ABSTRACT
Local search methods based on explicit neighborhood enumeration require at least $O(n)$ time to identify all possible improving moves. For $k$-bounded pseudo-Boolean optimization problems, recent approaches have achieved $O(k^2 \cdot 2^k)$ runtime cost per move, where $n$ is the number of variables and $k$ is the number of variables per sub-function. Even though the bound is independent of $n$, the complexity per move is still exponential in $k$. In this paper, we propose a second order partial derivatives-based approach that executes first-improvement local search where the runtime cost per move is time polynomial in $k$ and independent of $n$. This method is applied to NK-landscapes, where larger values of $k$ may be of particular interest.

Categories and Subject Descriptors
I.2.8 [Artificial Intelligence]: Problem Solving, Control Methods, and Search

General Terms
Theory, Algorithms

Keywords
Local Search, Pseudo-Boolean Optimization, Complexity per Move

1. INTRODUCTION
NK-landscapes are as a special class of pseudo-Boolean functions \[2\]. NK-landscapes yield a tunable landscape that becomes more multimodal and nonlinear as $K$ is increased. Let $k$ denote the number of variables in each subfunction of the NK-landscape; since each bit interacts with $K$ other bits, $k = K + 1$. Let $n$ denote the number of variables over all subfunctions.

NK-landscapes were initially created to study biological systems \[8\], but have become a popular benchmark for testing the performance of search algorithms \[16\]. Since NK-landscapes are NP-Complete \[18\], for larger problems researchers and practitioners rely on incomplete solvers such as Stochastic Local Search (SLS) \[7\] and Evolutionary Algorithms \[3,4,16,14\]. First-Improvement Local Search (FILS) is a common form of SLS, which always takes a randomly selected move yielding an improvement in the objective function if one is available. (First-improvement is also often referred to as “next” improvement local search.) Generally, given more time to search, FILS will yield a better solution. It is therefore vital to execute each move efficiently.

Local search methods for NK-Landscapes have been developed that have complexity independent of $n$, but $O(k^2 \cdot 2^k)$ runtime cost per move \[19\]. This paper accelerates FILS for NK-landscapes by reducing the complexity per iteration from $O(k^2 \cdot 2^k)$ to $O(k^2)$. The new method maintains first order partial derivatives for each Boolean variable for determining which bit to flip and applies second order partial derivatives to update the first order partial derivatives after flipping a bit. We show that the first order partial derivatives can be acquired by computing the difference in objective function values between the current candidate and its neighbors under the 1-bit-flip neighborhood operator. Since second order partial derivatives are essentially the difference of first order partial derivatives after flipping another bit, they can also be computed from the objective function on the fly. Situating the computation at the sub-function level allows us to initialize the first derivatives for a given initial candidate solution in $O(k^2 \cdot n)$ time and to update the first order partial derivatives using the second order partial derivatives in $O(k^2)$ every time a bit is flipped. Detailed proofs are provided in section \[3\].

Our approach can dramatically speed up search on NK-landscapes for larger values of $K$. Neighborhood enumeration based implementations of SLS take at least $O(n)$ time to enumerate all $n$ search neighbors under the 1-bit-flip neighborhood operator; inefficient implementations can take $O(n^2)$ time if every subfunction is evaluated for every neighbor. In most real-world applications of Pseudo-Boolean Optimization (PBO), for example MaxSAT, $k$ is typically much smaller than $n$, the number of variables \[12,13\]. The most common benchmark problems for MaxSAT have $k = 3$. Recent studies \[19,20\] achieve $n$-independent complexity per move, i.e., $O(k^2 \cdot 2^k)$. These methods work well on instances as long as $k$ is small. Yet they are intractable on instances with moderately large $k$, and can be even slower than $O(n)$ neighborhood enumeration methods when $2^k > n$. Our derivative-based approach instead has a complexity per move of $O(k^3)$: the runtime cost is independent of $n$ so that the resulting method scales well for both $n$ and $k$.

2. RELATED WORK
As shown in equation \[1\], NK-landscapes \[8\] are constructed from $n$ subfunctions, where each subfunction $f_j(x_{I_j})$ maps a substring $x_{I_j}$ of binary string $x$ $(x \in \{0,1\}^n)$ to a real value drawn uni-
formally at random from \([0,1]\). \(I_j\) is a set of \(k = K + 1\) indices that extract a substring from the full binary string \(x\). The substring \(x_{I_j}\) includes \(x_j\) and \(K\) other randomly chosen bits.

\[
f(x) = \sum_{j=1}^{n} f_j(x_{I_j}). \tag{1}
\]

Evaluating a candidate solution \(x\) to obtain \(f(x)\) clearly takes \(O(n)\) time, since there are \(n\) subfunctions in the summation in equation (1). If all \(n\) neighbors are enumerated in each step of FILS (worst case in which no move is improving) and each full evaluation takes \(O(n)\) time, the complexity per move is \(O(n^2)\).

