THESIS

A STEP TOWARD CONSTANT TIME LOCAL SEARCH FOR OPTIMIZING PSEUDO BOOLEAN FUNCTIONS

Submitted by

Wenxiang Chen

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2013

Master's Committee:

Advisor: L. Darrell Whitley
Co-advisor: Adele E. Howe

Margaret Cheney

ABSTRACT

A STEP TOWARD CONSTANT TIME LOCAL SEARCH
FOR OPTIMIZING PSEUDO BOOLEAN FUNCTIONS

Pseudo Boolean Functions (PBFs) are the objective functions for a wide class of hard optimization problems, such as MAX-SAT and MAX-CUT. Since these problems are **NP**-Hard, researchers and practitioners rely on incomplete solvers, such as Stochastic Local Search (SLS), for large problems. Best-Improvement Local Search (BILS) is a common form of SLS, which always takes the move yielding the highest improvement in the objective function. Generally, the more runtime SLS is given, the better solution can be obtained. This thesis aims at algorithmically accelerating SLS for PBFs using Walsh Analysis.

The contributions of this thesis are threefold. First, a general approach for executing an approximate best-improvement move in constant time on average using Walsh analysis, "*Walsh-LS*", is described. Conventional BILS typically requires examining all n neighbors to decide which move to take, given the number of variables is n. With Walsh analysis, however, we can determine which neighbors need to be checked. As long as the objective function is epistatically bounded by a constant k (k is the number of variables per subfunctions), the number of neighbors that need to be checked is constant regardless of problem size. An impressive speedup of runtime (up to 449×) is observed in our empirical studies.

Second, in the context of Walsh-LS, we empirically study two key components of SLS from the perspectives of both efficiency and effectiveness: 1) Local optimum escape method: hard random or soft restarts; 2) Local search strategy: first-improvement or best-improvement.

Lastly, on average we can perform approximate BILS using the mean over a Hamming region of arbitrary radius as a surrogate objective function. Even though the number of points is exponential in the radius of the Hamming region, BILS using the mean value of points in the Hamming region as a surrogate objective function can still take each move in time independent of n on average. According to our empirical studies, using the average over

a Hamming region as a surrogate objective function can yield superior performance results on neutral landscapes like NKq-landscapes.

ACKNOWLEDGEMENTS

I would like to thank my advisors Dr. Darrell Whitley and Dr. Adele Howe for their patient guidance. I would also like to thank Dr. Margaret Cheney for serving as my external committee member.

I would like to thank the Air Force Office of Scientific Research for funding this work. My research was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number FA9550-11-1-0088.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

Chapter 1

Introduction

1.1 Local Search using Walsh Analysis

Pseudo-Boolean functions (PBFs) [BH02] act as the objective functions for a wide variety of optimization problems arising from various areas. The applications of PBFs include maximum satisfiability (MAX-SAT) from computer science [Zha04], spin glasses from statistical mechanics [SK75], and NK-landscapes from theoretical biology [KW89]. Optimizing PBFs in general is **NP**-hard [KS11][Sut11], implying that no polynomial-time algorithm has been found. Instead of solving these problems using complete algorithms, which takes exponential time in the worst case with respect to the size of input, researchers and practitioners settle for good suboptimal solutions using metaheuristics [BR03]. Metaheuristics are designed to return a solution of good quality (not necessarily the optimal one) in a reasonable amount of time.

Stochastic Local Search (SLS) [HS04] is a simple yet effective heuristic that iteratively improves a candidate solution through choosing better moves among neighbors under a predefined neighborhood operator. Best-Improvement Local Search (BILS) is a common form of SLS. BILS chooses the most improving move among n neighbors. Conventionally, each move requires a complete scan over all n neighbors, and examining each neighbor takes $O(n)$ time on PBFs. Overall, the conventional approach takes $O(n^2)$ time to execute one best-improvement move on PBFs.

Since SLS with more runtime usually returns a better solution, we first present a Walsh analysis approach to algorithmically accelerate the conventional implementation of BILS. Second, empirical evidence about the impact of two key components of BILS, local optimum escape method and move selection strategy, on effectiveness and efficiency are presented. Last, we demonstrate how smoothing the search space by replacing the standard objective

function with the mean value over a Hamming region can improve the solution quality with *no extra cost* in runtime.

Walsh analysis allows us to design an *approximate constant time BILS* on uniform random PBF instances, which we call "Walsh-LS" . The Walsh transform [Gol89a][Gol89b] decomposes an objective function into a linear combination of Walsh bases. The corresponding coefficients for the Walsh bases serve to capture the interaction among variables. Walsh analysis is a set of techniques built on the basis of the Walsh transform. In this thesis, we implement a general approach that maintains a proxy vector for the objective function values of neighbors, and utilizes Walsh coefficients to update the proxy vector during search instead of recomputing the objective function values of neighbors every time after flipping a bit. Assume on a uniform random instance, the proxy vector can be updated in $O(1)$ time in expectation after flipping a bit. Since the list of improving bits can be $O(n)$, only a constant number of bits are sampled from the list to select the *approximate* best improving move. We therefore reduce the average amount of computation required for taking one best-improvement move from $O(n^2)$ to $O(1)$ on uniform random PBF instances. [2]

We also explore two other ways of achieving the expected $O(1)$ complexity per move: 1) replacing hard random with soft restarts; 2) replacing best-improving local search with first-improving local search. The impact of these changes on solution quality is investigated in our empirical study.

Sutton *et al.* [SHW10] conduct an initial study of bringing theoretical results of Walsh analysis into algorithm development. They introduce Directed Plateau Search (DPS) [SHW10]. DPS employs the mean value of the objective function over the localized Hamming region of the search space as a surrogate "gradient" function. They empirically show that formal analysis of the search space structure is able to direct search to escape plateaus (connected equal moves) in a more principled fashion. Rather than using the mean value over a Hamming

[2]This thesis is primarily based on two conference papers that we have published [WC12][WCH12].

region only to assist the search in escaping plateaus, we explore the possibility of directly replacing the objective function with the surrogate function. Surrogate functions can be defined as the mean value over Hamming regions. One can view the replacement of the objective function as a transformation of the search space. We argue that the search space after the transformation is smoother, has fewer plateaus, and therefore is easier for search.

In this thesis, we focus on NK-landscapes as optimization problems whose objective functions are a particular class of PBFs. There are two often studied classes of NK-landscapes: unrestricted NK-landscapes and nearest-neighbor NK-landscapes [Pel10]. Nearest-neighbor NK-landscapes restrict the pattern of neighbors to be consecutive in position. This class of NK-landscapes can be solved exactly using dynamic programming as discussed in [PSG+09]. Unrestricted NK-landscapes, however, have been proven **NP**-hard [Wei96]. We limit our investigation to unrestricted NK-landscapes, since these problems are often beyond complete solvers and require the power of SLS.

1.2 Related Work

The idea of applying the Walsh transform to search dates back to 1980s. Bethke [Bet81] introduces a direct method to compute the average schema fitness value (schema is a partition of the search space [Hol92]). The direct calculation of the average fitness value is significant because genetic algorithms compare schemata and allocate more computational resources to those with higher fitness. The efficient computation of average schema fitness has been used to help explain the search behavior of genetic algorithms.

Walsh analysis was not practical however, as the Walsh transform requires complete enumeration over the entire search space. Rana *et al.* [RHW98] argue that for k-bounded pseudo-Boolean functions, the cost of the Walsh transform is $O(m2^k)$, given that m is the number of subfunctions and k is the maximum number of variables in all subfunctions. This can be achieved by performing the Walsh transform on each subfunction and summing up the Walsh polynomial to obtain the results for the overall function. This discovery makes many calculations based on Walsh analysis tractable for practical purposes.

Walsh analysis can be employed to exactly calculate summary statistics of the entire search space or partitions of the search space. Heckendorn *et al.* [HRW99] introduce an approach to compute summary statistics for the entire search space for a generalization of MAX-SAT and NK-landscapes, called *embedded landscapes*, in polynomial time. They later propose a method to directly compute summary statistics for hyperplanes (partitions of the search space) [Hec02]. Recently, Zhou *et al.* [ZHS08] extend these results to a black-box function with a domain of arbitrary-cardinality under the bounded epistasis. Rather than considering summary statistics for the entire space or hyperplanes, Sutton *et al.* [SWH12] concentrate on a localized subspace. They [SHW09] disclose that MAX-k-SAT can be decomposed into k elementary landscapes [Gro92] [Sta95]. A landscape is called *elementary*, if the mean value over the neighborhood of any candidate solution $\mathbf{x}$, denoted as $\mathbb{E}(f(\mathbf{y}))$, can be computed directly based on $\mathbf{x}$'s objective function value $f(\mathbf{x})$ using equation (1.1).

$$\mathbb{E}(f(\mathbf{y})) = f(\mathbf{x}) + \frac{k}{d}(\bar{f} - f(\mathbf{x})) \tag{1.1}$$

where $\mathbf{y} \in N(\mathbf{x})$, $N(\mathbf{x})$ is the neighborhood of x, k is a constant relative to the problem, and d is the size of neighborhood. They show that the elementary landscape decomposition of MAX-k-SAT allows us to directly compute the expectation of the objective function evaluated across neighboring points without enumeration. This result is then used to prove previously unknown bounds for local maxima and plateau width in the 3-SAT search space.

Sutton *et al.* [SWH12] further propose a general method to compute moments of k-bounded PBFs over a Hamming region of arbitrary radius. They discover that the moments of k-bounded PBFs over Hamming regions of arbitrary radius can be applied to approximate the fitness distribution over Hamming regions by solving a system of linear equations.

Walsh polynomials can also be applied to compute some "global" problem hardness measures. These measures typically project features of the entire search space onto a single quantity. Sutton *et al.* [SWH09] introduce a polynomial time computation of the autocorrelation function for k-satisfiability landscapes. Auto-correlation [Sta96] is a statistical quantity that captures ruggedness of a fitness landscape, and it is one of the problem hardness measures that are widely used to connect the performance of algorithms to problem difficulty.

Another well-known measure of problem hardness is fitness distance correlation [JF95], which quantifies the correlation between fitness and the distance from a current candidate solution to the closest optimum. Chicano and Alba provide a closed-form formula for calculating the fitness-distance correlation for PBFs with one global optimum using Walsh analysis [CA12].

Walsh analysis also can be applied in understanding a search algorithm's behavior. For instance, Walsh analysis can be used to compute the expectation curves for the uniform crossover operator [CWA12] and the bit-flip mutation operator [CA11].

Even though there are quite some interesting theoretical works on Walsh analysis, relatively little work is devoted to exploring Walsh analysis to actually *improve* the search process, either for effectiveness or efficiency. My thesis explores how the Walsh analysis can be utilized to speed up a widely used algorithm, and furthermore, to improve the solution quality of local search algorithms.

Chapter 2

Background

2.1 Pseudo-Boolean Functions

Pseudo-Boolean Functions (PBFs) map an n-dimensional Boolean vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ to a value f. x_i ($i \in [1, n]$) is the i^{th} element of $\mathbf{x}$[3]. Every PBF can be uniquely written as a *multi-linear polynomial* [BH02]:

$$f(\mathbf{x}) = f(x_1, \ldots, x_n) = \sum_{S \subseteq \mathbf{V}} c_s \prod_{j \in S} x_j. \tag{2.1}$$

where $\mathbf{V} = \{1, 2, \ldots, n\}$, $\mathcal{B} = \{0, 1\}$, $\mathbf{x} \in \mathcal{B}^n$. Conventionally, $\prod_{j \in \emptyset} x_j = 1$. The size of the largest subset S such that $c_s \neq 0$ is the degree of *PBF* f, denoted as $deg(f)$.

In the field of optimization, however, PBFs are more commonly represented as *posiforms* [BH02]. Posiforms build on the basis of *literals*. Let the complement of a Boolean variable x_i be $\bar{x}_i$, i.e., $\bar{x}_i \stackrel{def}{=} 1 - x_i$ for $i \in \mathbf{V}$. Both x_i and $\bar{x}_i$ are called *literals*. The set of literals is denoted as $\mathbf{L} = \{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$. The following equation demonstrates the representation of PBFs in posiforms:

$$\phi(\mathbf{x}) = \sum_{T \subseteq \mathbf{L}} a_T \prod_{u \in T} u, \tag{2.2}$$

where T is a subset of L. Even though it might be more intuitive to interpret PBFs in posiforms, there can be different posiforms representing the same PBF.

Example 1 ([BH02])**.** *Equation* (2.3) *and Equation* (2.4) *show two different posiforms for the same pseudo-Boolean function, while Equation* (2.5) *shows its unique multi-linear polynomial form.*

$$\psi_1(\mathbf{x}) = 5x_1 + 4\bar{x}_1\bar{x}_2x_3 + 7x_1x_2x_4 + 9x_3\bar{x}_4, \tag{2.3}$$

[3]In this thesis, **bold** fonts are used for vectors while normal fonts with subscripts are used for elements in a vector.

$$\psi_2(\mathbf{x}) = x_1 + 4x_1x_2 + 4x_1\bar{x}_2\bar{x}_3 + 7x_1x_2x_4 + 4\bar{x}_2x_3 + 9x_3\bar{x}_4. \tag{2.4}$$

$$f(x_1, x_2, x_3, x_4) = 5x_1 + 13x_3 - 4x_1x_3 - 4x_2x_3 - 9x_3x_4 + 4x_1x_2x_3 + 7x_1x_2x_4 \tag{2.5}$$

One can easily verify the claim in Example 1 by enumerating all possible assignments for Equation (2.3), Equation (2.4) and Equation (2.5). Also note that all representations in Example 1 have degree of 3, which indicates the degree of a PBF is an inherent property and is independent of its representation.

The optimization problem of maximizing or minimizing a PBF is **NP**-Hard in general [Sut11]. Based on the widely believed conjecture that $\mathbf{P} \neq \mathbf{NP}$ [For09], there is no polynomial time algorithm for problems that belong to the **NP** class. Regardless of the fact that some problems in the **NP** class can be solved with a polynomial-time algorithm [CKT91], optimizing PBFs can take exponential time of input size n in the worst case. Finding an exact solution can be prohibitively time consuming. Incomplete solvers thus become a compelling topic for researchers and a more reasonable choice for practitioners.

2.2 NK-landscapes

NK-landscapes are the class of PBFs under investigation in this thesis. NK-Landscapes [KL87] are constructed from N subfunctions, where each subfunction $f_j(\mathbf{x}_{\mathbf{I}_j})$ evaluates a substring $\mathbf{x}_{\mathbf{I}_j}$ of binary string $\mathbf{x}$. $\mathbf{x}_{\mathbf{I}_j}$ includes x_j and k other randomly chosen bits, where $\mathbf{I}_j$ is a set of $k+1$ indices that extract a substring from the full binary string $\mathbf{x}$. In other words, each subfunction $f_j(\mathbf{x}_{\mathbf{I}_j})$ is a PBF of length $K = k+1$, and the entire NK-Landscape is a K-bounded pseudo-Boolean function, where

$$f(\mathbf{x}) = \sum_{j=1}^{N} f_j(\mathbf{x}_{\mathbf{I}_j}). \tag{2.6}$$

2.3 Stochastic Local Search for NP-Hard Problems

Local search algorithms move from one candidate solution to its neighboring candidate solution in the search space by applying local changes, until an optimal solution is found

or a time limit has been reached. SLS takes advantage of randomized choices in generating or selecting moves. The best-performing algorithms for many NP-Hard problems are SLS algorithms, such as LKH [Hel00] for the Traveling Salesperson Problem (TSP) and WalkSAT for satisfiability problem [SKC94].

Algorithm 1 [HS04] demonstrates a general outline of SLS for a maximization problem[4]. SLS takes a problem instance π' as input and returns the solution sol. We denote the search space for π' as $\mathsf{S}(\pi')$, the optimal solution sets as $\mathsf{S}'(\pi')$, candidate solution(s) as s, best-so-far (bsf) candidate solution as $\hat{\mathsf{s}}$, and a memory state as m (such as in Tabu Search [GL97]).

Algorithm 1: $\mathsf{sol} \leftarrow$ SLS-MAXIMIZATION(π')

```
1 (s,m)← INIT(π',m) ;        // initialize candidate solution and memory state
2 ŝ ← BEST(π', s) ;              // save best initial solution as bsf solution
3 while TERMINATE(π',s,m)==False do
4 |   (s,m) ← STEP(π',s,m);                  // apply (randomized) local changes
5 |   if f(π', s) > f(π',ŝ) then
6 |   |   ŝ ← s;                                 // update bsf candidate solution
7 return ŝ;
```

In this thesis, we address Best-Improvement Local Search (BILS), a common form of SLS used with the one bit-flip neighborhood operator for PBFs, a single search thread and no external memory. BILS initializes with a single point in the search space, and always moves it to the neighbor that yields the greatest improvement in the objective function (the so-called best-improvement move) if possible, otherwise after hitting a local optimum it perturbs the candidate solution in hope of ending up with a better local optimum (as pictured in Figure 2.1). The BILS is sketched in Algorithm 2. BESTIMPR function in line 4 of Algorithm 2 examines all n neighbors of s under the one bit-flip neighborhood operator, and returns the most improving neighbor if any, otherwise it returns s.

