An Empirical Evaluation of O(1) Steepest Descent for NK-Landscapes.

Darrell Whitley, Wenxiang Chen, and Adele Howe

Colorado State University, Fort Collins, CO, USA

Abstract. New methods make it possible to do approximate steepest descent in O(1) time per move for k-bounded pseudo-Boolean functions using stochastic local search. It is also possible to use the average fitness over the Hamming distance 2 neighborhood as a surrogate fitness function and still retain the $O(1)$ time per move. These are average complexity results. In light of these new results, we examine three factors that can influence both the computational cost and the effectiveness of stochastic local search: 1) Fitness function: $f(x)$ or a surrogate; 2) Local optimum escape method: hard random or soft restarts; 3) Descent strategy: next or steepest. We empirically assess these factors in a study of local search for solving NK-landscape problems.

Keywords: Stochastic Local Search, NK-Landscapes, Surrogate Fitness

1 Introduction

The objective function for a number of combinatorial optimization problems, including MAX-kSAT [10] and NK-Landscapes [5], can be expressed as k-bounded pseudo-Boolean functions [7]. New results show that a form of approximate steepest descent can be implemented that requires on average $O(1)$ per move for k-bounded pseudo-Boolean functions [9].

Let X represent the set of candidate solutions, where each solution is a binary string of length N. Let $z \in X$ be the current solution. Let the function $N(z)$ generate the neighbors of solution z under the Hamming distance 1 "bit-flip" neighborhood. Thus, $x \in N(z)$ denotes that $x \in X$ is a neighbor of z. Typically, steepest descent Stochastic Local Search (SLS) requires $O(N)$ time. Using Walsh analysis it is possible to achieve an $O(1)$ average time complexity for each approximate steepest descent move [9].

The Walsh analysis makes it possible to compute the average fitness of the neighborhood reachable via each of the potential next moves. Viewing the search space as a tree rooted at the current solution z where x is a child of z such that $x \in N(z)$, it is also possible to compute neighborhood average of solutions reachable in two moves from vertex z via vertex x (i.e., these are the grandchildren of z that are also children of x).

$$Avg(N(x)) = 1/N \sum_{\mathbf{i}=1}^{N} f(y_i) \quad \text{where } y_i \in N(x) \text{ and } x \in N(z) \tag{1}$$

Whitley and Chen [9] prove that approximate steepest descent using $Avg(N(x))$ as a surrogate function can execute in $O(1)$ time.

In this paper we explore several decisions related to designing an effective local search algorithm based on using $Avg(N(x))$. First, we explore whether there is any advantage to using $Avg(N(x))$ as a surrogate fitness function over using $f(x)$. We hypothesize that more plateaus are found in NKq-landscapes, which should favor the use of the $Avg(N(x))$ fitness function [1,8].

Second, a fundamental decision is what to do when a local optimum is encountered. One simple answer is to use a random restart. However, the proof of $O(1)$ complexity suggests that there is a significant advantage to using soft-restarts: executing a small number of random moves to escape a local optimum, where each move can be executed in $O(1)$ time. The use of soft restarts transforms the search into an Iterated Local Search algorithm [4]. Since a hard random restart requires a complete $O(N)$ reinitialization of the search, there is an advantage to using random walks to escape local optima.

Finally, our new theoretical results now make it possible to perform both *steepest* descent or *next* descent move selection in $O(1)$ time. Given that there is now no difference in cost, is there now an advantage to using *steepest* descent in place of *next* descent?

Given the constant time $O(1)$ move selection, we empirically evaluate the impact of critical design decisions on performance for k-bounded pseudo-Boolean NK-landscape and NKq-landscape problems. We control for runtime and assess solution quality.

2 The Theoretical $O(1)$ Result

This section briefly summarizes the theoretical results of Whitley and Chen [9]. Any discrete function $f : \{0,1\}^N \implies \mathbb{R}$ can be represented in the Walsh basis:

$$f(x) = \sum_{i=0}^{2^n-1} w_i \psi_i(x)$$

where w_i is a real-valued constant called a Walsh coefficient and $\psi_i(x) = -1^{i^T x}$ is a Walsh function that generates a sign. Alternatively, $\psi_i(x) = -1^{bitcount(i \wedge x)}$ where $bitcount(i \wedge x)$ counts how many 1-bits are in the string produced by the logical operation $i \wedge x$. The *order* of the i^{th} Walsh function is $bitcount(i)$.