Derivative based methods have previously been developed for SAT, which is probably the most extensively studied class of pseudo-Boolean functions \([17]\). Hoos and Stützle discuss the efficient implementation of Fils for SAT in chapter 6 of their book \([7]\). They suggest that MAXSAT local search algorithms can achieve \(O(k)\) complexity per move \([6]\) using derivative information for uniform random MAXSAT problems. Their analysis, however, does not consider the cost of tracking improving moves, which can have \(O(n)\) complexity \([19]\). AdaptG\(^2\)W\(\)SAT, one of the current best-performing incomplete SAT solvers, adopts such a technique in its local search \([10,11]\). The analytic form of the objective function for SAT allows the derivatives of any order to be pre-computed beforehand. During search, the gradient at any given candidate solution can be calculated by substituting the candidate solution into the analytic form of the derivatives. In this manner, the derivative based method can be directly applied.

However, these techniques exploit specific characteristics of MAXSAT, such as the discreteness of the co-domain and analytic form of the objective function. As a consequence, they cannot be directly applied to NK-landscapes.

A recent Walsh analysis based method, Walsh-LS, addresses the complexity per move issue of local search for SAT and NK-landscapes \([19,20]\). Walsh-LS consists of three phases: 1) pre-processing phase, 2) initialization phase and 3) update phase. Walsh-LS uses the Walsh transform to decompose the objective function into a linear combination of Walsh bases in the pre-processing phase. This phase takes \(O(2^k \log(2^k) + n) = O(k2^k + n)\) using the fast Walsh transform \([5]\). In the initialization phase, Walsh-LS employs Walsh coefficients to construct the first order partial derivatives for each of the \(n\) variables, which forms a vector of length \(n\). The vector yields the change in objective function value if a bit is flipped. One can then decide which move to take solely from the vector. The initialization phase takes \(O(n)\) time \([19]\). After every move, the vector has to be updated to reflect the first order partial derivatives for the new candidate solution (called the “update phase”). This phase can be achieved by using the second order partial derivatives from Walsh coefficients. Walsh-LS computes derivative information on the fly from Walsh coefficients, which requires \(O(k^2 + 2^k)\) time \([19]\).

Even though the complexity for update operations is independent of \(n\), it grows exponentially in \(k\). According to the empirical studies in \([19]\), Walsh-LS can be even slower than the neighborhood enumeration based method when \(2^k > n\). The complexity of the update runtime cost is due to the \(2^k\) Walsh coefficients associated with each subfunction. We suggest that one can avoid this cost by not using the Walsh polynomial. Instead, we can compute the derivatives on the fly at the level of subfunctions.

### 3. DERIVATIVE-BASED LOCAL SEARCH

In this section, we present Derivative-based Local Search for NK-landscapes, DLS. There are two variants of DLS: 1) Local Search based on First Order Partial Derivatives, \(D1LS\) and 2) Local Search based on Second Order Partial Derivatives, \(D2LS\). Each algorithm description includes a complexity analysis.

#### 3.1 First Order Partial Derivatives

\(D1LS\) applies updates directly to the objective function \(f\). Suppose \(x = [x_1, x_2, \ldots, x_n]\) is the current candidate solution. We denote the candidate solution as \(x^{(q)}\) after flipping a bit \(x_q\), where \(q \in [1,n]\), i.e., \(x^{(q)} = x_q \leftarrow x\). We define first order partial derivatives for \(f\) as below.

\[
\forall q \in [1,n], \quad \frac{\partial f(x)}{\partial x_q} = \frac{f(x^{(q)}) - f(x)}{x^{(q)}_q - x_q} \tag{2}
\]

After flipping a bit \(x_q\), the new objective function value \(f(x^{(q)})\) can be updated using equation (3).

\[
\forall q \in [1,n], \quad f(x^{(q)}) = f(x) + (x^{(q)}_q - x_q) \frac{\partial f(x)}{\partial x_q}
\]

\[
= f(x) + \sum_{q \in I_j} [-f_j(x) + f_j(x^{(q)})] \tag{3}
\]

The idea behind equation (3) is rather intuitive. When a bit \(x_q\) is flipped, only those subfunctions that contain \(x_q\) can possibly change their values (“\(q \in I_j\) gives the indices of such subfunctions). To reflect these changes, we simply need to subtract the old subfunction values \(f_j(x)\), and substitute those with the new ones \(f_j(x^{(q)})\).