[4]In all pseudocodes in this thesis, sans-serif fonts represent vectors, *italic* fonts represent variables and ALL-CAPITAL-LETTERS fonts present functions.

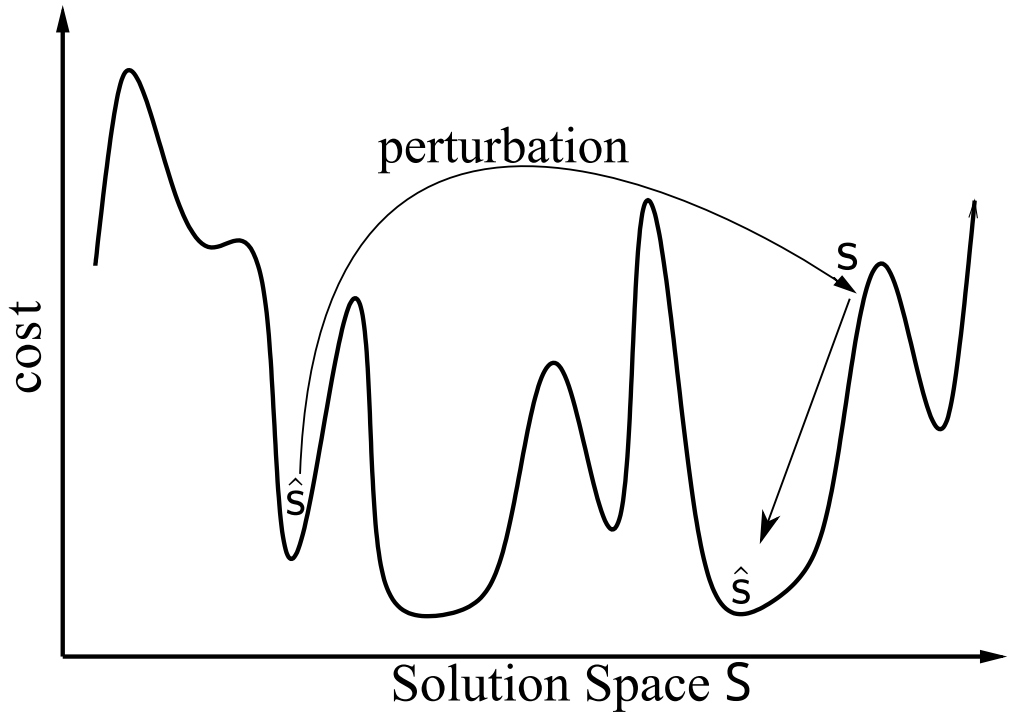

Figure 2.1: Pictorial View of Best-Improvement Local Search [LMS03]

Algorithm 2: sol ← BILS(π', escape)

```
1  s ← INIT(π') ;                                   // initialize candidate solution
2  ŝ ← s;
3  while TERMINATE(π',s)=False do
4      s' ← BESTIMPR(π',s);
5      if s' = s then                               // no improving move in neighborhood
6          if f(π', s) > f(π',ŝ) then
7              ŝ ← s;                               // update bsf candidate solution
8          s ← PERTURBS(s, escape) ; // local optimum:  perturbs s
9      else move over to the best improving neighbor
10         s ← s'
11 return ŝ;
```

2.4 Walsh Analysis for Pseudo-Boolean Functions

The Walsh transform [Wal23] (also called Hadamard transform) is an example of a discrete Fourier Transform [BM67]. It decomposes any given PBF $f(\mathbf{x})$ defined over $\mathcal{B}^n \rightarrow \mathcal{R}$ into a

linear combination of orthonormal bases $\psi_{\mathbf{i}}(\mathbf{x})$, where $\mathbf{i}, \mathbf{x} \in \mathcal{B}^n$, as shown in Equation (2.7).

$$f(\mathbf{x}) = \sum_{\forall \mathbf{i} \in \mathcal{B}^n} w_{\mathbf{i}} \psi_{\mathbf{i}}(\mathbf{x}) \tag{2.7}$$

where $w_i \in \mathcal{R}$ is called the *Walsh coefficient* and $\psi_{\mathbf{i}}(\mathbf{x})$ is called the *Walsh function* that generates a sign as defined in Equation (2.8).

$$\psi_{\mathbf{i}}(\mathbf{x}) = (-1)^{\mathbf{bc}(\mathbf{i} \wedge \mathbf{x})} = (-1)^{\mathbf{i}^T \mathbf{x}} \tag{2.8}$$

$\mathbf{bc}(\mathbf{i})$ counts the number of 1's in bit string $\mathbf{i}$. The Walsh coefficients are then added or subtracted in the calculation of $f(\mathbf{x})$ depending on the sign generated by the Walsh function $\psi_{\mathbf{i}}(\mathbf{x})$. *The Walsh transform* enables us to compute Walsh coefficients directly from $f(\mathbf{x})$, as shown in Equation (2.9).

$$w_{\mathbf{i}} = \frac{1}{2^n} \sum_{\forall \mathbf{j} \in \mathcal{B}^n} f(\mathbf{j}) \psi_{\mathbf{i}}(\mathbf{j}) \tag{2.9}$$

Since the summation in Equation (2.9) goes over all possible bit strings of length n, direct computation of Walsh coefficients requires the enumeration of the entire search space. The exponential cost renders the Walsh transform intractable for practical purposes. However, Rana *et al.* [RHW98] take advantage of the linearity of the Walsh transform. They show that the Walsh transform can be performed on a linear combination of subfunctions, therefore reducing the complexity of the Walsh transform from $O(2^n)$ to $O(m2^k)$, where m is the number of subfunctions (assuming the posiform representation) and $k = deg(f)$. In addition, they prove that the number of non-zero Walsh coefficients is also $O(m2^k)$. In other words, the complexity of performing the Walsh transform on PBFs is determined by the subfunction with the maximum number of variables.

Fortunately, in many real-world applications of optimizing PBFs such as industrial instances of MAX-SAT [ABJ13] [BF98], k is typically very small (2 or 3) and $n \gg 2^k$. The subfunction-based Walsh transform is efficient on these problems. In this thesis we mostly address PBFs with small K, yet still shed some light on the scalability of Walsh analysis on larger K.

Chapter 3

Average Constant Time Approximate Best-Improvement Local Search

Taking one best-improvement move (which corresponds to Line 4 of Algorithm 2) in Best-Improvement Local Search (BILS) requires checking all n neighbors of the current candidate solution `s` under the single bit-flip neighborhood operator. In this chapter, we describe an approximate best-improvement local search algorithm that only takes constant time per move in expectation on random instances of unrestricted NK-landscape. We name this algorithm "Walsh-LS", as it is a local search algorithm based on Walsh analysis. Walsh-LS was initially proposed by Whitley [Whi11] to handle MAX-SAT problems. It was later extended to NK-landscapes domain by Whitley and Chen [WC12]. To make the thesis self-contained, we present a new proof for the main results in [WC12] in the theory part of this chapter.

The rest of this chapter is organized as follows. First, we review current implementations of BILS for NK-landscapes and analyze its complexity per move. Second, the Walsh-LS algorithm is introduced. We discuss how to execute BILS entirely on the basis of the Walsh transform. Third, we investigate the complexity per move of Walsh-LS and derive analytic bounds of cost per best-improvement move for both the average case and the worst case. Last, we study empirically how much speedup Walsh-LS can achieve over an implementation of BILS on NK-landscapes with various settings of N and K.

3.1 BILS Implementation

A naive implementation (denoted as *naive-BILS* for convenience) makes no use of the characteristics of k-bounded PBFs and takes $\Theta(n^2)$ time per move (page 273 in [HS04]). BestImpr function at line 4 in algorithm 2 is critical for the complexity per move of BILS. Also, it is exactly the place where naive-BILS behaves naively.

We present the naive-BILS's implementation of BestImpr in algorithm 3. Rather than using problem instance π' as an abstract input, we detail the inputs as the number of variables n, the number of subfunctions m, the evaluation function f, the current candidate solution s and the evaluation value $fitS$ associated with s. The m subfunctions f_j that compose f are also needed for evaluating a candidate solution. Let c be the subfunction to variable ratio (i.e., $c = \frac{m}{n}$). Clearly, algorithm 3 runs in $O(mn) = O(c * n^2) = O(n^2)$ time.

Algorithm 3: $\mathsf{bestS}' \leftarrow$ Naive-BestImpr(n, m, f, s, $fitS$)

```
1  bestFit ← fitS;
2  bestSList ← {s};
3  for i ← 1 to n do
4      s' ← Flip(s, i) ;                    // flip the i^th bit of binary string s
5      fitCur ← 0;
6      for j ← 1 to m do                          // sum over each subfunction
7          fitCur ← fitCur + f_j(s');
8      if fitCur > bestFit then                          // new best improving
9          bestSList ← {s'};
10         bestFit ← fitCur;
11     else if fitCur = bestFit and fitCur > fitS then          // equally best
12         bestSList ← bestSList ∪ {s'}
13 return RandomChoice(bestSList) ;                  // break ties arbitrarily
```

Each subfunction f_j contains exactly k variables. If the flipped bit i is not in the set of k variables, f_j will remain unchanged. This suggests that naive-BILS redundantly re-evaluates some (potentially large number of) subfunctions to compute f.

3.1.1 An Improved Implementation of BILS

Instead of re-evaluating all subfunctions to compute f, we consider only re-evaluating the subfunctions that are affected by the flipped bit. In this way, we can reduce the cost per move. In some sense, we are trying to *partially evaluate* a candidate solution on a subfunction f_j basis rather than on a function f basis. Motivated by this idea, an improved implementation of BILS based on partial evaluation, *PE-BILS*, is introduced.

In an NK-landscape instance, n subfunctions are evaluated and summed up together to construct the objective function. Each subfunction involves $k = K + 1$ variables. Let the current candidate solution be $\mathbf{x}$, and its adjacent neighbor after flipping the p^{th} bit be $\mathbf{x}^{(p)}$, i.e., $\mathbf{x}^{(p)} \stackrel{def}{=} \mathbf{x}[x_p \leftarrow \bar{x}_p]$, where $p \in [1, n]$.

We capture the changes in evaluation function f after flipping the p^{th} bit using equation (3.1).

$$\begin{aligned} f(\mathbf{x}^{(p)}) - f(\mathbf{x}) &= \sum_{j=1}^{n} f_j(\mathbf{x}^{(p)}_{\mathbf{I}_j}) - \sum_{j=1}^{n} f_j(\mathbf{x}_{\mathbf{I}_j}) \\ &= \sum_{j \in [1,n] \wedge p \in \mathbf{I}_j} (f_j(\mathbf{x}^{(p)}_{\mathbf{I}_j}) - f_j(\mathbf{x}_{\mathbf{I}_j})) \end{aligned} \tag{3.1}$$

We shall then partially evaluate $f(\mathbf{x}^{(p)})$ as in equation (3.2).

$$f(\mathbf{x}^{(p)}) = f(\mathbf{x}) + \sum_{j \in [1,n] \wedge p \in \mathbf{I}_j} (f_j(\mathbf{x}^{(p)}_{\mathbf{I}_j}) - f_j(\mathbf{x}_{\mathbf{I}_j})) \tag{3.2}$$

We now present PE-BILS in algorithm 4. PE-BILS requires $\mathbf{P}$, a length-n vector of lists. $\mathbf{P}_j$ contains the list of indices of all subfunctions that contain the j^{th} variable. $\mathbf{P}$ needs to be constructed before performing BILS, and requires a $\Theta(m * k)$ one-time cost.

Algorithm 4: $\mathsf{bestS}' \leftarrow$ PE-BILS(n, m, f, s, $fitS$, $\mathbf{P}$)

```
1  bestFit ← fitS;
2  bestSList ← {s};
3  for i ← 1 to n do
4      s' ← FLIP(s, i) ;                    // flip the i^th bit of binary string s
5      fitCur ← fitS;
6      for j ← 1 to m and j ∈ P_i do        // sum subfunctions containing i^th bit
7          fitCur ← fitCur + f_j(s') − f_j(s) ;          // by equation (3.2)
8      if fitCur > bestFit then                          // new best improving
9          bestSList ← {s'};
10         bestFit ← fitCur;
11     else if fitCur = bestFit and fitCur > fitS then   // equally best
12         bestSList ← bestSList ∪ {s'}
13 return RANDOMCHOICE(bestSList) ;                      // break ties arbitrarily
```

Lemma 1. *The complexity per move for PE-BILS is $O(n)$ on any k-bounded pseudo-Boolean function.*

Proof. Line 4 to line 7 in algorithm 4 traverses $\mathbf{P}$. Therefore, the number of iterations is in fact the number of all indices in $\mathbf{P}$, i.e., $\sum_{j=1}^{m} |\mathbf{P}_j|$. On the other hand, the index of each subfunction is added to $\mathbf{P}$ exactly k times since there are k variables in a subfunction. We thus have $\sum_{j=1}^{m} |\mathbf{P}_j| = m * k = c * n * k = O(n)$. Algorithm 4 describes the operations needed for taking one best improvement move, hence the complexity per move for PE-BILS is $O(n)$ on all k-bounded pseudo-Boolean functions. □

As PE-BILS is provably superior to naive-BILS, we will use PE-BILS as the baseline for evaluating the runtime of Walsh-LS.

3.2 Walsh-LS: Best-Improvement Local Search based on Walsh Analysis

After a best-improvement move is taken (flipping one bit x_i), only those subfunctions that reference x_i change, and all the other subfunctions remain the same. In the worst case, where all subfunctions contain the bit x_i, the number of subfunctions that changes their evaluations is n. On a uniform random instance of NK-landscapes, however, only a constant number of subfunctions change their evaluations in expectation. With approximation in selecting the best move from the list of improving moves and maintaining extra data structures, the $O(1)$ average complexity can be achieved using Walsh analysis.

We apply a Walsh decomposition of the objective function to expose variable interactions. The information about interactions is used to construct a *relative objective function* that keeps the difference in the objective function between the current candidate solution and its n neighbors. The relative objective function computes which neighbors can yield an improving move after a single bit flip. On a uniform random instance, in expectation only a constant number of the neighbors should change the relative objective function after a bit flip. A list of improving moves with respect to the current solution is also maintained. The

length of this list can be $O(n)$ in the worst case. With approximation in selecting the best move (i.e., sampling from improving bits list instead of doing a full scan over the list), the best-improvement move can be performed in constant time on average.

The concept of relative objective function relates to the *score vector* that is widely used in SAT community. The score of a variable p with respect to $\mathbf{x}$, $score_{\mathbf{x}}(p)$, is the decrease of the objective function when p is flipped. Modern incomplete SAT solvers such as GSAT [SLM92] and AdaptG^{2}WSAT [LH05] have utilized the score vector to efficiently determine next move to take.

3.2.1 Compute Difference between Adjacent Candidate Solutions in Standard Fitness Space

We start with computing the relative objective functions for all n neighbors of current candidate solutions.

Lemma 2 (Section 2 in [WC12])**.** *For any* $\mathbf{i}, \mathbf{x} \in \mathcal{B}^n$*,*

$$\forall p \in [1, n],\ \psi_{\mathbf{i}}(\mathbf{x}^{(p)}) - \psi_{\mathbf{i}}(\mathbf{x}) = \begin{cases} -2\psi_{\mathbf{i}}(\mathbf{x}) & \text{if } i_p = 1, \\ 0 & \text{if } i_p = 0. \end{cases}$$

Proof. By definition in equation (2.8),

$$\psi_{\mathbf{i}}(\mathbf{x}^{(p)}) = (-1)^{\mathbf{bc}(\mathbf{i} \wedge \mathbf{x}^{(p)})}.$$

No matter what x_p originally is, flipping x_p will change the parity of the number of 1's in $\mathbf{x}$. If $i_p = 1$, the change of parity is captured by $\mathbf{i} \wedge \mathbf{x}^{(p)}$,

$$\psi_{\mathbf{i}}(\mathbf{x}^{(p)}) = (-1)^{\mathbf{bc}(\mathbf{i} \wedge \mathbf{x}^{(p)})} = (-1)(-1)^{\mathbf{bc}(\mathbf{i} \wedge \mathbf{x})} = -\psi_{\mathbf{i}}(\mathbf{x}).$$

Otherwise $i_p = 0$, the change of parity is *not* captured by $\mathbf{i} \wedge \mathbf{x}^{(p)}$.

$$\psi_{\mathbf{i}}(\mathbf{x}^{(p)}) = (-1)^{\mathbf{bc}(\mathbf{i} \wedge \mathbf{x}^{(p)})} = (-1)^{\mathbf{bc}(\mathbf{i} \wedge \mathbf{x})} = \psi_{\mathbf{i}}(\mathbf{x}).$$

□

By equation (2.8), calculating the sign (Walsh basis) for Walsh coefficients every time takes $O(n)$ time. In order to achieve the $O(1)$ bound, we maintain the Walsh coefficient along with its corresponding sign. We shall let $w'_{\mathbf{i}}(\mathbf{x}) = w_{\mathbf{i}}\psi_{\mathbf{i}}(\mathbf{x})$. We call $w'_{\mathbf{i}}(\mathbf{x})$ *Walsh terms*, then $\mathbf{w}'(\mathbf{x})$ is the *vector of Walsh terms*. There are $O(n * 2^k)$ non-zero Walsh coefficients for a NK-landscape instance, thus $\mathbf{w}'(\mathbf{x})$ is a vector of length $O(n * 2^k)$.