Normally generating the Walsh coefficients requires that the entire search space be enumerated. However, Rana et al. [6] show for the MAX-kSAT problem that if a function is composed of subfunctions, each of which is a function over k Boolean variables, then the order of nonzero Walsh coefficients of the corresponding subfunction f_j is also bounded by 2^k. This result holds for all k-bounded pseudo-Boolean functions, including NK-Landscapes [2,3,7].

We will use a vector denoted by w' to store the Walsh coefficients, which will include the sign relative to solution x such that : $w'_i(x) = w_i \psi_i(x)$.

We assume the Walsh coefficients and their signs have been computed for some initial solution x. We will use b to index all of the Walsh coefficients in vector $w'(x)$. Let p be a string with exactly one bit set to 1. Let $p \subset b$ denote that bit p has value 1 in string b (i.e., $p \wedge b = p$). We can then compute the sum of all of the components in w' that are affected when local search flips bit p.

$$S_p(x) = \sum_{\forall b,\ p \subset b} w'_b(x)$$

Let $y_p \in N(x)$ be the neighbor of string x generated by flipping bit p. Then $f(y_p) = f(x) - 2(S_p(x))$ for all $y_p \in N(x)$. Assuming $y_i \in N(x)$ and $y_j \in N(x)$, then $f(x)$ can be viewed as a constant in the local neighborhood and

$$S_i(x) > S_j(x) \iff f(y_i) < f(y_j)$$

and thus selecting the maximal $S_i(x)$ yields the minimal neighbor of $f(x)$ [9].

If bit p is flipped, we must update element S_q by flipping the sign of the Walsh coefficients that are jointly indexed by p.

$$S_q(y_p) = S_q(x) - 2 \sum_{\forall b,(p \wedge q) \subset b} w'_b(x)$$

$$\forall b, \quad \text{if } (p \subset b) \quad \text{then } w'_b(y_p) = -w'_b(x) \quad \text{else } w'_b(y_p) = w'_b(x)$$

The vector S needs to be initialized after the first local search move. After that, only select elements of the vector S must be updated after a bit flip.

Whitley and Chen show that on average the expected number of elements in the vector S that must be updated is $O(1)$. The proof assumes those variables that appear more than $\mathbf{T}$ times across all subfunctions become Tabu after 1 flip, where $\mathbf{T}$ is a constant. All other variables can be flipped. A variable that is Tabu remains Tabu for N bit flips. The exact number of the updates is $k(k-1)+1 = O(1)$ when $\mathbf{T}$ is equal to the expected number of times a variable would appear in a randomly generated subfunction of an NK-landscape. However, empirical data suggest that no Tabu mechanism is necessary: during local search the waiting time between bit flips for variables that appear more than $\mathbf{T}$ times across all subfunctions is $\geq N$.

For constant time steepest descent, it also must be true that there cannot be too many unexploited improving moves. For example, assume that search starts from an extremum, and *every move is an improving move.* To do true steepest descent, we must use a priority queue in the form of a heap to select the best improving move, which results in $O(lg\ N)$ time to select the best improving move. However, in practice, there are typically few improving moves. We can implement approximate steepest descent as follows: assume that a threshold M is selected. We will store the location of improving moves in a buffer B. Let $|B|$ denote the number of elements stored in B. If $|B| \leq M$, then we will scan B and select the steepest descent improving move. If $|B| > M$, then we sample M moves from B (a form of "tournament selection") and select the best improving move from sample M. This yields an approximation to the steepest descent

improving move. In practice, we implemented true steepest descent because the number of improving moves is typically small [9].

$Avg(N(x))$ (see equation 1) can also be computed in $O(1)$ time on average. Let $\varphi'_{p,j}$ sum all Walsh coefficients of order j that reference bit p.

$$\varphi'_{p,j}(x) = \sum_{\forall b, bitcount(b)=j, p \subset b} w'_b(x)$$

We can now define the update for the vector S as follows, for $y_p \in N(x)$:

$$S_i(x) = \sum_{j=1}^{k} \varphi'_{i,j}(x) \quad \text{and} \quad S_i(y_p) = S_i(x) - 2 \sum_{\forall b, (p \wedge i) \subset b} w'_b(x) \tag{2}$$

We next define a new vector Z that computes a parallel result

$$Z_i(x) = \sum_{j=1}^{k} j\varphi'_{i,j}(x) \quad \text{and} \quad Z_i(y_p) = Z_i(x) - 2 \sum_{\forall b, (p \wedge i) \subset b} bitcount(b)(w'_b(x)) \tag{3}$$

where $bitcount(b)$ is a lookup table that stores the order of Walsh coefficient w'_b.