The next issue is how to retrieve “\(q \in I_j\)” efficiently. A vector of lists \(\text{SubFuncListOne}\) is maintained for this purpose. The \(i^{th}\) list in \(\text{SubFuncListOne}\) is exactly the indices of subfunctions that contain \(x_i\).

We now develop several lemmas addressing the complexity for \(D1LS\). We first establish a lemma regarding a useful property of NK-landscapes.

**Lemma 1.** On a uniform random instance of NK-landscape, the expected number of subfunctions in which any given variable appears is \(k\).

**Proof.** We shall view the number of subfunctions that any given variable appears in as a random variable \(v\). \(v\) follows the binomial distribution \(B(n, \frac{k}{n})\). The expected value of \(v\) is \(n \cdot \frac{k}{n} = k\). □

**Lemma 2.** \(\text{SubFuncListOne}\) can be generated in \(O(nk)\) time.

**Proof.** Initializing \(\text{SubFuncListOne}\) to a vector of \(n\) empty lists takes \(O(n)\) time. For the \(i^{th}\) subfunction \(f_i\), it contains \(k = K + 1\) variables. \(i\) is appended to exactly \(k\) lists. Since there are \(n\) such subfunctions, \(\text{SubFuncListOne}\) can be generated in \(O(nk)\) time. □

In the worst case, one needs to compute \(f(x^{(q)})\) for all \(q \in [1,n]\) in order to discover an improving move. We provide the complexity per move for \(D1LS\) in lemma 3.

**Lemma 3.** The complexity per move for \(D1LS\) is \(O(nk^2)\).

**Proof.** We analyze the complexity from outside in. At the outermost level, there are \(O(n)\) speculative bits to flip in order to discover an improving move. For a particular speculative bit \(x_q\), there are \(k\) subfunctions in which \(x_q\) appears. By lemma 1! For each term inside the summation, \(O(k)\) operations are required to extract \(k\) bits from the length-\(n\) bit string, so that the extracted \(k\) bits can be used as input to subfunctions. As a whole, computing all \(n\) neighbors \((f(x^{(q)}))\) according to equation (4) takes \(O(nk^2)\) time. □
By equation (5) and equation (7), initializing \(D\) generates data structures required for executing FILS efficiently, and 2) a recurring update phase that performs the actual search.

3.2 Second Order Partial Derivatives

The \(O(n)\) complexity per move in D1LS can be overcome by maintaining a vector of \(n\) delta functions \(\delta v\) initialized by equation (3) and updated using second order partial derivatives. Moreover, updating \(\delta v\) on the level of subfunctions rather than Walsh coefficients makes the complexity both independent of \(n\) and polynomial in \(k\).

D2LS has two phases: 1) a one time initialization phase that generates data structures required for executing FILS efficiently, and 2) a recurring update phase that performs the actual search. 
\[
\delta v = [\delta v_1, \delta v_2, \ldots, \delta v_n]\text{ is defined in equation (4).}
\]

\[
\forall p \in [1, n], \ \delta v_p = f(x^{(p)}) - f(x) \tag{4}
\]

Intuitively, \(\delta v_p\) gives the change in objective function after flipping \(x_p\). Rewriting equation (3) yields the initialization of \(\delta v\) (see equation 5).

\[
\forall p \in [1, n], \ \delta v_p = (x^{(p)} - x_p) \frac{\partial f(x)}{\partial x_p} = \sum_{j \in \Gamma_j} [-f_j(x) + f_j(x^{(q)})] \tag{5}
\]

By lemma 3 initializing \(\delta v\) takes \(O(nk^2)\) time.

We now consider how \(\delta v\) can be updated after flipping a bit \(x_q\) based on second order partial derivatives, where \(q \in [1, n]\). The new vector of delta functions is denoted as \(\delta v^{(q)} = [\delta v_1^{(q)}, \delta v_2^{(q)}, \ldots, \delta v_n^{(q)}]\).