Lemma 3 (Lemma 1 in [WC12])**.** *For any* $\mathbf{i}, \mathbf{x} \in \mathcal{B}^n$*, the relative objective function value of* $\mathbf{x}$ *after flipping its* p^{th} *bit is*

$$f(\mathbf{x}^{(p)}) - f(\mathbf{x}) = -2 \sum_{\mathbf{i} \in \mathcal{B}^n \wedge i_p = 1} w'_{\mathbf{i}}(\boldsymbol{x}),$$

where $p \in [1, n]$.

Proof.

$$\begin{aligned} f(\mathbf{x}^{(p)}) - f(\mathbf{x}) &= \sum_{\forall \mathbf{i} \in \mathcal{B}^n} w_{\mathbf{i}}\psi_{\mathbf{i}}(\mathbf{x}^{(p)}) - \sum_{\forall \mathbf{i} \in \mathcal{B}^n} w_{\mathbf{i}}\psi_{\mathbf{i}}(\mathbf{x}) \quad \text{By equation (2.7)} \\ &= \sum_{\mathbf{i} \in \mathcal{B}^n} w_i(\psi_{\mathbf{i}}(\mathbf{x}^{(p)}) - \psi_{\mathbf{i}}(\mathbf{x})) \\ &= -2 \sum_{\mathbf{i} \in \mathcal{B}^n \wedge i_p = 1} w'_{\mathbf{i}}(\mathbf{x}) \quad \text{By lemma 2} \end{aligned}$$

The above derivation holds for $\forall p \in [1, n]$. □

In lemma 3, direct computation of the relative objective function for all n possible values of p takes at least $O(n)$ time. We again overcome this by maintaining an extra data structure and updating it accordingly during search. We shall let $S_p(\mathbf{x}) = \sum_{\mathbf{i} \in \mathcal{B}^n \wedge i_p = 1} w'_{\mathbf{i}}(\mathbf{x})$, by lemma 3 we have

$$f(\mathbf{x}^{(p)}) - f(\mathbf{x}) = -2S_p(\mathbf{x}) \quad \text{(Lemma 1 in [WC12])} \tag{3.3}$$

The relative objective function $f(\mathbf{x}^{(p)}) - f(\mathbf{x})$ can serve as a proxy for $f(\mathbf{x}^{(p)})$, because $f(\mathbf{x})$ is invariant as p varies. For a given candidate solution $\mathbf{x}$, all its n neighbors can be represented via the *proxy vector* $\mathbf{S}(\mathbf{x})$ of length n. Selecting the optimum $S_p(\mathbf{x})$ yields the best-improvement move in the neighborhood of $f(\mathbf{x})$. $S(\mathbf{x})$ is only a constant difference from $score_{\mathbf{x}}$. More precisely, $-2S(\mathbf{x}) = score_{\mathbf{x}}$.

One can determine whether flipping any given bit will yield an improving move entirely from the proxy vector $\mathbf{S}(\mathbf{x})$. Of course, we shall not check the entire $\mathbf{S}(\mathbf{x})$ vector of length n to decide which bit to flip, because that would break the $O(1)$ time bound. Instead, we keep track of the *improving moves* in a dynamic-length list $\mathbf{Impr}(\mathbf{x})$, which is a subset of all n possible positions $\{1, 2, \ldots, n\}$. At the initialization stage, we have to scan the whole vector $\mathbf{S}(\mathbf{x})$ for the purpose of constructing the initial $\mathbf{Impr}(\mathbf{x})$. After that, however, we only need to examine those positions (only a constant number, which we will address in detail later) in the vector $\mathbf{S}(\mathbf{x})$ that can possibly be *affected* by flipping a certain bit q.

3.2.2 Update Data Structures after a Single Bit Flip

We have introduced three data structures, namely vector $\mathbf{w}'(\mathbf{x})$, vector $\mathbf{S}(\mathbf{x})$ and list $\mathbf{Impr}(\mathbf{x})$. Our high-level guideline is to only compute the data structures once for some initial solution $\mathbf{x}_{init}$, then apply updates induced by a certain flipped bit q to generate new ones. We first study how to compute $\mathbf{w}'(\mathbf{x}^{(q)})$ on the basis of $\mathbf{w}'(\mathbf{x})$.

Lemma 4 (Section 3 in [WC12]). *Suppose* $\mathbf{i}, \mathbf{x} \in \mathcal{B}^n$. *After flipping a bit* q, *the vector* $\mathbf{w}'(\mathbf{x}^{(\mathbf{q})})$ *can be computed using*

$$\forall \mathbf{i} \in \mathcal{B}^n,\ w'_{\mathbf{i}}(\mathbf{x}^{(\mathbf{q})}) = \begin{cases} -w'_{\mathbf{i}}(\mathbf{x}) & \text{if } i_q = 1, \\ w'_{\mathbf{i}}(\mathbf{x}) & \text{if } i_q = 0. \end{cases}$$

Proof. Immediately by equation (2.8) and definition of $w_{\mathbf{i}}(\mathbf{x})$. □

Intuitively, any Walsh term that references the bit q changes its sign because flipping the q bit changes the parity of $\mathbf{i} \wedge \mathbf{x}$. We then focus on the updates on $S(\mathbf{x})$ after flipping the p^{th} bit. Lemma 5 gives the recursive form.

Lemma 5 (Section 3 in [WC12]). *Suppose* $\mathbf{i}, \mathbf{x} \in \mathcal{B}^n$. *After flipping a bit* q, *the vector* $\mathbf{S}(\mathbf{x}^{(\mathbf{q})})$ *can be computed using*

$$\forall p \in [1, n],\ S_p(\mathbf{x}^{(q)}) = S_p(\mathbf{x}) - 2 \sum_{\mathbf{i} \in \mathcal{B}^n \wedge i_p = 1 \wedge i_q = 1} w'_{\boldsymbol{i}}(\boldsymbol{x})$$

Proof. $\forall p \in [1, n]$, the difference between $S_p(\mathbf{x}^{(\mathbf{q})})$ and $S_p(\mathbf{x})$ can be calculated as follows:

$$\begin{aligned} S_p(\mathbf{x}^{(q)}) - S_p(\mathbf{x}) &= \sum_{\mathbf{i} \in \mathcal{B}^n \wedge i_p = 1} w_{\mathbf{i}} \psi_{\mathbf{i}}(\mathbf{x}^{(q)}) - \sum_{\mathbf{i} \in \mathcal{B}^n \wedge i_p = 1} w_{\mathbf{i}} \psi_{\mathbf{i}}(\mathbf{x}) \quad \text{By definition} \\ &= \sum_{\mathbf{i} \in \mathcal{B}^n \wedge i_p = 1} w_{\mathbf{i}} (\psi_{\mathbf{i}}(\mathbf{x}^{(q)}) - \psi_{\mathbf{i}}(\mathbf{x})) \\ &= -2 \sum_{\mathbf{i} \in \mathcal{B}^n \wedge i_p = 1 \wedge i_q = 1} w_{\mathbf{i}} \psi_{\mathbf{i}}(\mathbf{x}) \quad \text{By lemma 2} \\ &= -2 \sum_{\mathbf{i} \in \mathcal{B}^n \wedge i_p = 1 \wedge i_q = 1} w'_{\mathbf{i}}(\mathbf{x}) \quad \text{By definition} \end{aligned}$$

□

On a uniform random instance of NK-landscapes, the number of non-zero Walsh terms that are affected by flipping the q^{th} bit is constant given that c is constant. These Walsh terms can be described as the set in equation (3.4)

$$\{w'_{\mathbf{i}} | i_q = 1 \wedge w_{\mathbf{i}} \neq 0 \wedge \mathbf{i} \in \mathcal{B}^n\} \tag{3.4}$$

Lemma 6 describes how $\mathbf{Impr}(\mathbf{x}^{(q)})$ can be efficiently computed based on $\mathbf{Impr}(\mathbf{x})$ and $\mathbf{S}(\mathbf{x}^{(q)})$.

Lemma 6 (Section 3.2 in [WC12])**.** *Suppose* $\mathbf{i}, \mathbf{x} \in \mathcal{B}^n$*. After flipping a bit* q*, the list* $\mathbf{Impr}(\mathbf{x}^{(\mathbf{q})})$ *can be computed by the following steps.*

1. $\boldsymbol{Impr}(\boldsymbol{x}^{(q)}) \leftarrow \boldsymbol{Impr}(\boldsymbol{x})$.
2. *For all* $p \in \{j \in [1, n] | i_j = 1 \wedge i_q = 1 \wedge w_{\mathbf{i}} \neq 0 \wedge \mathbf{i} \in \mathcal{B}^n\}$*, repeat:*

$$\mathbf{Impr}(\mathbf{x}^{(q)}) \leftarrow \begin{cases} \mathbf{Impr}(\mathbf{x}^{(q)}) \cup \{p\} & \text{if} S_p(\mathbf{x}^{(q)}) \textit{ is an improving move} \wedge p \notin \mathbf{Impr}(\mathbf{x}), \\ \mathbf{Impr}(\mathbf{x}^{(q)}) \setminus \{p\} & \text{if} S_p(\mathbf{x}^{(q)}) \textit{ is a disimproving move} \wedge p \in \mathbf{Impr}(\mathbf{x}). \end{cases}$$

Proof. We first consider the positions that are *not affected* by flipping q, namely any position p where $p \notin \{j \in [1, n] | i_j = 1 \wedge i_q = 1 \wedge w_{\mathbf{i}} \neq 0 \wedge \mathbf{i} \in \mathcal{B}^n\}$. In this case, there is no non-zero Walsh term that references both positions p and q. Therefore, the change of parity induced by flipping the q^{th} bit has no way to propagate to position p. $S_p(\mathbf{x}^{(q)})$ stays the same as $S_p(\mathbf{x})$ for $p \notin \{j \in [1, n] | i_j = 1 \wedge i_q = 1 \wedge w_{\mathbf{i}} \neq 0 \wedge \mathbf{i} \in \mathcal{B}^n\}$. The initialization in Step 1 guarantees the correctness for this case.

We next consider the positions that are *affected* by flipping q, that is any position p where $p \in \{j \in [1,n] | i_j = 1 \wedge i_q = 1 \wedge w_{\mathbf{i}} \neq 0 \wedge \mathbf{i} \in \mathcal{B}^n\}$. Since there are Walsh coefficients that reference both position p and q, $S_p(\mathbf{x}^{(\mathbf{q})})$ can possibly differ from $S_p(\mathbf{x})$ by lemma 5. To reflect this difference, $S_p(\mathbf{x}^{(q)})$ needs to be checked for updating $\mathbf{Impr}(\mathbf{x})$ to generate $\mathbf{Impr}(\mathbf{x}^{(q)})$. Those affected positions are either appended to $\mathbf{Impr}(\mathbf{x}^{(\mathbf{q})})$ if they yield improving moves and were not in $\mathbf{Impr}(\mathbf{x})$, or removed from $\mathbf{Impr}(\mathbf{x}^{(\mathbf{q})})$ if they yield disimproving moves and were previously in $\mathbf{Impr}(\mathbf{x})$. □

3.3 Runtime Complexity Analysis for Walsh-LS

In this section, we justify our claims about the complexity per move of Walsh-LS. We start by analyzing the cost of initialization, then discuss the worst case scenario for taking one best-improvement move, and lastly prove the $O(1)$ time bound for the average case.

3.3.1 Initialization Costs

Operations in the initialization stage fall into two main categories: the Walsh transform and data structures initialization. We first recall Rana *et al.*'s analysis on the Walsh transform on k-bounded pseudo-Boolean functions (PBFs).

Theorem 1 ([RHW98])**.** *The Walsh transform on k-bounded pseudo-Boolean functions takes $O(m2^k)$, where m is the number of subfunctions.*

Proof. We can perform the Walsh transform on the basis of each subfunction of PBF, and then synthesize subfunction results for generating the Walsh decomposition for the original PBF. Since each subfunction involves at most k variables, the Walsh transform on a subfunction takes $O(2^k)$ time. Conducting this process on m subfunctions takes $O(m2^k)$ time in total. □

In the case of unrestricted NK-landscapes, the number of subfunctions m is the same as the number of variables n. Theorem 1 applies as well to NK-landscapes, thus the Walsh transform on NK-landscapes takes $O(n2^k)$. However, one could also directly generate NK-landscapes by using randomly generated numbers as Walsh coefficients, and no Walsh transform is needed.

We now concentrate on the complexity of initializing the data structures required in Walsh-LS. The major data structures in Walsh-LS are vector $\mathbf{w}'(\mathbf{x})$, vector $\mathbf{S}(\mathbf{x})$ and list $\mathbf{Impr}(\mathbf{x})$.

Lemma 7. *On a k-bounded pseudo Boolean function, initialization of the vector* $\mathbf{w}'(\mathbf{x})$ *in Walsh-LS for any given candidate solution* $x \in \mathcal{B}^n$ *takes* $O(mk2^k)$*, where* n *is the number of variables and* m *is the number of subfunctions.*

Proof. $\mathbf{w}'(\mathbf{x})$ can be calculated on a subfunction basis. Since there are k variables in a subfunction and 2^k Walsh terms associated with it, $\mathbf{w}'(\mathbf{x})$ for a subfunction can be computed in $O(k2^k)$ time by equation (2.8) and equation (2.9). For m subfunctions, the overall complexity is $O(mk2^k)$. □

Lemma 8. *On a k-bounded pseudo Boolean function, initialization of the vector* $\mathbf{S}(\mathbf{x})$ *in Walsh-LS for any given candidate solution* $x \in \mathcal{B}^n$ *takes* $O(mk2^{k-1})$*, where* m *is the number of subfunctions.*

Proof. In calculating the vector $\mathbf{S}(\mathbf{x})$, i.e., $S_p(\mathbf{x}) = \sum_{\mathbf{i} \in \mathcal{B}^n \wedge i_p = 1} w'_{\mathbf{i}}(\mathbf{x})$, Walsh terms with order higher than 1 can be reused. In fact, any Walsh term with order j will be used exactly j times in computing $\mathbf{S}(\mathbf{x})$. For the $O(2^k)$ non-zero Walsh terms induced by one subfunction, we calculate the cost related to these Walsh terms by counting how many times the Walsh terms induced by a single subfunction are reused, as shown in equation (3.5).

$$\begin{aligned} & \sum_{i=1}^{k} i\binom{k}{i} \\ &= k + \sum_{i=2}^{k} k\binom{k-1}{i-1} \\ &= k + k\sum_{i=1}^{k-1} \binom{k-1}{i} \\ &= k(\sum_{i=0}^{k-1} \binom{k-1}{i}) \\ &= k2^{k-1}. \end{aligned} \tag{3.5}$$

Summing up all m subfunctions, we get the bound $O(mk2^{k-1})$. □

Lemma 9. *On a k-bounded pseudo Boolean function, initialization of the list* $\mathbf{Impr}(\mathbf{x})$ *in Walsh-LS for any given candidate solution $x \in \mathcal{B}^n$ takes $O(n)$, where n is the number of variables.*

Proof. A complete scan over the vector $\mathbf{S}(\mathbf{x})$ is sufficient to construct **Impr**. **Impr** is initially $\emptyset$. For every variable (assume the p^{th}), check $S_p(\mathbf{x})$. If $S_p(\mathbf{x})$ yields an *improving* move, then p is appended to **Impr**. The complete scan over $\mathbf{S}(\mathbf{x})$ takes $O(n)$ time. □

Theorem 2. *The initialization stage of Walsh-LS runs in $O(mk2^k)$ on a k-bounded pseudo-Boolean function.*

Proof. This follows immediately from Lemmas 7, 8, 9 and Theorem 1, since constructing $\mathbf{w}'(\mathbf{x})$ forms the bottleneck and takes $O(mk2^k)$ time. □

We are going to investigate the cost of updating the data structures to reflect the impact of flipping a bit in the following subsection.

3.3.2 The $O(n)$ Worst Case Analysis

We address the situation where the bit to flip appears in $O(n)$ subfunctions, or in a more general sense, appears in αn subfunctions, where $\alpha \in (0, \frac{m}{n}]$. We will prove that flipping such bits are "expensive" moves to take. The costs for updates of $\mathbf{w}'(\mathbf{x})$, $\mathbf{S}(\mathbf{x})$ and $\mathbf{Impr}(\mathbf{x})$ are studied separately.