Whitley and Chen [9] prove that for any k-bounded pseudo-Boolean function, when flipping bit p and moving from solution z to solutions $x \in N(z)$:

$$Avg(N(x)) = Avg(N(z)) - 2S_p(z) + \frac{4}{N} Z_p(z) \tag{4}$$

Note that the vector S and Z are updated using exactly the same Walsh coefficients. Thus the cost of updating vector Z is the same as the cost of updating the vector S and the same $O(1)$ average time complexity holds for computing the approximate steepest descent move for $Avg(N(x))$.

3 Implementation Details

Algorithm 1, which we refer to as Walsh-LS, outlines the inputs which define 1) the fitness function to use (eval), 2) the descent method to use (descMeth), and 3) the escape scheme to use when a local optimum is reached (escape). Algorithm 2 implements the Update of the S, Z and w vectors.

Descent decides on the bit to be flipped. Improving moves are stored in the buffer. For the current experiments we implemented true steepest descent in which the index of the best improving bit is returned as bestI, with ties being broken randomly. For *next descent*, the first bit in buffer is returned.

Escape Method is triggered when the algorithm reaches a local optimum. If escape is *random walk* then 10 bits are randomly flipped. Each bit flip requires an update to the S vector; the cost on average is still $O(1)$. If escape is *random restart* then the S vector must be reinitialized at $O(N)$ cost.

Algorithm 1: Sol ← WALSH-LS(eval, descMeth, escape)

```
1  bsfSol ← curSol ← INIT() ;              // current and best-so-far solutions
2  s, z, buffer ← WALSHANALYSIS(f, curSol) ;
3  while Termination criterion not met do
4      improve, bestI ← DESCENT(buffer, descMeth) ;
5      if improve == True then
6          w, s, z, buffer ← UPDATE(w, s, z, buffer, bestI, eval);
7          curSol ← FLIP(curSol, bestI) ;      // flips the bestI^th bit of curSol
8      else                                    // local optimum: perturbs current solution
9          bsfSol ← SELECT(curSol, bsfSol) ; // select sol with better fitness
10         curSol ← ESCAPE METHOD(curSol, escape) ;
11         for i in DifferentBit(bsfSol, curSol) do // for each different bit
12             w, s, z, buffer ← UPDATE(w, s, z, buffer, i, eval);
13 bsfSol ← SELECT(curSol, bsfSol) ;
14 return bsfSol
```

3.1 Experiment Design

Our experiments explore how the use of the surrogate fitness function interacts with the Descent and Escape methods. Based on our understanding of the surrogate fitness function, we posit two hypotheses.

1. The low complexity and the lookahead nature of the surrogate will support superior performance on problems where plateaus frequently appear in the

Algorithm 2: w, s, z, buffer ← UPDATE(w, s, z, buffer, p, eval)

```
1  s[p] ← -s[p];
2  for each q interacts with p do // update s vector
3      for each w[i] touching both p and q do
4          s[q] ← s[q] - 2 * w[i];
5  if eval is f(x) then
6      for each s[i] touching p do // update buffer
7          if s[i] is an improving move then BUFFER ← APPEND(buffer, i);
8  else                                        // eval is Avg(N(f))
9      z[p] ← -z[p];
10     for each q interacts with p do // update z vector
11         for each w[i] touching both p and q do
12             z[q] ← z[q] - 2 * ORDER(w[i]) * w[i];
13     for each z[i] touching p do // update buffer
14         if z[i] is an improving move then BUFFER ← APPEND(buffer, i);
15 For each w[i] touching p do w[i] ← -w[i]     \\ update Walsh coefficients
```

fitness landscape, as is the case for NKq-landscapes. Because of the low complexity of evaluating the possible moves, the effect may be independent of descent method, but because different descents may lead to different portions of the search space, it would not be surprising to find an interaction effect.
2. Performing an $O(1)$ cost random walk upon reaching a local optimum should provide an advantage over an $O(N)$ cost hard random restart. This effect should become more pronounced as the problem size increases.

We chose NKq-landscapes with q=2 to increase the occurrence of plateaus. We limited the values of K to 2, 4 and 8 and N to 50, 100, 200 and 500 for two reasons. First, using the fast Walsh update is only advantageous when $N \gg 2^k$. Second, for $K > 8$ the random nature of the subfunctions used in NK-landscapes makes such problems become increasingly random and disordered as K increases. We randomly generated 24 problems, one for each combination of problem type, K and N value. Each combination of factors was run for 30 trials on each problem. The computational platform was Fedora 16 using Xeon Processor E5450 at 3.00GHz. All algorithms are implemented in Python 2.7.2.