The definition of the second order partial derivative is given in equation (6).

\[
\frac{\partial^2 f(x)}{\partial x_p \partial x_p} = \frac{\partial f(x^{(q)})}{\partial x_p} \frac{\partial f(x)}{\partial x_p} = \frac{\partial f(x^{(q)})}{\partial x_p} \tag{6}
\]

Rewrite equation (5) and multiply both sides by \((x^{(p)} - x_p)(x^{(q)} - x_q)\), it follows

\[
\frac{\partial f(x^{(q)})}{\partial x_p} (x^{(p)} - x_p) = \frac{\partial f(x^{(q)})}{\partial x_p} (x^{(p)} - x_p) \tag{7}
\]

By equation (5) and equation (7), \(\delta v^{(q)}\) can be computed based on \(\delta v\) as in equation (5) (on the next page).

Note that if \(\{ j : p \in \Gamma_j \text{ and } q \in \Gamma_j \} = \emptyset\), the summation in equation (3) contains no terms and \(\delta v_p^{(q)} = \delta v_p\). In that case, no updates are required.

Another degenerative case is where \(p = q\), i.e., how to update \(\delta v_q\) after flipping \(x_q\). It is addressed in equation (8).

\[
\delta v_q^{(q)} = f(x^{(qq)}) - f(x^{(q)}) \text{ by equation (3)}
\]

\[
= f(x) - f(x^{(q)})
\]

\[
= \delta v_q \tag{9}
\]

The update to \(\delta v\) after flipping a bit \(x_q\) is summarized in equation (10) (on the next page). With \(\delta v_q\) available, we can recursively compute \(\delta v_p^{(q)}\) based on only the subfunctions that contain both \(x_q\) and \(x_p\).

Most of the above equations are similar to the ones in Li and Huang’s paper on AdaptG^2WSAT [10]. The main difference is: AdaptG^2WSAT computes the explicit form of the second order partial derivatives from the analytic form of clauses, which is not available in NK-landscapes. Instead, we calculate the \(\delta v\) (i.e., the difference in objective function after flipping a bit) from the lookup table corresponding to each subfunction.

For the implementation of D2LS, two more auxiliary data structures besides SubFuncListOne are required to perform updates as listed in equation 10. The first is Inter, the vector that lists the variables that interact with a given variable, e.g., Interv is a list of variables that interact with \(x_p\). The second is SubFuncListTwo: a hash table that maps an ordered pair (\(i, j\)), where \(i < j\), to a list of indices of subfunctions that contain both \(x_i\) and \(x_j\). Generating Inter and SubFuncListTwo is a one-time initialization cost for a given problem instance. We analyze this initialization cost in the next two lemmas.

**LEMMA 4.** Initializing Inter takes \(O(nk^2)\) time.

**PROOF.** Any given function containing \(k\) variables contributes to \(\binom{k}{2} = O(k^2)\) pair-wise interactions. Suppose \(x_i\) and \(x_j\) are two variables that both appear in a subfunction. We then need to insert \(i\) into Interv and insert \(j\) into Interv. To avoid duplicate variables in the list of interacting variables, a hash-table is used to store the list; a key will only be inserted when it does not exist yet.

The search and add operations both take \(O(1)\) time on average. As there are \(n\) subfunctions in total, the initialization of Inter takes \(O(nk^2)\).

**LEMMA 5.** Initializing SubFuncListTwo takes \(O(nk^2)\) time.

**PROOF.** For any given function \(f_j\) that contains \(k\), its index \(j\) is added \(\binom{k}{2} = O(n^2)\) times. \(n\) such subfunctions make the complexity for initialization \(O(nk^2)\).

The complexity of the initialization phase of D2LS is determined in theorem 1.

**THEOREM 1.** The initialization phase of D2LS runs in \(O(nk^2)\).

**PROOF.** This is obvious from lemma 3, lemma 4 and lemma 5.

We next prove why D2LS can achieve \(O(k^3)\) complexity per move in the update phase.

**THEOREM 2.** The complexity per move in the update phase is \(O(k^3)\).

**PROOF.** We consider how many subfunctions are involved in the summation in equation 10 and how many times the involved subfunctions are reused. For a subfunction \(f_j\) where a given bit \(x_q\) appears, there are \(k-1\) other bits in \(f_j\). This particular subfunction occurs in the summation exactly \(k-1\) times. In expectation, \(x_q\) bit appears in \(k\) subfunctions by lemma 1. Each subfunction evaluation takes \(O(k)\), since extracting related bits for the given subfunction takes \(O(k)\). Overall, the complexity for updates is \(O((k-1) \cdot k \cdot k) = O(k^3)\).