Lemma 10. *Assume the q^{th} variable appears in αn subfunctions and suppose $\mathbf{x} \in \mathcal{B}^n$ and $q \in [1, n]$, updating $\mathbf{w}'(\mathbf{x})$ in Walsh-LS after flipping a bit q takes $O(n)$ time on a k-bounded pseudo-Boolean function.*

Proof. According to lemma 4, the required update for $\mathbf{w}'(\mathbf{x})$ is just flipping the sign of all $w'_{\mathbf{i}}(\mathbf{x})$ terms that reference bit q. Since the bit q appears in αn subfunctions, these subfunction generates up to $\alpha 2^k n$ Walsh terms. Half of the $\alpha 2^k n$ Walsh terms contain bit q. Therefore, there are at most $\alpha 2^{k-1} n$ number of $w'_{\mathbf{i}}(\mathbf{x})$ terms that reference bit q. The update cost for $\mathbf{w}'(\mathbf{x})$ $O(\alpha 2^{k-1} n) = O(n)$. □

Lemma 11. *Assume the q^{th} variable appears in αn subfunctions and suppose $\mathbf{x} \in \mathcal{B}^n$ and $p \in [1, n]$, updating $\mathbf{S}(\mathbf{x})$ in Walsh-LS after flipping a bit q takes $\alpha n(k-1)2^{k-2}$ time on a k-bounded pseudo-Boolean function.*

Proof. For a subfunction where the q^{th} variable appears, it contains the q^{th} variable and $k-1$ other variables. We consider how the Walsh terms induced by this subfunction are used in the formula of lemma 5. The $k-1$ other variables can generate 2^{k-1} non-zero Walsh terms. We make the q^{th} a mandatory variable included in these Walsh terms, each Walsh term decreases its order by 1, which does not change the amount of non-zero Walsh terms. Because of the restriction $i_q = 1$ in the formula of lemma 5, this is the same situation that we faced in the proof of lemma 8, except that each subfunction now only contains $k-1$ free variables (the q^{th} mandatory variable is omitted). Equation (3.5) applies here as well, we then get the bound $(k-1)2^{k-2}$ for one subfunction. Since there are αn such subfunctions, the update cost is $\alpha n(k-1)2^{k-2}$. □

Lemma 12. *Assume the q^{th} variable appears in αn subfunctions and suppose $\mathbf{x} \in \mathcal{B}^n$ and $p \in [1, n]$, updating $\mathbf{Impr}(\mathbf{x})$ in Walsh-LS after a bit q takes $O(n)$ time on a k-bounded pseudo-Boolean function.*

Proof. For any given subfunction that includes the bit q, there are $k-1$ other variables that interact with the bit q. According to lemma 5, each of these $(k-1)$ variables generates non-zero Walsh terms that induce updates to a single element in the vector $\mathbf{S}(\mathbf{x})$. Since the q^{th} variable appears in αn subfunctions, there can be at most $\alpha n(k-1)$ elements in $\mathbf{S}(\mathbf{x})$ changes. We conclude that at most $\alpha n(k-1)$ variables need to be examined for eligibility in $\mathbf{Impr}(\mathbf{x}^{(q)})$. □

Theorem 3. *In Walsh-LS, the update cost associated with flipping a bit that appears in αn subfunctions is $O(n)$ on k-bounded pseudo-Boolean functions.*

Proof. Based on the results by lemma 10, lemma 11 and lemma 12, it is immediately obvious because the operations that update $\mathbf{S}(\mathbf{x})$ after a single bit flip forms the bottleneck and that takes $\alpha n(k-1)2^{k-2}$. □

Another concern about the runtime of Walsh-LS is deciding improving moves from **Impr**. If the length of the list **Impr** is $O(n)$, selecting the best-improvement move can take $O(n)$ time. Since the update cost of $\alpha n(k-1)2^{k-2}$ in the worst case is even more expensive than this, we postpone our solution until next subsection for the $O(1)$ average case analysis.

3.3.3 The $O(1)$ Average Case Analysis

We showed that Walsh-LS can take up to $O(n)$ time for selecting one best-improvement move in the worst case. The worst-case scenario happens when the flipped bit appears in $O(n)$ subfunctions. We argue, however, the worst-case scenario is unlikely to happen and Walsh-LS can achieve $O(1)$ runtime on uniform random instances in expectation. Assuming a uniform random instance, we start justifying our argument from the fact that the probability of any variable appearing in $O(n)$ subfunctions is low, then stress that the expected number of subfunctions in which any given variable can appear is a constant with respect to n.

Lemma 13. *Suppose $\beta \in \mathcal{R} \wedge \beta \in (0, \frac{m}{n}]$, a uniform random instance of k-bounded pseudo-Boolean functions, the probability of at least one variables appearing in at least βn (as a representative of function class $O(n)$) subfunctions is $\sum_{i=\beta n}^{m} \binom{n}{i} (\frac{k}{n})^i (1-\frac{k}{n})^{(m-i)}$.*

Proof. For a given variable p, consider its appearing in one subfunction as an event A. The probability of p appearing in one subfunction (denoted as $P(A)$) is $\frac{k}{n}$. Now that there are m subfunctions, the experiment is repeated m times. Generally, the probability of A happening exactly i times ($0 \leq i \leq m$) among m independent experiments is $\binom{n}{i} P(A)^i (1-P(A))^{(m-i)}$. Applying to our problem, the probability of p appearing in at least βn subfunctions is $\sum_{i=\beta n}^{m} \binom{n}{i} (\frac{k}{n})^i (1-\frac{k}{n})^{(m-i)}$. □

For $n = 100, m = 100, k = 4$ and $\beta = 0.1$,

$$\sum_{i=\beta n}^{m} \binom{n}{i} (\frac{k}{n})^i (1-\frac{k}{n})^{(m-i)} = \sum_{i=0.1*100}^{100} \binom{100}{i} (\frac{4}{100})^i (1-\frac{4}{100})^{(100-i)} = 0.0068.$$

This indicates that it is already very unlikely for a variable to appear in more than 20% of all subfunctions on uniform random instances of k-bounded pseudo-Boolean functions, especially with large n and small k, where our main focus is.

Lemma 14. *On a uniform random instance of k-bounded pseudo-Boolean functions, the expected number of subfunctions in which any given variable appears is ck.*

Proof. We shall view the number of subfunctions that any given variable appears in as a random variable $\mathcal{V}$. $\mathcal{V}$ follows the binomial distribution $B(m, \frac{k}{n})$. The expectation of the random variable $\mathbb{E}(\mathcal{V})$ is $m * \frac{k}{n}$. Given $m = c * n$, the expected number of subfunctions in which any given variable appears is ck. □

Lemma 14 shows the expected number of subfunctions in which any variable appears is bounded by k. It does not indicate, however, all variables on a uniform random instance appear in exactly ck subfunctions. We need to consider the variance as well. As explained in the proof of Lemma 14, $\mathcal{V}$ follows the binomial distribution $B(m, \frac{k}{n})$. The variance of $\mathcal{V}$ is $m * \frac{k}{n} * (1 - \frac{k}{n}) = ck(1 - \frac{k}{n})$. Given $k \ll n$, $ck(1 - \frac{k}{n}) \approx ck$. The variance of $\mathcal{V}$ is also bounded by k. Moreover, we learn from lemma 10, lemma 11 and lemma 12 that the complexity for updating $\mathbf{w}'(\mathbf{x})$, $\mathbf{S}(\mathbf{x})$ and $\mathbf{Impr}(\mathbf{x})$ is linear in $\mathcal{V}$. We can therefore consider the expectation of $\mathcal{V}$ as an average-case scenario without changing the asymptotic bound.

According to the arguments in lemmas 13 and 14, the worst-case scenarios discussed in subsection 3.3.2 are unlikely to happen for uniform random variable placement. We shall now explore how to achieve the $O(1)$ average cost.

Lemma 15. *Assume the q^{th} variable appears in ck subfunctions and suppose $\mathbf{x} \in \mathcal{B}^n$ and $p \in [1, n]$, updating $\mathbf{w}'(\mathbf{x})$ after a bit q takes $O(1)$ time on a k-bounded pseudo-Boolean function.*

Proof. Most of the proof of lemma 11 still holds here. The only difference is that there are ck subfunctions that include the bit q, thus the cost for updating $\mathbf{w}'(\mathbf{x})$ is $ck2^{k-1}$. □

Lemma 16. *Assume the q^{th} variable appears in ck subfunctions and suppose $\mathbf{x} \in \mathcal{B}^n$ and $p \in [1, n]$, updating $\mathbf{S}(\mathbf{x})$ after a bit q takes $ck(k-1)2^{k-2}$ time on a k-bounded pseudo-Boolean function.*

Proof. Most of the proof of lemma 11 still holds here. The only difference is that there are ck subfunctions that include the bit q, thus the cost for updating $\mathbf{S}(\mathbf{x})$ is $ck(k-1)2^{k-2}$. □

Lemma 17. *Assume the q^{th} variable appears in ck subfunctions and suppose $\mathbf{x} \in \mathcal{B}^n$ and $p \in [1, n]$, updating* $\mathbf{Impr}(\mathbf{x})$ *after a bit q takes $O(1)$ time on a k-bounded pseudo-Boolean function.*

Proof. Most of the proof of lemma 12 still holds here. The only difference is that there are ck subfunctions that include the bit q, thus the cost for updating $\mathbf{Impr}(\mathbf{x})$ is $ck(k-1)$. □

Theorem 4. *In Walsh-LS, the update cost associated with flipping a bit that appears in ck subfunctions is $O(1)$ in expectation on uniform random k-bounded pseudo-Boolean functions.*

Proof. Based on Lemma 15, Lemma 16 and Lemma 17, it is obvious because the operations that update $\mathbf{S}(\mathbf{x})$ after a single bit flip forms the bottleneck and that takes $ck(k-1)2^{k-2}$. □

3.3.3.1 Approximation in Move Selection

We are now able to achieve the $O(1)$ average cost on *updating data structures*. There is only one issue left: the length of list **Impr**. Determining the best-improvement move from *Impr* of length $O(n)$ can take $O(n)$ time, since a complete scan over *Impr* is required. We overcome this by approximating the best-improvement move. Let $\theta \in [1, n]$ and $\mathbf{Impr}.len$ be the length of the list **Impr**. The constant θ serves as a threshold. The approximate best-improvement move is generated by algorithm 5.

Algorithm 5: BestI ← ApproxBestImprMove(**Impr**)

```
1 if Impr.len > θ then                                  // take the approximate move
2     approxImpr ← sample(Impr) ;                       // sample θ elements from Impr
      uniformly at random
3     BestI ← SelectBest(approxImpr);
4 else                                                  // take the standard best-improvement move
5     BestI ← SelectBest(Impr);
```

Theorem 5. *The complexity per move for Walsh-LS with the approximation (called approx-Walsh-LS) described in algorithm 5 is $O(1)$ on average.*

Proof. Since θ is a constant and determining the approximate best-improvement move requires examining no more than θ candidate improving moves, algorithm 5 returns the index of the

bit to flip $BestI$ in $O(1)$ time. Combined with Theorem 4, we conclude that approx-Walsh-LS requires $O(1)$ time for taking one approximate best-improvement move. □

Whitley and Chen [WC12] accomplish the $O(1)$ complexity per move result in a slightly different approach. Besides the approximation in selecting best moves, they adopt a Tabu mechanism that prevents high-cost bits (that appear in many subfunctions) from being flipped too many times. They can then prove an amortized $O(1)$ complexity per move *without* assuming uniformity of the variable distribution over subfunctions.

3.4 Empirical Studies

In order to match the solution obtained in standard BILS, a complete scan over **Impr** is implemented in our empirical studies on Walsh-LS. To avoid confusion, we denote Walsh-LS without approximation in selecting best improvement moves as *exact-Walsh-LS*.

3.4.1 Can exact-Walsh-LS run faster than PE-BILS?

Three major concerns on the performance of exact-Walsh-LS when compared with PE-BILS on uniform random instances are: 1) The $O(n)$ initialization cost needs to be charged for every restart. 2) **Impr**.*len* can be $O(n)$, which leads to the $O(n)$ complexity of scanning over the whole list. 3) The setting of k will affect the runtime. These bring up the first question: can exact-Walsh-LS run faster than BILS in practice? The question is addressed by posing the following hypotheses.

When applying SLS to solve binary-encoded combinatorial problems, the number of maximum bit-flips (*MaxBitFlip*) is typically predefined to terminate the execution after some computational time. In the widely used UBCSAT library[5] for solving SAT problems using SLS [TH04], two parameters are defined for terminating SLS: the number of trials *MaxTries* and the number of bit-flips within one trial *MaxBitFlipPerTrial*. Each trial consists

[5]UBCSAT :: A Stochastic Local Search SAT Solver Framework. http://ubcsat.dtompkins.com/

of *MaxBitFlipPerTrial* number of bit-flips, and between any two consecutive trials, a random restart is enforced to escape possible deep local optima. By default, MaxBitFlipPerTrial is set to 10000 and MaxTries is set to 10. Overall, the maximum number of bit-flips allowed when evaluating a SLS is $10000 * 10 = 100000$. We claim that the $O(n)$ one-time initialization cost in theorem 2 should comprise only a small portion of overall runtime, and can be amortized over this amount of bit-flips. Regarding the $O(n)$ scanning cost during the update stage, we argue that most of the time $\mathbf{Impr}.len$ is *small*. The hidden constant in the $O(n)$ complexity is *small*. To sum up, we expect exact-Walsh-LS to have less runtime than PE-BILS.

Hypothesis 1. *On uniform random NK-landscape instances with small K (less than 5), not only does exact-Walsh-LS make best-improvement moves (only considering update operations) more efficiently than PE-BILS, but also the overall exact-Walsh-LS (including the initialization stage) runs more efficiently than PE-BILS.*

All algorithms[6] are implemented in Python 2.7. The computational platform was Fedora 16 using Xeon5450 at 3.00GHz with 16Gb main memory.

Experiment 1. We run both exact-Walsh-LS and PE-BILS on uniform random NK-landscapes instances[7] with $n = \{20, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500\}$ and $K = \{2, 4\}$. 100000 bit-flips (denoted as *MaxBitFlip*) are allowed for each algorithm on every instance. To make the results statistically sound, the trials for each configuration are repeated 10 times independently. We record best-so-far solutions, overall runtime, and time spent on performing updates, for both exact-Walsh-LS and PE-BILS.

Table 3.1 compares the overall runtime of exact-Walsh-LS and PE-BILS, as well as the speedups of exact-Walsh-LS versus PE-BILS as n increases ($Speedup = \frac{T_{BILS}}{T_{exact-Walsh-LS}}$) on

[6]Our source code can be checked out at https://github.com/qzcwx/WalLS.

[7]Our benchmark problem instances as well as the problem generators written in Python are publicly available at http://bitbucket.org/qzcwx/nk-q-instances.

uniform random instances, while table 3.2 only considers the time spent on performing updates. We observe from table 3.1 that Walsh-LS does run faster than PE-BILS in practice. In fact, the speedups of Walsh-LS versus PE-BILS range from 17.5× to 110×, and the speedups grow as n increases. As shown in table 3.2, without taking the $O(n)$ initialization stage of Walsh-LS into account, the speedups are even more pronounced. It is not surprising though because PE-BILS does not have any initialization cost, and thus the time spent on update is nearly identical to the overall runtime for PE-BILS. The range of speedups is now from 85.6× to 449×. Hypothesis 1 is validated.