To control for the amount of computation done in each configuration, we counted the number of times that UPDATE is executed and terminated a trial when the number is $100 \times N$. Normally the number of fitness evaluations would be counted. However, Walsh-LS requires only partial fitness evaluation (line 6 and line 12 in Algorithm 1) and we count only the updates that need to be recomputed. This gives a clear advantage to random walk over random restart as an escape mechanism. A single random restart uses N updates because $O(N)$ of the elements stored in vector S must be updated. A random walk of length 10 does 10 updates to the S vector.

3.2 Solution Quality

Without a baseline, it can be difficult to know whether differences in performance are meaningful. Observed quality may be near optimal (producing a floor effect) or far away, suggesting considerable room for improvement. In addition, observed differences may be small or mitigated by variance in results across trials. To establish a "best" solution, we run SLS with steepest descent and random walk for both fitness functions 50 times longer than in the normal experiments and harvest the overall $Best$ solution. Then we normalize solution quality to the value $(\frac{f(x)-Best}{Best})$ for each problem instance.

Figure 1 shows the normalized quality of solutions. Some results are within a fraction of the baseline solution while other are 6 times worse. The variance in solution quality was approximately 10 times greater on NKq-landscapes compared to NK-landscapes. For each combination of K and NK/NKq instances, we ran two-way ANOVAs with problem size N and algorithm as the independent variables and solution quality as the dependent. All main effects and interaction effects were significant at $p < .00001$ level, which indicates distinctions between algorithms' performance that varies with problem size.

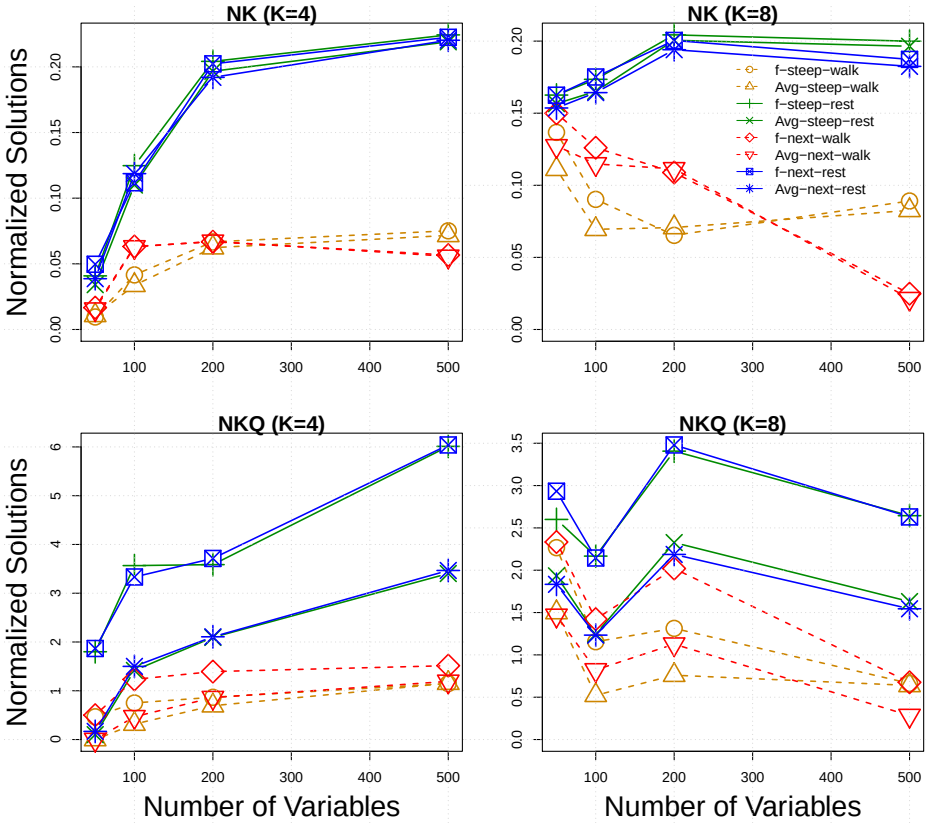

Fig. 1. Normalized Solutions, averaged across trials, found by Walsh-LS across factors. Upper graphs show NK-landscape problems; lower show NKq-landscape problems.