In Table 1, we summarize the complexity for the four algorithms in the current paper. There are two concerns while evaluating these complexity results. First, \(n\) plays a dominant role in growth of runtime, as \(n \gg k\) in most applications. Second, both preprocessing and initialization are one-time costs, while the update cost is recurring. Therefore, we shall mainly focus on the complexity per move (i.e., update complexity).
\[ \forall p \in [1, n], \quad \delta \nu_p^{(q)} \]
\[ = \delta \nu_p + \frac{\partial f(x)}{\partial x_p}(x^{(q)}_p - x_p)(x^{(p)}_p - x_p) \]
\[ = \delta \nu_p + \frac{\partial f(x^{(q)})}{\partial x_p}(x^{(p)}_p - x_p) - \frac{\partial f(x)}{\partial x_p}(x^{(p)}_p - x_p) \text{ by equation (7)} \]
\[ = \delta \nu_p + \sum_{j,p \in I_j \land q \in I_j} [-f_j(x^{(q)}) + f_j(x^{(p)})] - \sum_{j,p \in I_j} [-f_j(x) + f_j(x^{(p)})] \text{ by equation (3)} \]
\[ = \delta \nu_p + \sum_{j,p \in I_j \land q \in I_j} [f_j(x^{(q)}) - f_j(x^{(p)}) - f_j(x^{(q)}) + f_j(x)] + \sum_{j,p \in I_j \land q \in I_j} [f_j(x^{(q)}) - f_j(x^{(p)}) - f_j(x^{(q)}) + f_j(x)] \]
\[ = \delta \nu_p + \sum_{j,p \in I_j \land q \in I_j} [f_j(x^{(q)}) - f_j(x^{(p)}) - f_j(x^{(q)}) + f_j(x)] + \sum_{j,p \in I_j \land q \in I_j} [(f_j(x^{(q)}) - f_j(x^{(p)})) - (f_j(x^{(q)}) - f_j(x))] \]
\[ = \delta \nu_p + \sum_{j,p \in I_j \land q \in I_j} [f_j(x^{(q)}) - f_j(x^{(p)}) - f_j(x^{(q)}) + f_j(x)] + \sum_{j,p \in I_j \land q \in I_j} [0] \]
\[ = \delta \nu_p + \sum_{j,p \in I_j \land q \in I_j} [f_j(x^{(q)}) - f_j(x^{(p)}) - f_j(x^{(q)}) + f_j(x)] \]
\[ = \delta \nu_p + \sum_{j,p \in I_j \land q \in I_j} [f_j(x^{(q)}) - f_j(x^{(p)}) - f_j(x^{(q)}) + f_j(x)] \text{ if } \{j,p \in I_j \land q \in I_j \land p \neq q\} \neq \emptyset, \delta \nu_p, \text{ otherwise.} \]

\[ \delta \nu_p^{(q)} = \begin{cases} 
\delta \nu_p + \sum_{j,p \in I_j \land q \in I_j} [f_j(x^{(q)}) - f_j(x^{(p)}) - f_j(x^{(q)}) + f_j(x)] & \text{if } \{j,p \in I_j \land q \in I_j \land p \neq q\} \neq \emptyset, \\
\delta \nu_p & \text{otherwise.} \end{cases} \]

Table 1: Complexity Comparison among Implementations of First-Improvement Local Search Algorithms for NK-landscapes.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Preprocessing</th>
<th>Initialization</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neigh. Enum.</td>
<td>N/A</td>
<td>(O(n))</td>
<td>(O(n)^2)</td>
</tr>
<tr>
<td>Walsh-LS</td>
<td>(O(k^2n))</td>
<td>(O(kn))</td>
<td>(O(k^2))</td>
</tr>
<tr>
<td>D1LS</td>
<td>N/A</td>
<td>(O(kn))</td>
<td>(O(k^2n))</td>
</tr>
<tr>
<td>D2LS</td>
<td>N/A</td>
<td>(O(k^2n))</td>
<td>(O(k^3))</td>
</tr>
</tbody>
</table>

According to Table 1, D2LS has several advantages over the other algorithms. Unlike Walsh-LS, D2LS needs no preprocessing. As for the one-time initialization cost, D2LS takes time polynomial in \(k\) while Walsh-LS takes at least exponential time in \(k\). Most importantly, D2LS achieves a complexity per move that is both independent of \(n\) and polynomial in \(k\), which helps offset the initialization cost.

4. EMPIRICAL STUDIES

Complexity analysis reveals the worst case performance of algorithms. To determine performance in practice, we examine the actual runtimes of the algorithms on varying problems. Another goal is to shed light on how to choose a suitable FLS method depending on the characteristics of problem instances.