Table 3.1: Overall runtime in seconds (including the initialization step) on *uniform* instances. The median runtimes over 10 runs is presented, and the numbers colored in grey after the "±" symbol are the corresponding standard deviations. Under the "Speedup" column, the speedups of exact-Walsh-LS over PE-BILS are shown.

k	n	exact-Walsh-LS	PE-BILS	Speedup
2	20	1.19 ± 0.01	29.50 ± 0.16	24.69 ± 0.35
	50	1.52 ± 0.01	67.10 ± 0.34	44.07 ± 0.20
	100	2.12 ± 0.00	135.00 ± 0.88	63.53 ± 0.35
	150	2.71 ± 0.01	203.50 ± 1.79	75.19 ± 0.58
	200	3.27 ± 0.01	284.00 ± 2.88	86.59 ± 0.91
	250	3.98 ± 0.01	349.50 ± 2.88	88.16 ± 0.72
	300	4.54 ± 0.01	428.50 ± 2.41	94.28 ± 0.40
	350	5.12 ± 0.02	507.50 ± 1.78	99.02 ± 0.40
	400	5.80 ± 0.02	587.50 ± 5.10	101.21 ± 0.97
	450	6.31 ± 0.02	660.50 ± 3.18	104.69 ± 0.50
	500	6.99 ± 0.02	768.00 ± 4.00	109.95 ± 0.66
4	20	3.08 ± 0.02	53.80 ± 0.20	17.48 ± 0.15
	50	4.40 ± 0.04	127.00 ± 0.32	28.86 ± 0.32
	100	6.39 ± 0.05	243.00 ± 0.52	38.08 ± 0.29
	150	8.30 ± 0.07	362.50 ± 4.85	43.60 ± 0.94
	200	10.00 ± 0.13	490.00 ± 6.65	48.75 ± 1.21
	250	12.10 ± 0.74	637.00 ± 7.02	52.27 ± 3.10
	300	14.20 ± 0.29	735.50 ± 5.90	51.65 ± 1.46
	350	16.70 ± 0.51	869.00 ± 8.38	51.95 ± 2.28
	400	19.20 ± 0.85	997.00 ± 5.13	51.90 ± 2.74
	450	21.25 ± 0.70	1140.00 ± 18.14	53.41 ± 2.85
	500	23.10 ± 0.63	1250.00 ± 14.49	54.00 ± 2.24

Table 3.2: Best-improve move time in seconds (only considering update operations) on *uniform* instances. The median runtimes over 10 runs are presented, and the number colored in grey after the "±" symbol is the corresponding standard deviations. Under the "Speedup" column, the speedups of Walsh-LS over PE-BILS are shown.

k	n	exact-Walsh-LS	PE-BILS	Speedup
2	20	0.34 ± 0.00	29.10 ± 0.15	85.61 ± 1.46
	50	0.44 ± 0.00	66.75 ± 0.33	150.62 ± 1.01
	100	0.56 ± 0.00	134.50 ± 0.92	239.75 ± 1.78
	150	0.72 ± 0.00	203.50 ± 1.55	281.27 ± 2.01
	200	0.85 ± 0.00	283.50 ± 2.95	331.39 ± 3.66
	250	1.02 ± 0.00	349.50 ± 2.88	342.65 ± 3.39
	300	1.15 ± 0.00	428.00 ± 2.22	372.17 ± 1.93
	350	1.30 ± 0.01	507.50 ± 1.65	389.66 ± 2.29
	400	1.43 ± 0.01	587.50 ± 5.06	410.53 ± 4.10
	450	1.55 ± 0.01	660.00 ± 3.17	427.18 ± 2.22
	500	1.71 ± 0.00	768.00 ± 3.89	449.12 ± 2.42
4	20	0.91 ± 0.01	53.15 ± 0.20	58.40 ± 0.70
	50	1.22 ± 0.02	126.00 ± 0.48	103.28 ± 1.22
	100	1.63 ± 0.02	243.00 ± 0.67	148.77 ± 1.65
	150	2.02 ± 0.02	362.00 ± 4.54	178.77 ± 4.10
	200	2.37 ± 0.04	489.00 ± 6.34	206.54 ± 5.99
	250	2.84 ± 0.23	636.00 ± 7.08	221.97 ± 17.28
	300	3.34 ± 0.09	735.50 ± 6.00	219.28 ± 7.90
	350	3.90 ± 0.16	868.00 ± 8.21	222.54 ± 12.78
	400	4.45 ± 0.27	997.00 ± 5.20	224.30 ± 16.35
	450	4.88 ± 0.23	1135.00 ± 18.53	233.06 ± 16.67
	500	5.39 ± 0.21	1250.00 ± 14.91	230.84 ± 13.20

In order to investigate more on the growth of speedup, we plot the speedup versus n in figure 3.1. We find the speedup grows in a sublinear manner as n increases, which is expected because there is no difference in exact-Walsh-LS and PE-BILS from runtime complexity point of view. To better understand the speedup plots, we need to go back to the growth of runtime for individual algorithms. We plot $T_{exact-Walsh-LS}$ and $T_{PE-BILS}$ versus n separately in figure 3.2. Nonlinear regression is used to fit the sample points using R[8]. This enables us to quantitatively analyze the growth of runtime. We can see from figure 3.2 that the

[8]The R Project for Statistical Computing. http://www.r-project.org/

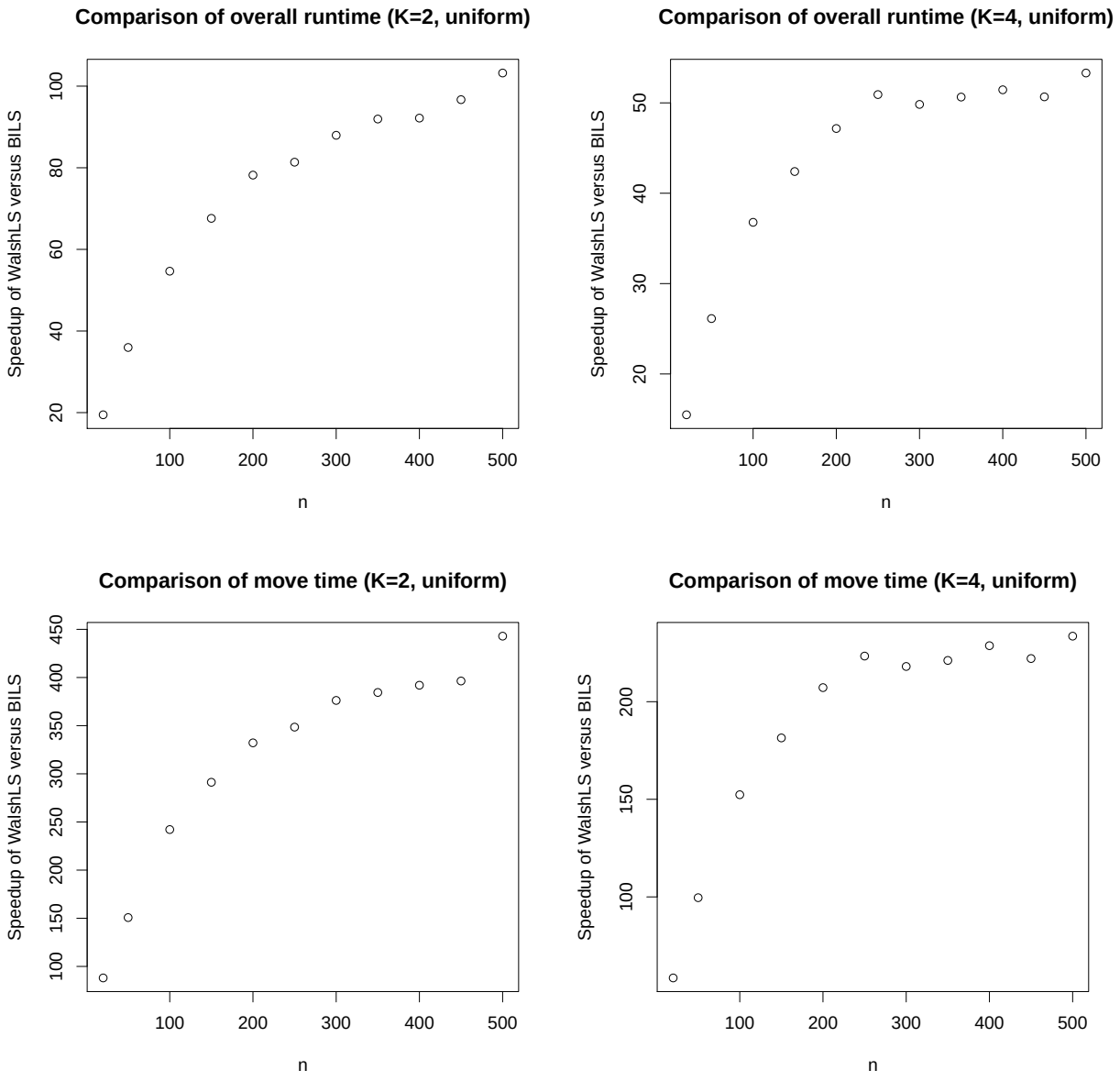

Figure 3.1: Speedup on *uniform* instances of NK-landscapes.

difference in runtime (which leads to substantial speedup) is mainly captured by the constant b in the nonlinear model $F = a + b * N^c$. [9] The parameter b for exact-Walsh-LS is always much smaller than that for PE-BILS.

3.4.2 Will "expensive" bits break the efficiency of exact-Walsh-LS?

Exact-Walsh-LS runs far more efficiently than PE-BILS on uniform random instances without "expensive". Will "expensive" bits break the efficiency of exact-Walsh-LS? As a rule of thumb, we call a bit "expensive" if it appears in more than $\frac{1}{10}n$ subfunctions. Otherwise, it is called a "cheap" bit.

We first investigate how to create instances that contain such "expensive" bits. To create non-uniform random NK-landscape instances, we employ the binomial distribution instead of a uniform random distribution. The generator for non-uniform random instances is presented in algorithm 6. BINOMIAL function returns an integer sampled from a binomial distribution $B(n, 0.5)$, and the integer serves as the index for selecting an element from permutation Π.

Algorithm 6: Non-uniform random NK-landscapes generator

```
1 Π ← PERM({1,...,n}) ;          // Π is a permutation of sequence {1,...,n}
2 for i ← 1 to N do
3     I_i ← {i};                 // I_i is the set of indices of variables appear in f_i
4     for j ← 1 to K − 1 do
5         cand ← Π[BINOMIAL(N, 0.5)];
6         while cand ∈ I_i do
7             cand ← Π[BINOMIAL(N, 0.5)];
8         I_i ← I_i ∪ {cand}
```

"Expensive" bits could possibly break the efficiency of exact-Walsh-LS only if they are frequently flipped. Based on the knowledge of the generator for non-uniform instances that

[9] The constant term a typically captures the initialization time. However, the initialization time in exact-Walsh-LS is $O(n)$. Therefore, the factor of linear term, b, captures both initialization time and update time.

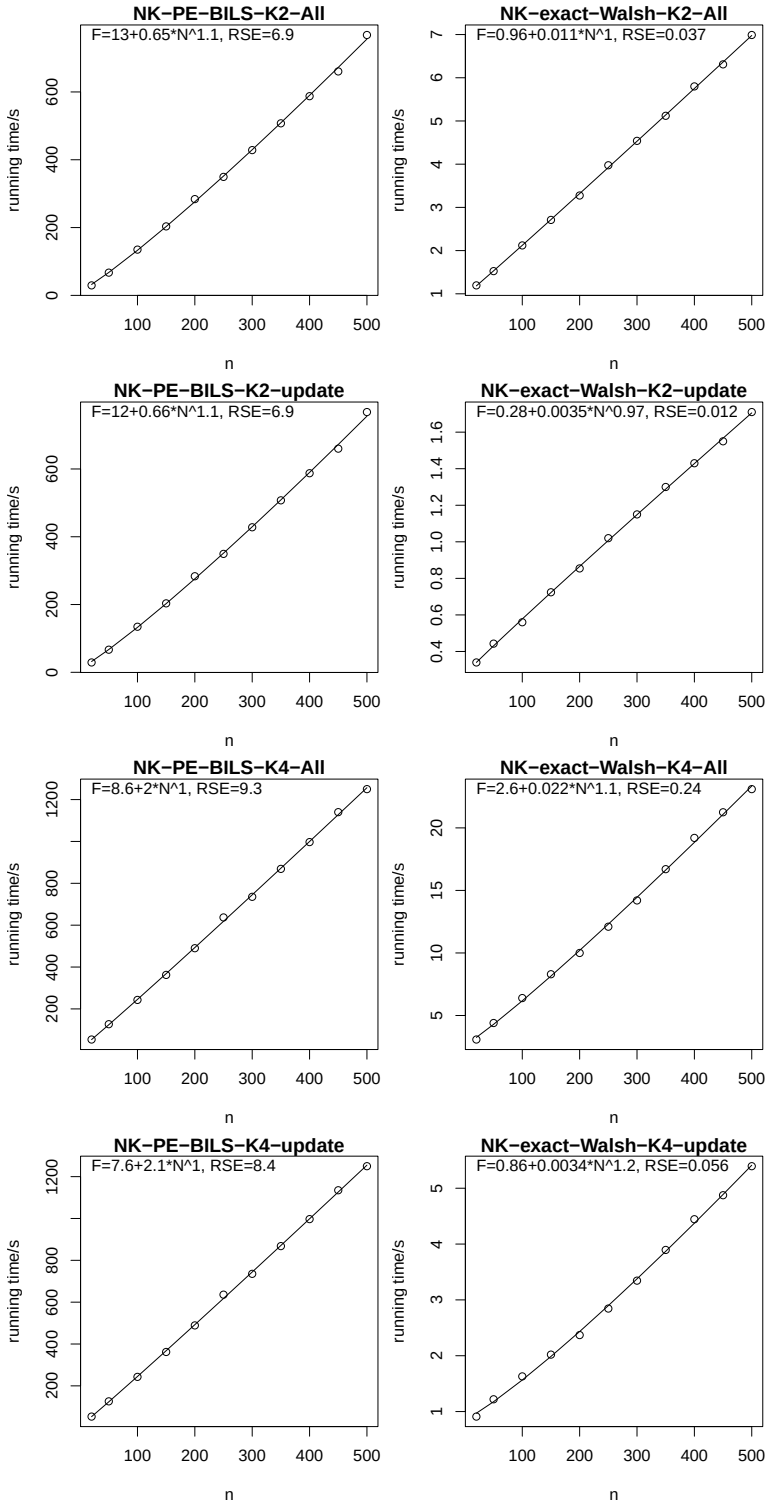

Figure 3.2: Runtime in seconds on *uniform* instances. The fitted curve is generated using the nonlinear regression "nls" from R with the formula $F = a + b * N^c$. The residual standard error ("RSE") is reported to indicate the goodness of fitting.

variables are uniformly distributed and have no interaction, we conjecture that variables that appear in many subfunctions typically have a greater impact on the overall evaluation function. Once these variables are flipped, then they rarely change again. On the contrary, "cheap" bits that appear in a small number of subfunctions tend to be flipped many times.

Hypothesis 2. *On non-uniform random NK-landscapes instances, exact-Walsh-LS tends to flip "expensive" bits less times than "cheap" bits.*

Experiment 2. We run exact-Walsh-LS on both uniform and non-uniform random NK-landscapes instances with $n = \{20, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500\}$ and $K = \{2, 4\}$. MaxBitFlip is set to 100000 for each algorithm on every instance. In order to make the results statistically sound, the trials for each configuration are repeated 10 times independently. The mean bit-flip counts for each variable over 10 runs (denoted as *FlipCount*) are stored. A script parses input instances to count the number of occurrences of variables (denoted as *NumOccur*) in subfunctions.

We pick three representative values 100, 200 and 500 for n, and plot FlipCount versus NumOccur for uniform random instances in figure 3.3, and for non-uniform random instances in figure 3.4. The trends shown in the figures are consistent across tested instances with all n. We report the data for uniform random instances mostly for reference purposes, since in expectation there will be few "expensive" bits. The vertical dashed lines are drawn for distinguishing "expensive" bits from "cheap" ones. Points on the right side of dashed lines correspond to "expensive" bits. We also report the expected bit-flip counts if all bits are flipped uniformly at random (i.e., $\frac{MaxBitFlip}{n}$) as an horizontal dotted lines. Preferably, all points on the right side of dashed lines will sit below dotted lines.

From figure 3.3, we observe that there are indeed very few "expensive" bits except when $n = 100$ and $K = 4$. That is an artifact of the way we define "expensive" bit ($0.1n$=10) and the impact of K on the expected number of occurrences ($ck = 4$). Nevertheless, "expensive" bits are generally flipped less than "cheap" bits.

We then examine figure 3.4. Clearly, the distribution of NumOccur is much less uniform. In the case where $n = 500$ and $K = 4$, NumOccur for random instance ranges from 1 to

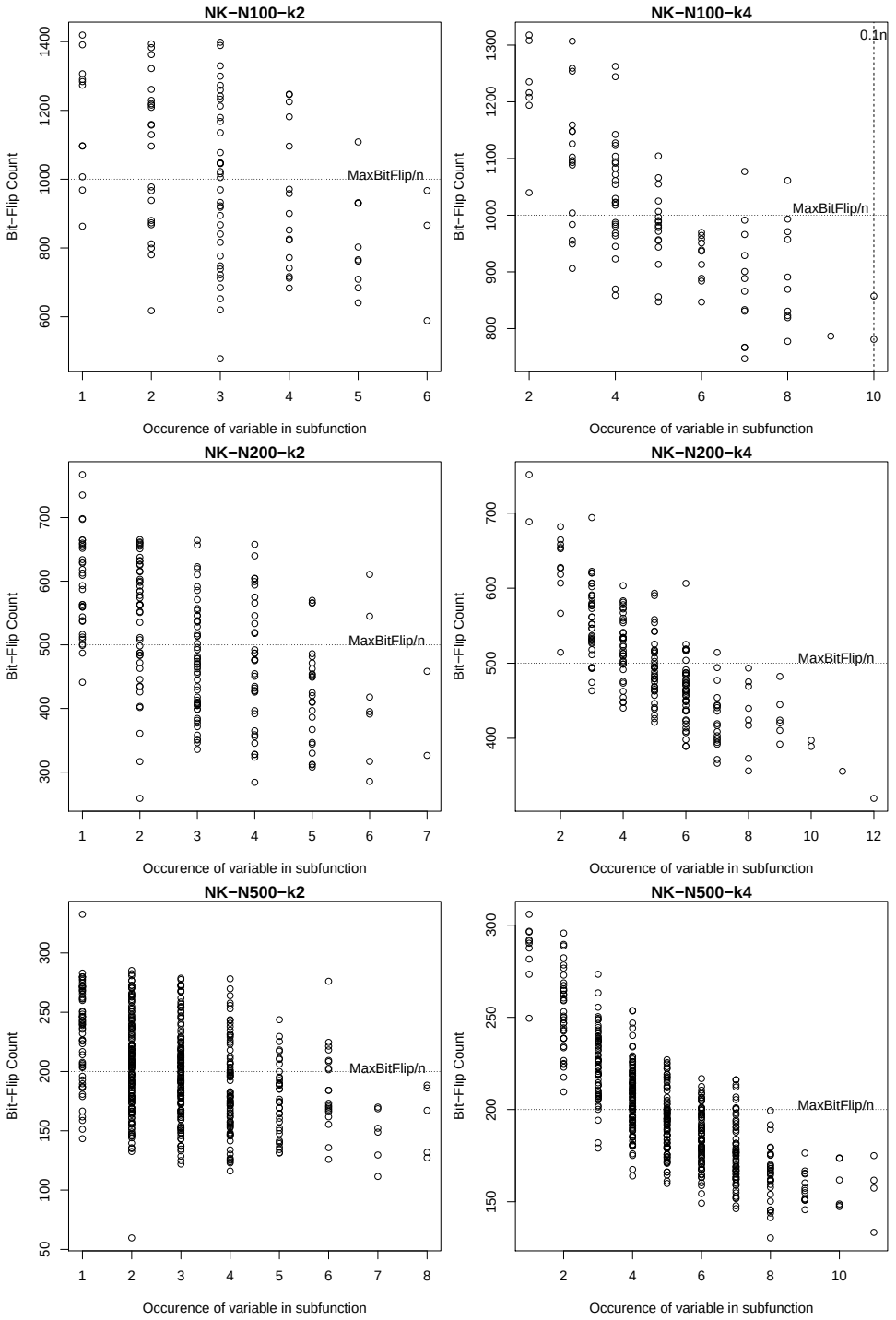

Figure 3.3: Correlation between frequency of a variable appearing in subfunctions and the number of times being flipped on *uniform* instances.