In Figure 1, dashed lines are for configurations that employed a random walk; solid lines employed hard random restarts. As expected, hard random restarts produced poorer results compared to the soft-restarts using a random walk to escape local minima, and the effect appears independent of other design decisions. Table 1 shows that Walsh-LS with random walk visits more local optima and usually visits more *distinct* local optima than Local Search with random restart. Even short random walks are not simply returning back where they started. The ratio of distinct local optima to total local optima suggests that a walk length of 20 might be better than a walk length of 10.

Tables 2 and 3 show means and standard deviations of fitness evaluations. We ran Wilcoxon rank-sum tests for two values of N and two values of K using only random walk for escape and comparing f to Avg in each case. For NK landscapes, statistical tests indicate that in all cases but one the differences are not significant (at the p=0.05 level) when using $Avg(N(x))$ vs. $f(x)$. But for $N = 100$ and K=8 we find p=0.007 for the steepest descent case.

Problem	Eval	# of Locals	Walsh-LS random restarts	Walk Length 10	20	30	40	50
NK-Landscape	$f(x)$	Total	326	619	460	387	344	322
		Distinct	326	219	422	385	343	322
	$Avg(N(x))$	Total	324	622	465	383	345	322
		Distinct	324	234	431	382	345	322
NKq-Landscape	$f(x)$	Total	431	697	553	484	431	427
		Distinct	431	659	552	484	431	427
	$Avg(N(x))$	Total	325	671	516	420	363	324
		Distinct	324	347	481	414	356	323

Table 1. The number of local optima visited by Walsh-LS with steepest descent for problems with $N = 100$, $K = 2$ and $q = 2$. The "Total" rows indicate the number of local optima encountered; "Distinct" rows show the number of those that are unique. We used a Walk Length of 10 in all experiments, but present various Walk Lengths here to show the impact on the number of optima visited.

For NKq problems, the advantage of utilizing $Avg(N(X))$ is clearer. The results using $Avg(N(x))$ are better than $f(x)$ in most cases. In all but two cases, $p < 0.0001$; for $N = 500$ when steepest descent is used the p values are $p = 0.9$ for $K = 4$ and $p = 0.5$ for $K = 8$.

3.3 Runtime Results

The number of "updates" (and thus the number of "moves") was used to control termination. A random restart is implemented as a series of "moves" from the current solution to the new randomly generated solution, which has $O(N)$ complexity. Thus the total execution time used by hard random restarts and the random walk soft restarts is basically the same. There was also virtually no difference in the steepest descent and the next descent runtimes; next descent was faster, but only by a small insignificant amount.

In our current implementation, computing $Avg(N(x))$ requires the use of both the S and Z vectors; Thus the cost of computing $Avg(N(x))$ was approximately twice the cost of computing $f(x)$ for each move.

Descent	Eval	K=4 N=100	K=4 N=500	K=8 N=100	K=8 N=500
steepest	f	.231 ± .003	.223 ± .004	.231 ± .006	.238 ± .005
	Avg	**.229** ± .004	**.222** ± .004	**.226** ± .006	**.236** ± .005
next	f	.236 ± .004	.219 ± .003	.238 ± .006	.224 ± .005
	Avg	.236 ± .003	.219 ± .003	**.236** ± .006	**.223** ± .003

Table 2. Means and standard deviations of fitness evaluations for *NK* problems, organized by configurations of fitness function and descent method (using only random walk for escape), best values in bold.

Descent	Eval	K=4		K=8	
		N=100	N=500	N=100	N=500
steepest	f	.035 ± .007	.039 ± .006	.065 ± .010	.060 ± .005
	Avg	**.026** ± .005	.039 ± .005	**.046** ± .006	**.059** ± .006
next	f	.045 ± .008	.045 ± .004	.073 ± .010	.060 ± .005
	Avg	**.029** ± .004	**.039** ± .003	**.055** ± .009	**.046** ± .004

Table 3. Means and standard deviations of fitness evaluations for NKq problems, organized by configurations of fitness function and descent method (using only random walk for escape), best values in bold.