4.1 Selection of NK-landscape Instances

We test Walsh-LS, D1LS and D2LS on 22 randomly generated NK-landscape instances: 7 instances with \(n = 500\) and \(K = \{2, 4, 6, 8, 10, 12, 14\}\), 6 instances with \(n = 2000\) and \(K = \{2, 4, 6, 8, 10, 12\}\), 5 instances with \(n = 10000\) and \(K = \{2, 4, 6, 8, 10\}\), 4 instances with \(n = 50000\) and \(K = \{2, 4, 6, 8\}\). We chose these instances based on a limitation imposed by the sizes of files expressing problem instances. For all explored values of \(n\), the problems associated with the largest values of \(k\) required between 200M and 400M of storage. A file of an NK-landscape instance contains two parts: a \(n \times k\) matrix expressing the indices of variables that is included in each subfunction, and a \(n \times 2^k\) matrix representing the lookup tables for the evaluation values for \(n\) subfunctions. An instance file as a whole occupies \(nk + n2^k = O(n2^k)\) space.

4.2 Runtime Comparison among Local Search Algorithms

The purpose of our paper is to introduce a general technique that can be plugged into any FLS to algorithmically accelerate the execution of solving NK-landscape problems, rather than to propose a specific solver. As a consequence, we only run the three algorithms up to the first local optimum where there is no improving move in the neighborhood. We record the runtime data separately for the three phases. This allows us to validate the theoretical analysis for each phase directly from runtime data. Nonlinear regression is used to fit the sample points and quantitatively model the growth of runtime.

The experiment under every setting is repeated 10 times on servers configured with Intel Xeon E7-4830 processors at 2.13GHz and 128GB main memory. The three algorithms are implemented in Python 2.7.3.

Only Walsh-LS requires a preprocessing phase (i.e., the Walsh transformation); the preprocessing time for Walsh-LS is provided in Table 2. We expect the cost of the Walsh transform to be \(O(k2^k)\) \(n\). The two parameters \(n\) and \(k\) are related to the complexity of NK-landscape instances. To study the growth of runtime with re-
R with formula curve is generated using the nonlinear regression function from Figure 1: Regression plots for Walsh transformation times. The with formula \( t = a + b \cdot K^n + c \cdot 2^K \) for the left subfigure and with formula \( t = a + b \cdot n^c \) for the right, in which \( a, b \) and \( c \) are the parameters. The residual standard error (“RSE”) is reported to indicate the goodness of fit.

We therefore pick \( n = 500 \) for studying \( k \) and \( k = 2 \) for studying \( n \), and we also find the trends are consistent across different settings of \( n \) and \( k \). As illustrated in Figure 1, the time for the Walsh transform grows exponentially with \( k \) and grows linearly in \( n \), which supports the complexity results in Table 1. For the largest values of \( k \), preprocessing can take substantial time (up to 258 seconds).

We next present the actual time consumed by the initialization phases of the three algorithms in Table 2. We mark the one using the least initialization time in bold. D1LS dominates D2LS on all tested instances, while D2LS dominates Walsh-LS.

Figure 2 shows the regression fit to the initialization time. The three right subfigures indicate that the initialization time for all three methods is linear in \( O(n) \). The subfigures on the left indicate that D2LS and D1LS both take time polynomial in \( k \), while Walsh-LS takes at least exponential time in \( k \). The orders of polynomials in \( k \) for D2LS and D1LS almost exactly match the complexity results summarized in Table 1. According to the bottom-left subfigure in Figure 2, the initialization cost is in fact heavily affected by \( k \). The regression suggests the initialization time for Walsh-LS grows as \( O(k^3 \cdot 2^n) \) using our implementation. We have not fully optimized our implementation of the initialization code because this is a one time cost; nevertheless the initialization costs are not trivial for large \( K \). On the instance with \( n = 500 \) and \( k = 14 \), D1LS is 685714 times faster than Walsh-LS in the initialization phase. D2LS can also be 14117 times faster than Walsh-LS. D1SL is 50 times faster that D2LS in terms of initialization costs.

Table 2: Walsh transform times in seconds.

<table>
<thead>
<tr>
<th>( n )</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>0.077</td>
<td>0.21</td>
<td>0.76</td>
<td>2.9</td>
<td>12</td>
<td>48</td>
<td>194</td>
</tr>
<tr>
<td>2000</td>
<td>0.31</td>
<td>0.85</td>
<td>3</td>
<td>10</td>
<td>43</td>
<td>177</td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td>1.5</td>
<td>4.3</td>
<td>15</td>
<td>58</td>
<td>214</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50000</td>
<td>7.6</td>
<td>20</td>
<td>72</td>
<td>258</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NK-landscapes have randomly generated subfunctions, we could have also randomly generated \( n \) tables of \( 2^k \) Walsh coefficients with the proper distribution and have achieved exactly the same result. This would remove a significant amount of the initialization costs.