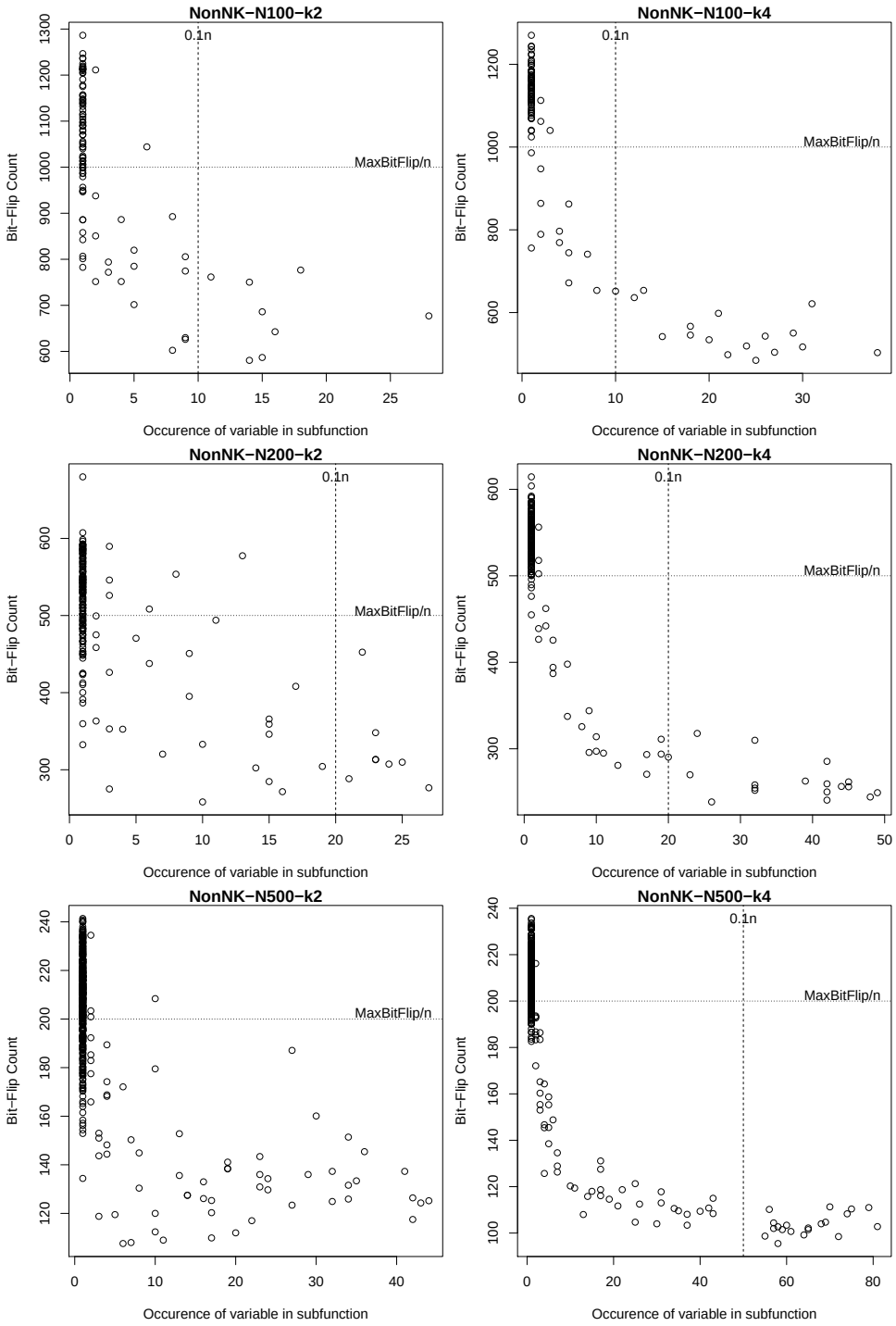

Figure 3.4: Correlation between frequency of a variable appearing in subfunctions and the number of times being flipped on *non-uniform* instances.

11, while that range for non-uniform is from 1 to 81. This is what we would expect from non-uniform NK-landscape instances. On the other hand, "expensive" bits are apparently flipped much less than "cheap" ones. In fact, all points corresponding to "expensive" bits are below the dashed lines. The trend is more remarkable for $K = 4$. One can think of the plot for $K = 4$ as stretched along the x-axis. In some sense, exact-Walsh-LS is more well-behaved on instances with larger K. We then verify hypothesis 2 empirically. We shall now be confident that "expensive" bits will not blow up the efficiency of exact-Walsh-LS. Even better, since exact-Walsh-LS keeps flipping "cheap" bits, exact-Walsh-LS might run faster on non-uniform instances with the same settings of n and K.

Hypothesis 3. *On non-uniform random NK-landscape instances, exact-Walsh-LS runs at least at a comparable speed (if not faster) than it does on uniform random instances with the same settings of n and K.*

Experiment 3. We run exact-Walsh-LS on both uniform random and non-uniform random NK-landscapes instances with $n = \{20, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500\}$ and $K = \{2, 4\}$. MaxBitFlip is set to 100000. The trials for each configuration are repeated 10 times independently. We record overall runtime for exact-Walsh-LS on both uniform random and non-uniform random instances.

We compare the overall runtime of exact-Walsh-LS on uniform random and non-uniform random instances in table 3.3. Surprisingly, exact-Walsh-LS consistently runs faster on non-uniform random instances than on uniform random instances with the same settings of n and K.

We are able to view the trend better in figure 3.5. Circles represent the runtime of exact-Walsh-LS on uniform random instances, while triangles stand for that on non-uniform random instances. The gap between circles and triangles becomes more pronounced when K gets larger. This matches the bit-flip pattern of exact-Walsh-LS we uncovered before. As shown in the empirical studies above, exact-Walsh-LS has a clear superiority over PE-BILS in runtime, while the solutions obtained from exact-Walsh-LS match exactly with PE-BILS.

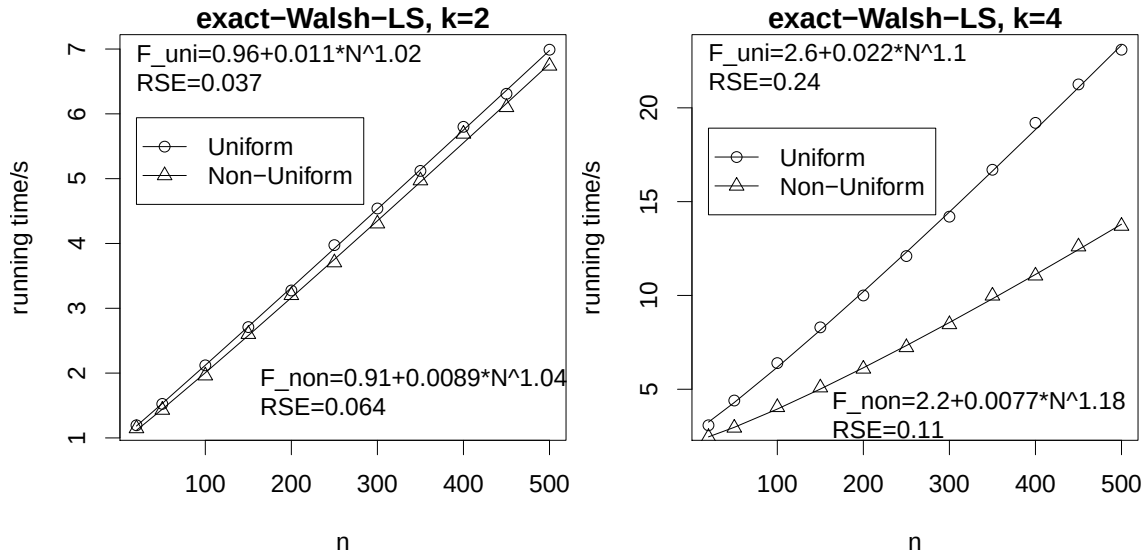

Figure 3.5: Overall runtime in seconds of exact-Walsh-LS on both uniform random and non-uniform random NK-landscape instances. The fitted curve is generated using the nonlinear regression "nls" from R with the formula $F = a + b * N^c$. The residual standard error ("RSE") is reported to indicate the goodness of fit.

It is important because we are now able to execute standard BILS far more efficiently via Walsh analysis. However, without approximation in selecting the best moves, the complexity per move is still $O(n)$. We will overcome this by parameterizing Walsh-LS.

Table 3.3: Overall runtime in seconds for uniform random and non-uniform random instances. The median runtimes over 10 runs are presented, and the number colored in grey after the "±" symbol is the corresponding standard deviations.

k	n	Uniform	Non-Uniform
2	20	1.19 ± 0.01	1.15 ± 0.01
	50	1.52 ± 0.01	1.43 ± 0.01
	100	2.12 ± 0.00	1.96 ± 0.01
	150	2.71 ± 0.01	2.60 ± 0.01
	200	3.27 ± 0.01	3.20 ± 0.02
	250	3.98 ± 0.01	3.71 ± 0.01
	300	4.54 ± 0.01	4.30 ± 0.02
	350	5.12 ± 0.02	4.97 ± 0.02
	400	5.80 ± 0.02	5.69 ± 0.03
	450	6.31 ± 0.02	6.10 ± 0.02
	500	6.99 ± 0.02	6.74 ± 0.02
4	20	3.08 ± 0.02	2.44 ± 0.02
	50	4.40 ± 0.04	2.94 ± 0.02
	100	6.39 ± 0.05	4.04 ± 0.03
	150	8.30 ± 0.07	5.08 ± 0.03
	200	10.00 ± 0.13	6.08 ± 0.06
	250	12.10 ± 0.74	7.23 ± 0.12
	300	14.20 ± 0.29	8.46 ± 0.16
	350	16.70 ± 0.51	9.98 ± 0.32
	400	19.20 ± 0.85	11.05 ± 0.35
	450	21.25 ± 0.70	12.60 ± 0.32
	500	23.10 ± 0.63	13.70 ± 0.42

Chapter 4

Parameterization of Constant Time Best-Improvement Local Search

In the previous chapter, we stated that the $O(n)$ complexity per move of exact-Walsh-LS on uniform random instances is a result of two conditions: 1) **Impr**.*len* is $O(n)$, and 2) determining the true best-improvement move requires a complete scan over **Impr**, which takes $O(n)$. In this chapter, we aim at overcoming the $O(n)$ complexity per move by breaking one of the two conditions.

The evaluation values of all subfunctions in the uniform random k-bounded pseudo-Boolean function considered in this section are independent and identically distributed random variables drawn from uniform distribution $U(0, 1)$.

4.1 Random Restart vs. Random Walk

4.1.1 Reducing Impr.len from $O(n)$ to $O(1)$

We conjecture that the hard random restart when Walsh-LS hits a local optimum is responsible for the $O(n)$-length of **Impr**. For a randomly initialized solution, we claim that **Impr**.*len* is $O(n)$ in expectation. On a uniform random k-bounded pseudo-Boolean function, only a constant number of bits are expected to be appended to or removed from **Impr** in each move. Since Walsh-LS keeps taking improving moves, we expect the bits are removed from **Impr** in most cases. In other words, **Impr**.*len* tends to decrease as the search progresses. When **Impr**.*len* goes down to zero, a hard random restart is triggered, which brings **Impr**.*len* back to $O(n)$ again. Amortized over a period of time, the **Impr**.*len* is $O(n)$.

Lemma 18. *On a uniform random k-bounded pseudo-Boolean function f in which the evaluation values of all subfunctions are independent and identically distributed random*

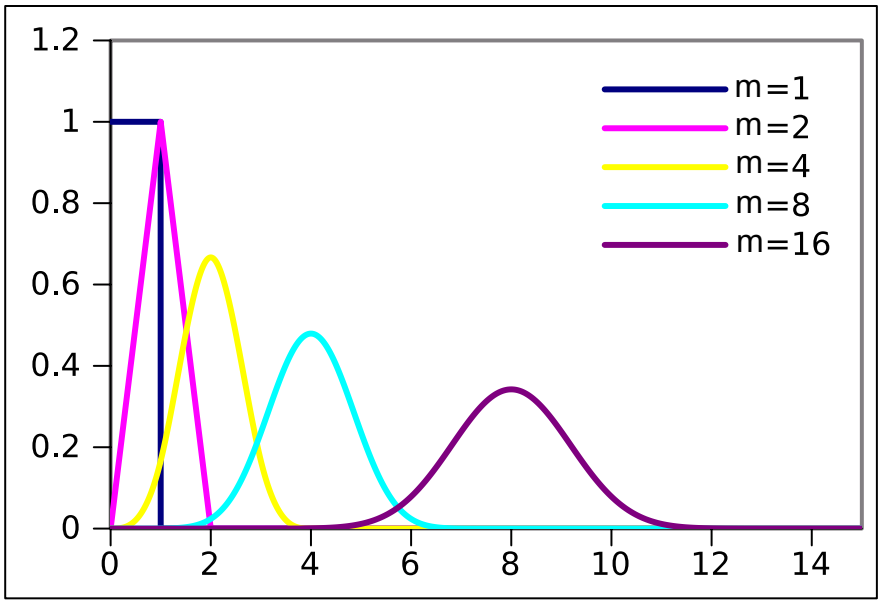

Figure 4.1: Probability density function of Irwin-Hall distribution.[10]

variables drawn from uniform distribution $U(0,1)$, **Impr**.*len associated with a randomly generated solution is* $O(n)$.

Proof. We first consider the distribution of evaluation values of f. The continuous probability distribution for the sum of m independent and identically distributed $U(0,1)$ random variables is Irwin-Hall distribution with mean value $\frac{m}{2}$ [Hal27] (see figure 4.1). Thus the evaluation value associated with a randomly generated solution $\mathbf{x}$ that can be viewed as a sample from Irwin-Hall distribution is expected to be $\frac{m}{2}$. The evaluation values of its n neighbors are basically n more samples from the same distribution. Since Irwin-Hall distribution is symmetric, there are expected to be $\frac{n}{2}$ neighbors better than $\mathbf{x}$. **Impr**.*len* associated with $\mathbf{x}$ is $O(n)$. □

We learn from lemma 17 that only a constant number of bits can possibly be added to or removed from **Impr**. In order to analyze the amortized **Impr**.*len* over the search process, we need to acquire the changing pattern of **Impr**.*len* as a randomly initialized

[10]Figure is modified from `http://en.wikipedia.org/w/index.php?title=File:Irwin-hall-pdf.svg`

candidate solution approaches a local optimum. Intuitively, we expect **Impr**.*len* to decrease as the candidate solution approaches a local optimum, and eventually reach 0 (hitting a local optimum). However, there might be some *exception* where a candidate solution moves to a better neighbor while its **Impr**.*len* increases as well. Such exceptions cannot occur all the time, since the best solution of the entire search space is bounded. The question is though how often such exceptions actually happen. We pose a hypothesis regarding the correlation between evaluation value of a solution and its **Impr**.*len*. If there is a strong negative correlation between these two factors, we are confident that the described exception rarely happens in practice.

