., Dzubera, J.: Evaluating Evolutionary Algorithms. Artificial Intelligence Journal **85** (1996) 1–32

[3] Ackley, D.: A Connectionist Machine for Genetic Hillclimbing. Kluwer Academic Publishers (1987)

[4] Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation **4** (1997) 67–82

[5] Radcliffe, N., Surry, P.: Fundamental limitations on search algorithms: Evolutionary computing in perspective. In van Leeuwen, J., ed.: Lecture Notes in Computer Science 1000. Springer-Verlag (1995)

[6] Culberson, J.: On the Futility of Blind Search. Evolutionary Computation **6(2)** (1998) 109–127

[7] Schumacher, C., Vose, M., Whitley, D.: The No Free Lunch and Problem Description Length. In: GECCO-01, Morgan Kaufmann (2001) 565–570