Because updates are done at every iteration of local search, the primary cost is the update cost per move. Since we are using first improvement local search, we do not need to track the best improving move; tracking the best improving move can dominate the update cost and overshadow other runtime costs. Since we are only concerned with the complexity per move, we record only the time of search from the initial starting point to the first local optimum. In addition, since all three local search methods execute FILS, the number of moves required to reach the first local optimum \( N_{move} \) and the final solution obtained should be the same in expectation across all three algorithms.

The search cost associated with the updates required to reach first local optimum \( T_{search} \) (the actual search time) is presented in
Table 3: Median initialization time (in seconds) for the three local search methods. The one taking the least time is marked in bold font.

<table>
<thead>
<tr>
<th>K</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>LS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D2LS</td>
<td>0.0099</td>
<td>0.023</td>
<td>0.039</td>
<td>0.061</td>
<td>0.097</td>
<td>0.13</td>
<td>0.17</td>
</tr>
<tr>
<td>D1LS</td>
<td>0.0012</td>
<td>0.0016</td>
<td>0.002</td>
<td>0.0024</td>
<td>0.0028</td>
<td>0.0028</td>
<td>0.0035</td>
</tr>
<tr>
<td>Walsh-LS</td>
<td>0.031</td>
<td>0.22</td>
<td>1.6</td>
<td>11</td>
<td>64</td>
<td>380</td>
<td>2400</td>
</tr>
<tr>
<td>n = 500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D2LS</td>
<td>0.044</td>
<td>0.095</td>
<td>0.18</td>
<td>0.31</td>
<td>0.51</td>
<td>0.73</td>
<td></td>
</tr>
<tr>
<td>D1LS</td>
<td>0.0048</td>
<td>0.006</td>
<td>0.0081</td>
<td>0.01</td>
<td>0.01</td>
<td>0.012</td>
<td></td>
</tr>
<tr>
<td>Walsh-LS</td>
<td>0.13</td>
<td>1.2</td>
<td>8.8</td>
<td>58</td>
<td>359</td>
<td>2344</td>
<td></td>
</tr>
<tr>
<td>n = 2000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D2LS</td>
<td>0.25</td>
<td>0.67</td>
<td>1.5</td>
<td>2.7</td>
<td>4.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D1LS</td>
<td>0.023</td>
<td>0.03</td>
<td>0.039</td>
<td>0.055</td>
<td>0.061</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Walsh-LS</td>
<td>0.93</td>
<td>9.1</td>
<td>6.1</td>
<td>409</td>
<td>2764</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n = 10000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D2LS</td>
<td>1.7</td>
<td>4.7</td>
<td>9.3</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D1LS</td>
<td>0.14</td>
<td>0.18</td>
<td>0.25</td>
<td>0.29</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Walsh-LS</td>
<td>6</td>
<td>48</td>
<td>344</td>
<td>2749</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n = 50000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Median initialization time (in seconds) for the three local search methods. The one taking the least time is marked in bold font.

Denote the time per move as $T_{\text{move}}$; we then have

$$T_{\text{search}} = T_{\text{move}} \times N_{\text{move}}.$$  \hspace{1cm} (11)

Since $N_{\text{move}}$ is the same in expectation for all three local search methods, Table 4 essentially compares $T_{\text{move}}$. We find Walsh-LS consistently achieves the lowest $T_{\text{search}}$ on all instances with $k$ up to 6, while on instances with greater $k$, D2LS is fastest. This is due to the fact that the hidden constant for D2LS is larger than for Walsh-LS, and the hidden constant is playing an even more important role when $k$ is small ($k \leq 6$ in our case). The biggest speedup of D2LS versus Walsh-LS is 10-fold (on instance with $n = 500$ and $k = 14$), while the greatest advantage when Walsh-LS is fastest is only 5-fold. As we would expect from the complexity analysis, the advantage of D2LS compared to Walsh-LS becomes increasingly pronounced as $k$ increases.