Hypothesis 4. *On a uniform random k-bounded pseudo-Boolean function, the evaluation value of candidate solutions sampled by exact-Walsh-LS is negatively correlated with their relative* **Impr**.*len.*

Experiment 4. We run exact-Walsh-LS on both uniform random and non-uniform random NK-landscapes instances with $n = \{20, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500\}$ and $K = \{2, 4\}$. MaxBitFlip is set to 1000000. We record the evaluation value of every candidate solution sampled by exact-Walsh-LS as well as their relative **Impr**.*len* on both uniform random and non-uniform random instances.

Previously in chapter 3, $MaxBitFlip$ was 100,000 and repeat trials 10 times for each configuration. In experiment 4, however, the manually enforced restart for 10 times is not necessary, because we can set $MaxBitFlip$ to be 1,000,000 and give exact-Walsh-LS the choice to restart a search thread when necessary.

We report the correlation between evaluation value and **Impr**.*len* on *uniform* random instances in figure 4.2. The results for *non-uniform* instances are also presented in figure 4.3 for reference purposes. The two factors under consideration are clearly negatively correlated. After taking an improving move, **Impr**.len is very unlikely to increase. Interestingly, the trend corresponding to non-uniform instances in figure 4.3 bends a bit when **impr**.*len* gets closer to 0, while on uniform instances in figure 4.2 it mostly stays straight. We observe that

when the search approaches to a local optimum, **Impr**.*len* on non-uniform instances is likely to be longer than that on uniform instances. We now pose the hypothesis regarding the expected **Impr**.*len* during the execution of exact-Walsh-LS.

Hypothesis 5. *The average* **Impr**.*len during exact-Walsh-LS is* $O(n)$.

We report the average **Impr**.*len* over one million bit-flips by exact-Walsh-LS in figure 4.4. Nonlinear regression is performed to compute a polynomial model for the points in figure 4.4. Indeed, on uniform instances, the growth is nearly linear ($=O(n)$). In contrast, the growth on non-uniform instances is a bit faster than linear. It is attributable to the "bend" we observe in figure 4.3.

We consider a way to break the average $O(n)$ **Impr**.*len*: what if we only force a constant number of bits to flip rather than trigger a random restart (which in expectation flips $\frac{n}{2} = O(n)$ bits) after hitting a local optimum? We refer to the former approach as *random walk* or *soft restart*, the number of bits to flip as λ, and the corresponding algorithm as "walk-Walsh-LS". We expect that flipping λ bits leads to a constant increase in **Impr**.*len* (which was previously zero). In that way, the only $O(n)$ cost is introduced by the one time initialization, which can be amortized over multiple random walks. As a result, the average **Impr**.*len* is $O(1)$. The $O(1)$ result should hold regardless of the distribution of variables over subfunctions.

Hypothesis 6. *On a k-bounded pseudo-Boolean function, the average* **Impr**.*len associated with a candidate solution sampled by walk-Walsh-LS is* $O(1)$.

Experiment 5. We run walk-Walsh-LS with $\lambda = 10$ on both uniform random and non-uniform random NK-landscapes instances with $n = \{20, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500\}$ and $K = \{2, 4\}$. MaxBitFlip is set to $1,000,000$. We record the **Impr**.*len* for each candidate solution sampled by walk-Walsh-LS on both uniform random and non-uniform random instances.

We report the average **Impr**.*len* versus n in figure 4.5. The average **Impr**.*len* is clearly bounded by some constant regardless of the setting of n, and this conforms to our previous

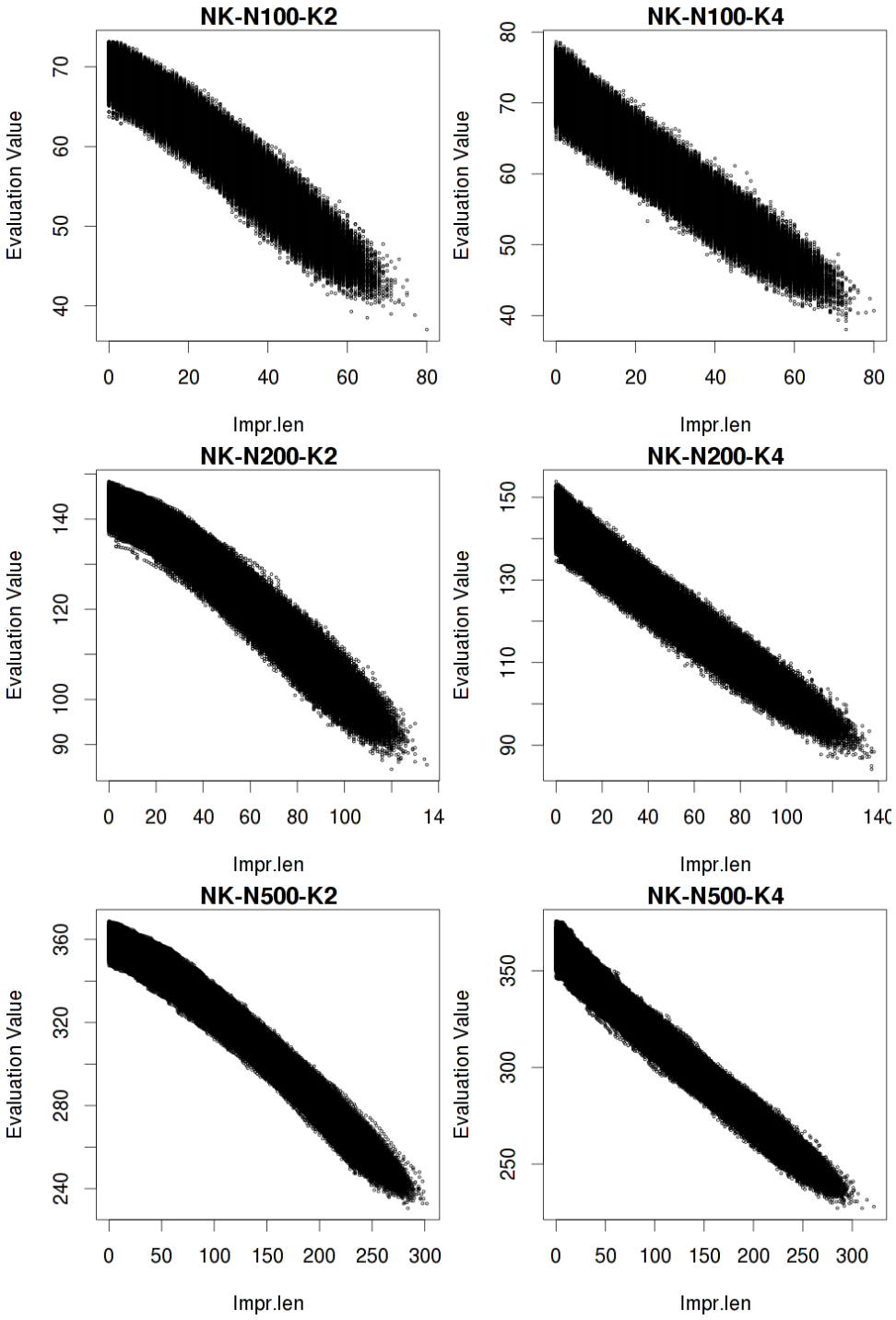

Figure 4.2: Correlation between evaluation value of candidate solutions sampled by exact-Walsh-LS and their relative **Impr**.*len* on *uniform* random NK-landscape instances.

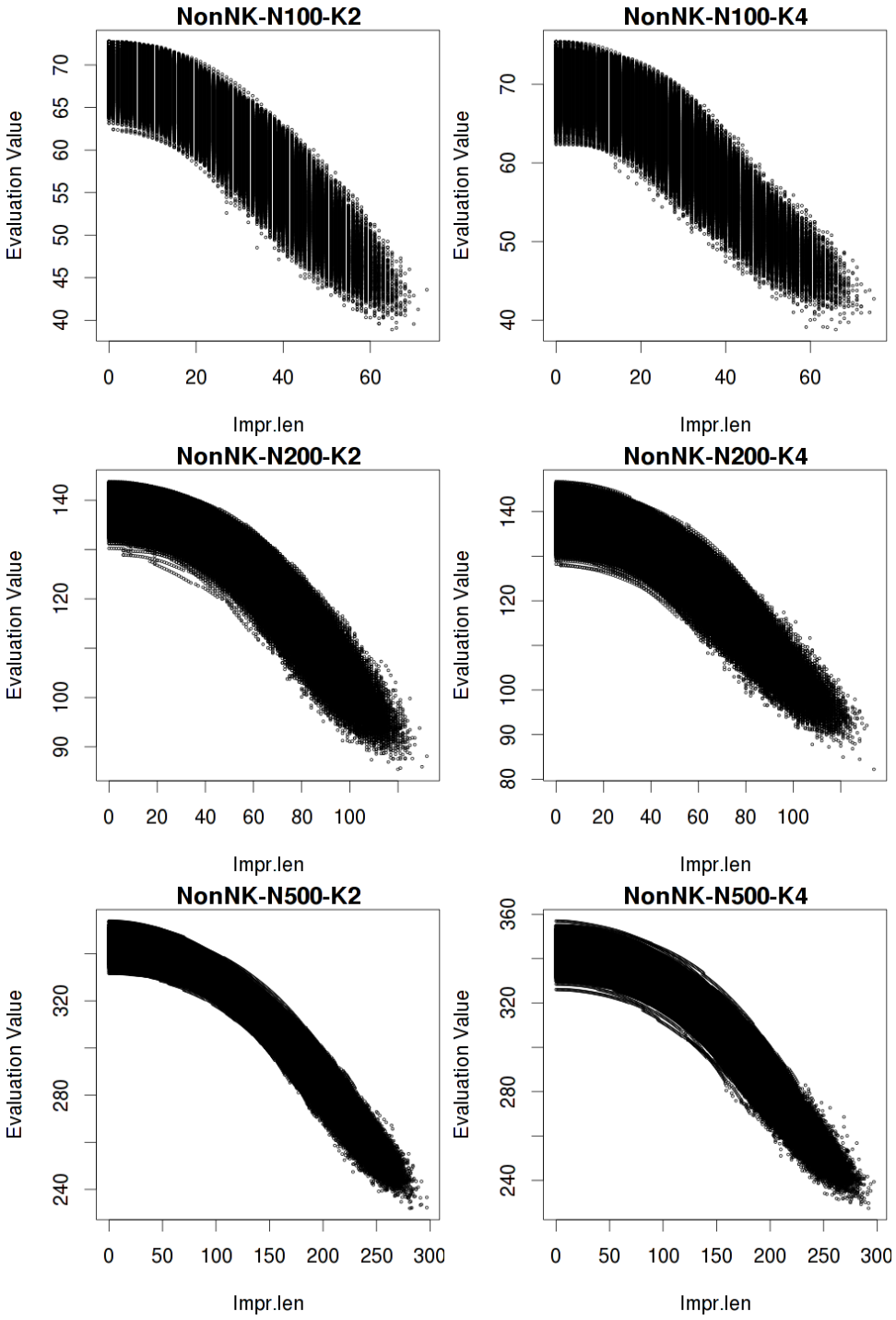

Figure 4.3: Correlation between evaluation value of candidate solutions sampled by exact-Walsh-LS and their relative **Impr**.*len* on *non-uniform* random NK-landscape instances.

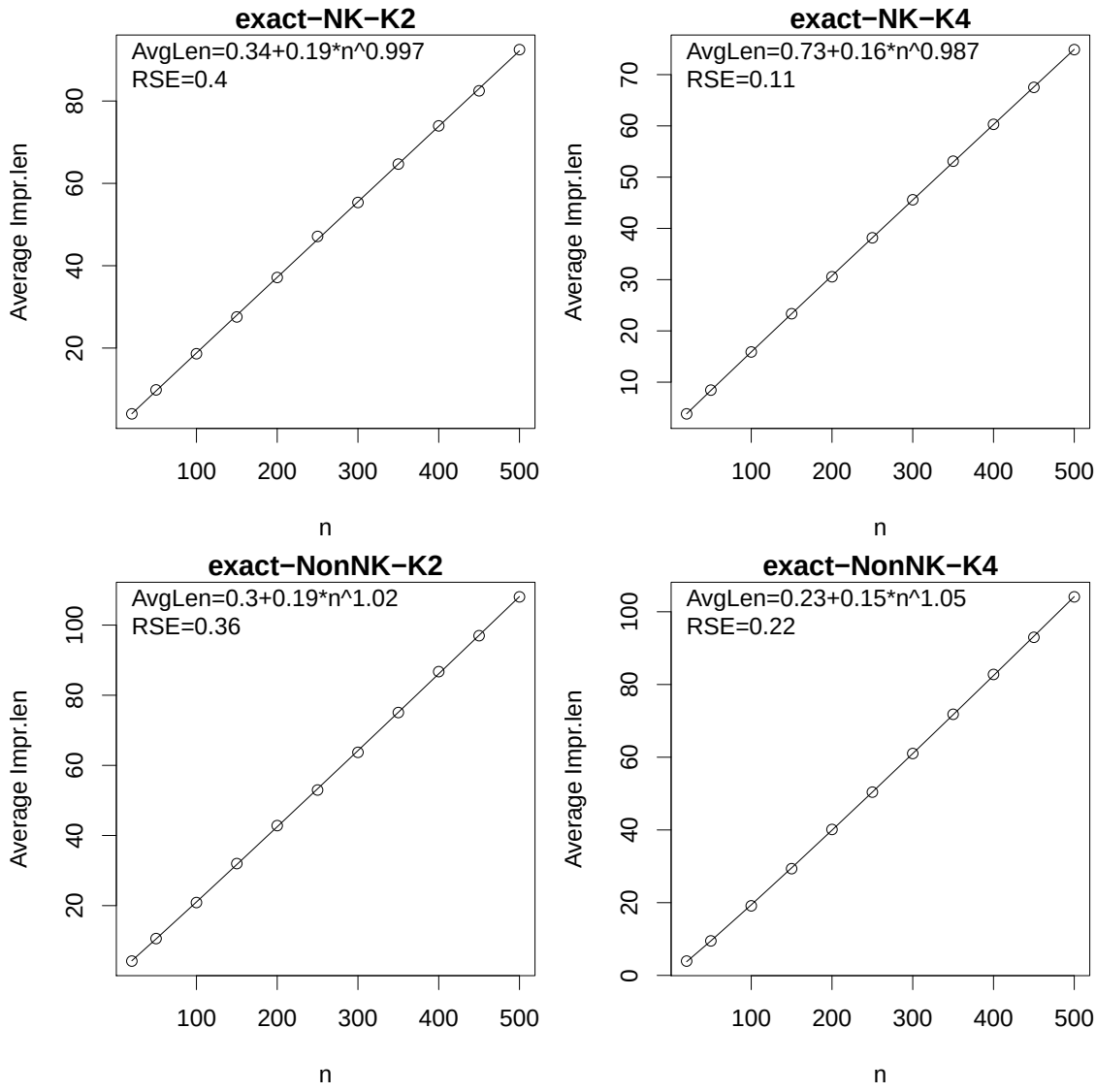

Figure 4.4: The average **Impr**.*len* over one million bit-flips by exact-Walsh-LS on uniform and non-uniform instances. The fitted curve is generated using the nonlinear regression "nls" from R with the formula $AvgLen = a + b * N^c$. The residual standard error ("RSE") is reported to indicate the goodness of fitting.

analysis. However, the average **Impr**.*len* corresponding to smaller n like 50 is remarkably lower than that with larger n. For example, in the top-left subfigure, the **Impr**.*len* for $n = 20$ is about 4, while those for n greater than 100 are around 8. This is because when n is as small as 20, a random restart is expected to flip $\frac{n}{2} = 10$ bits, which is just the same as λ.

4.1.2 Runtime

Three major data structures are maintained in Walsh-LS: vector $\mathbf{w}'(\mathbf{x})$, vector $\mathbf{S}(\mathbf{x})$ and list $\mathbf{Impr}(\mathbf{x})$. What prevents Walsh-LS from achieving $O(1)$ complexity in updating data structures is the complete scan over the $O(n)$-length **Impr**. Now that we have empirically shown that **Impr**.*len* after substituting random restart with random walk is $O(1)$, walk-Walsh-LS should achieve $O(1)$ complexity for updating data structures. We thus pose hypothesis 7.

Hypothesis 7. *On uniform random NK-landscape instances, the update time of walk-Walsh-LS is* $O(1)$.

The question now is how much the $O(n)$ initialization step would affect the overall runtime. We conjuncture that the $O(n)$ one time start-up cost can be amortized over a large number of moves (100,000 in our case), so the growth of the overall runtime should be slower than $O(n)$.

Hypothesis 8. *On uniform random NK-landscape instances, the overall runtime of walk-Walsh-LS grows sublinearly.*

Experiment 6. We run walk-Walsh-LS with $\lambda = 10$ on both uniform and non-uniform random NK-landscapes instances with $n = \{20, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500\}$ and $K = \{2, 4\}$. MaxBitFlip is set to 100000 for each algorithm on every instance. To make the results statistically sound, the trials for each configuration are repeated 10 times independently. Best-so-far solutions, overall runtime, and time spent on updating data structures are recorded.