However, to be more efficient note that

$$\begin{aligned} Avg(N(x_p)) &= Avg(N(z)) - 2S_p(z) + \frac{4}{N}Z_p(z) && \text{by Eqn 4} \\ &= Avg(N(z)) - 2(\sum_{j=1}^{k}\varphi'_{z,j}(x)) + \frac{4}{N}\sum_{j=1}^{k} j\varphi'_{z,j}(x) && \text{by Eqn 2,3} \\ &= Avg(N(z)) + \sum_{j=1}^{k}\frac{(4j-2N)}{N}\varphi'_{z,j}(x) \end{aligned}$$

Construct a new vector $w_i^*(x) = \frac{(4j-2N)}{N}w'(x) = \frac{(4j-2N)}{N}w(x)\psi_i(x)$.
Let $Z_p^*(z) = \frac{4}{N}Z_p(z) - 2S_p(z)$ which yields: $Avg(N(x)) = Avg(N(z)) + Z_p^*(z)$

Using the update rules for vectors S and Z when bit p is flipped:

$$\begin{aligned} Z_i^*(y_p) &= \frac{4}{N}Z_i(y_p) - 2S_i(y_p) \\ &= \frac{4}{N}[Z_i(x) - 2\sum_{\forall b,(p\wedge i)\subset b} j * w'_b(x)] - 2[S_i(x) - 2\sum_{\forall b,(p\wedge i)\subset b} w'_b(x)] \\ &= Z_i^*(x) - 2\sum_{\forall b,(p\wedge i)\subset b}\frac{4j-2N}{N}w'_b(x) \\ &= Z_i^*(x) - 2\sum_{\forall b,(p\wedge i)\subset b} w_b^*(x) \end{aligned}$$

Thus, $Avg(N(x))$ can be computed in precisely the same number of updates needed to compute $f(x)$ and $Z^*(x)$ can directly be used as a proxy for $Avg(N(x))$. Furthermore $f(x)$ can be efficiently computed on demand given $Avg(N(x))$ [7].

4 Conclusions

In light of new methods that allow steepest and next descent to be implemented in $O(1)$ time, we evaluated the impact of the fitness function, the descent method

and the method used to escape local optima on NK and NKq landscapes. Not surprisingly, random walks perform much better than random restarts; using random walks in place of random restarts transforms the algorithm into an Iterated Local Search algorithm [4].

Somewhat more surprising, we find little or no difference between steepest and next descent methods. This could be due to the fact that NK-landscapes have little inherent structure; the results might differ in other domains.

Finally, for NKq landscapes, using $Avg(N(x))$ as the evaluation function instead of $f(x)$ generally improved performance. This appears to be because $Avg(N(x))$ results in fewer plateaus and fewer local optima compared to $f(x)$.

5 Acknowledgments

This research was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number FA9550-11-1-0088. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

References

1. Geard, N.: A comparison of neutral landscapes: NK, NKp and NKq. In: IEEE Congress on Evolutionary Computation (CEC-2002). pp. 205–210 (2002)
2. Heckendorn, R., Whitley, D.: A Walsh analysis of NK-landscapes. In: Proceedings of the Seventh International Conference on Genetic Algorithms (1997)
3. Heckendorn, R.B.: Embedded landscapes. Evolutionary Computation 10(4), 345–369 (2002)
4. Hoos, H.H., Stützle, T.: Stochastic Local Search: Foundations and Applications. Morgan Kaufman (2004)
5. Kauffman, S., Weinberger, E.: The NK Model of Rugged Fitness Landscapes and its Application to Maturation of the Immune Response. Journal of Theoretical Biology 141(2), 211–245 (1989)
6. Rana, S.B., Heckendorn, R.B., Whitley, L.D.: A tractable Walsh analysis of SAT and its implications for genetic algorithms. In: Mostow, J., Rich, C. (eds.) AAAI/IAAI. pp. 392–397. AAAI Press / The MIT Press (1998)
7. Sutton, A.M., Whitley, L.D., Howe, A.E.: Computing the moments of k-bounded pseudo-Boolean functions over Hamming spheres of arbitrary radius in polynomial time. Theoretical Computer Science 425, 58 – 74 (2012)
8. Verel, S., Ochoa, G., Tomassini, M.: Local optima networks of NK landscapes with neutrality. IEEE Transactions on Evolutionary Computation 14(6), 783–797 (2010)
9. Whitley, D., Chen, W.: Constant Time Steepest Ascent Local Search with Statistical Lookahead for NK-Landscapes. In: GECCO '12: Proceedings of the annual conference on Genetic and Evolutionary Computation Conference (2012)
10. Zhang, W.: Configuration landscape analysis and backbone guided local search: part i: Satisfiability and maximum satisfiability. Artificial Intelligence 158, 1–26 (September 2004)