Notice that the complexity results concern $T_{\text{move}}$, while the presented empirical data concern $T_{\text{search}}$. Because of that, we should first investigate the growth of $N_{\text{move}}$ before studying $T_{\text{move}}$. This allows us to infer $T_{\text{move}}$ using equation (12).

$$T_{\text{move}} = \frac{T_{\text{search}}}{N_{\text{move}}}.$$  \hspace{1cm} (12)

The representative regression plots are displayed in Figure 3. From a random point in the search space, it takes roughly $\frac{n}{2}$ moves to reach a local optimum in expectation for NK-landscapes with $k = 0$. It is expected that $N_{\text{move}}$ grows linearly with $n$. The right subfigure in Figure 3 demonstrates that the growth of $N_{\text{move}}$ is linear with $n$. On the other hand, if we fix $n$ and increase $k$, we expect the landscape to be more rugged with more local optima in the search space. The left subfigure in Figure 3 suggests that $N_{\text{move}}$ in fact decreases linearly with $k$. Similar results are also reported in Table 2.4 on page 57 of Kauffman’s book [9] and by Ochoa et al. [15]. We are finally in a position to investigate the growth of $T_{\text{search}}$. Regression plots are shown in Figure 3 to demonstrate the growth of $T_{\text{search}}$ in terms of both $n$ and $k$. $T_{\text{search}}$ for both D2LS and D1LS are in $k$ (though the exponent for D1LS is a bit less than 1). By equation (12), we infer that $T_{\text{move}}$ for both D2LS and D1LS are quadratic in $k$. Surprisingly, the growth of $T_{\text{move}}$ for D2LS is even slower than our theoretical analysis, which suggests D2LS can have better performance in practice than the proven time bound. Nonetheless, the actual growth of $T_{\text{move}}$ again surpasses the analysis for Walsh-LS. The analysis provided by [19] suggests $T_{\text{move}} = O(k^3 \cdot 2^k)$, while the empirical data indicate that $T_{\text{move}}$ grows $O(k^3 \cdot 2^k)$ using our implementation. As shown in the right subfigures in Figure 3, $T_{\text{search}}$ for both D2LS and Walsh-LS are linear in $O(n)$. Together with the fact shown in Figure 3 that $N_{\text{move}} = O(n)$, $T_{\text{move}}$ is indeed independent of $n$ for D2LS and Walsh-LS. As for D1LS, the $O(n^2)$ growth of $T_{\text{search}}$ suggests $T_{\text{move}} = O(n)$, which again validates our previous theoretical analysis.

5. CONCLUSIONS

We proposed a second order partial derivatives based local search method, D2LS, for NK-landscapes. D2LS consists of two phases, an one-time initialization phase and a recurring update phase. The complexity per move in the update phase is $O(k^3)$, which is the best-known bound for NK-landscapes. We empirically compared
Table 4: Median update time (in seconds) spent up to the first local optimum. The one taking the least time among the three is marked in bold font.

D2LS with Walsh-LS. Empirical data showed that D2LS dominates Walsh-LS in the initialization phase on all tested instances. D2LS can in fact be about twenty thousand times faster than Walsh-LS on instances with large k. As for the update phase, the complexity per move for D2LS is O(k^3), while that for Walsh-LS is O(k^2 + 2^k). Empirical data suggested that D2LS can be more than 100 times faster than Walsh-LS. The preprocessing step in Walsh-LS can also require significant time. Thus, D2LS is particularly suitable for NK-landscapes with large n and moderately large k.

Some open questions remain. First, our method works especially well on instances with moderately large k, compared with Walsh-LS. On instances with k ≤ 6, however, Walsh-LS can be faster than D2LS in the update phase. Techniques can be used to map any pseudo-Boolean function to a quadratic pseudo-Boolean function with k = 2 in polynomial time [2]. This results in a problem with the same global optimum, but it does not induce the same landscape. If all we care about is the global solution to the problem, using this kind of transformation can be useful. It also has an initialization costs however. And this also destroys one of the desirable properties of NK-Landscapes: the ability to control the roughness and nonlinearity of the landscape.

Second, we showed that the second order partial derivatives based method is superior to the one based on first order partial derivatives. It is still unclear if the complexity per move can be further improved using even higher order partial derivatives.

6. ACKNOWLEDGEMENTS
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.

7. REFERENCES


---

Figure 4: Regression plots for update time. The curve is generated using the nonlinear regression function from R with formula $t = a + b \times K^c$ for bottom-left subfigure, with the formula $t = a + b \times K^c \times 2^K$ for two other left subfigures and with formula $t = a + b \times n^c$ for those on the right, in which $a$, $b$ and $c$ are parameters to adjust for the regression model. The residual standard error (“RSE”) is reported to indicate the goodness of fit.

[13] Vasco M. Manquinho and Olivier Roussel. The First