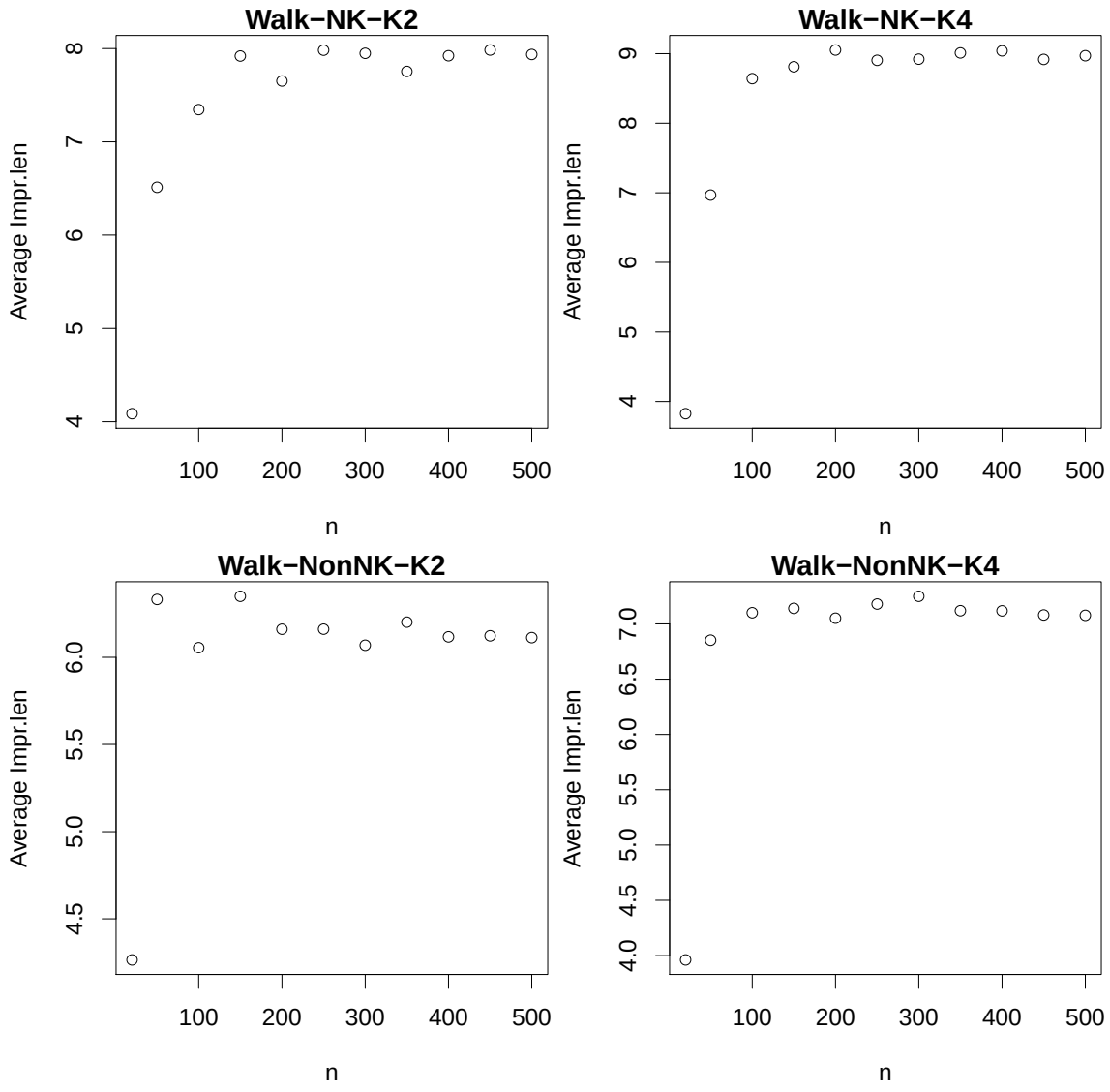

Figure 4.5: The average **Impr**.*len* over one million bit-flips by walk-Walsh-LS on uniform and non-uniform instances.

We report the overall runtime as well as the time for updating data structures on uniform instances in figure 4.6 and on non-uniform instances in figure 4.7. Non-linear regression is applied to fit a runtime model using formula $F = a + b * N^c$. We fail to obtain running models within the standard error threshold 10^{-6} for the update time under all settings of K on both uniform and non-uniform instances. It is mainly attributable to the fact that the update time for $n \geq 150$ flattens out while that for smaller n is substantially smaller. There is no polynomial model that can easily fit such growth. We conclude that the update time is indeed bounded by some constant, and there is barely any increase in the update time after $n = 150$. We therefore verify hypothesis 7. Regarding the growth of the overall runtime, the exponent of n in the formula clearly indicates that the overall runtime indeed increases in a sublinear manner. c ranges from 0.19 to 0.59. Hypothesis 8 is verified empirically.

We have empirically demonstrated the advantage of substituting random restart with random walk in terms of efficiency. Our next concern is whether the increase in efficiency is at the expense of sacrificing the solution quality.

4.1.3 Solution Quality

We now study the solution quality returned by exact-Walsh-LS and walk-Walsh-LS on both *uniform* random and *non-uniform* instances in experiment 6. The solution quality obtained by exact-Walsh-LS and walk-Walsh-LS on uniform random instances is presented in table 4.1. We perform Wilcoxon rank sum test on evaluations of solutions returned by exact-Walsh-LS and walk-Walsh-LS. Since 44 statistical tests (including those on both *uniform* random instances and *non-uniform* random instances) are conducted simultaneously, Bonferroni adjustment [Dun61] is applied to the overall significance level $\alpha = 0.05$. The adjusted significance level for each test is $\frac{\alpha}{n} = \frac{0.05}{44} = 0.0011$.

Except on small instances where both exact-Walsh-LS and walk-Walsh-LS find optimal solutions, walk-Walsh-LS consistently discovers better solutions. The superiority of random walk over random restart is due to the fact that a hard restart upon reaching a local optimum completely undoes the efforts of previous local search. Soft restart (random walk), however,

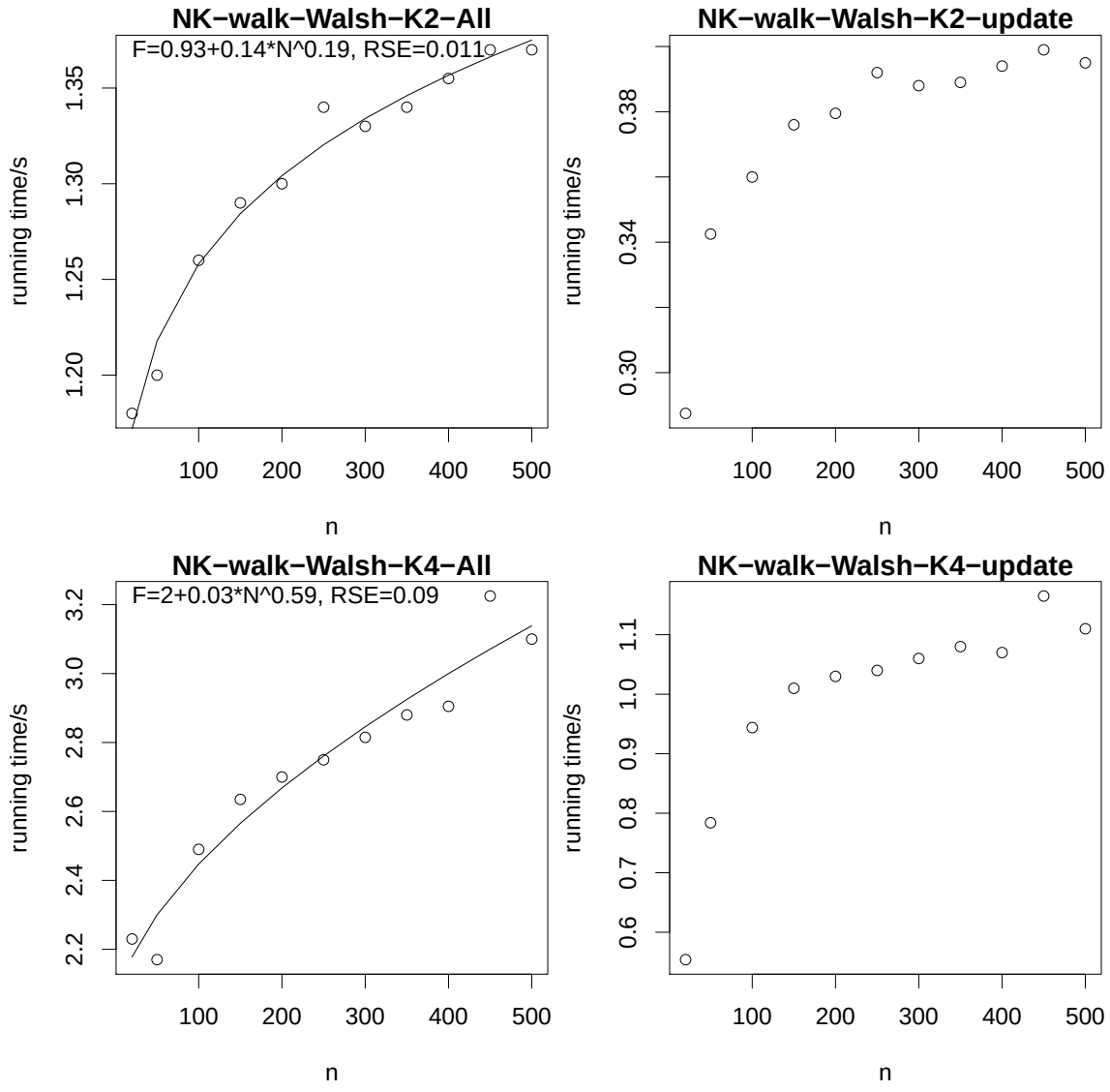

Figure 4.6: Runtime in seconds on *uniform* instances. The fitted curve is generated using the nonlinear regression "nls" from R with the formula $F = a + b * N^c$. The residual standard error ("RSE") is reported to indicate the goodness of fitting.

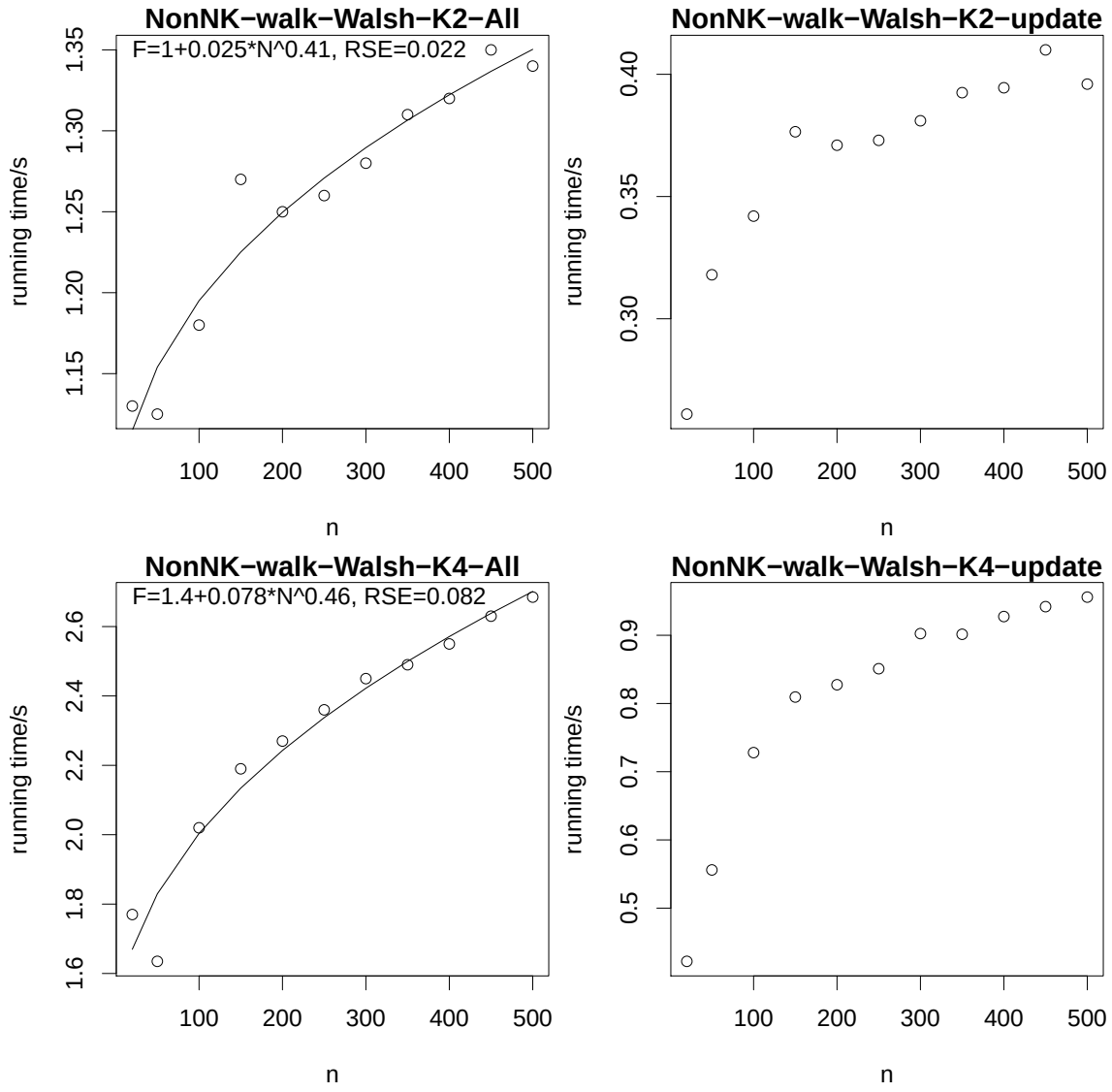

Figure 4.7: Runtime in seconds on *non-uniform* instances. The fitted curve is generated using the nonlinear regression "nls" from R with the formula $F = a + b * N^c$. The residual standard error ("RSE") is reported to indicate the goodness of fitting.

Table 4.1: Evaluations (maximization) of solutions on *uniform* random instances found by exact-Walsh-LS and walk-Walsh-LS. Mean values and standard deviations over 10 independent runs are reported. P-values calculated from Wilcoxon rank-sum test are also presented. Statistical significantly better solutions and the related p-values are marked in **bold** with Bonferroni adjusted significance level $\frac{0.05}{44} = 0.0011$.

	K=2			K=4		
n	exact-Walsh-LS	walk-Walsh-LS	p-value	exact-Walsh-LS	walk-Walsh-LS	p-value
20	14.36±0.00	14.36±0.00	NaN	15.43±0.00	15.43±0.00	NaN
50	38.65±0.00	38.65±0.00	NaN	39.38±0.13	39.44±0.00	0.1675
100	73.07±0.09	**73.16±0.00**	**< 0.0001**	77.81±0.43	**79.59±0.05**	**< 0.0001**
150	111.99±0.22	**112.80±0.02**	**< 0.0001**	114.92±0.57	**118.14±0.36**	**0.0002**
200	147.88±0.37	**149.17±0.05**	**0.0002**	152.03±0.95	**157.06±0.53**	**< 0.0001**
250	184.71±0.49	**187.26±0.14**	**0.0002**	189.66±0.75	**196.78±0.54**	**< 0.0001**
300	223.02±0.65	**226.50±0.17**	**< 0.0001**	225.02±0.93	**233.57±0.80**	**< 0.0001**
350	262.39±0.57	**266.93±0.19**	**0.0002**	262.20±1.35	**272.74±0.73**	**< 0.0001**
400	296.20±0.50	**302.24±0.20**	**< 0.0001**	297.58±1.14	**310.24±0.67**	**< 0.0001**
450	328.42±0.60	**335.00±0.54**	**< 0.0001**	335.04±0.51	**349.83±1.33**	**< 0.0001**
500	368.12±0.71	**374.98±0.30**	**< 0.0001**	374.28±1.15	**390.15±1.70**	**< 0.0001**

only partially reinitializes the solution, and search focuses more on close-to-local-optima regions.

We next study the solution quality by exact-Walsh-LS and walk-Walsh-LS on non-uniform random instances, which are generated using algorithm 6. The results are summarized in table 4.2. As opposed to the results on *uniform* random instances, exact-Walsh-LS with random restarts instead consistently returns solutions with better (or the same) mean values in evaluations, compared with walk-Walsh-LS on all *non-uniform* random instances. The difference in the evaluations of solutions is even more pronounced on *non-uniform* random instances with larger K. Exact-Walsh-LS finds statistically significantly better solutions on 7 out of 11 *non-uniform* random instances with $K = 4$, while it is only statistically significantly better than walk-Walsh-LS on 4 out of 11 *non-uniform* random instances with $K = 2$.

Non-uniform random instances seem to favor hard random restarts over soft random restarts upon reaching local optima. Similar phenomena have also been observed in the Maximum-Satisfiability (MAX-SAT) domain. Smyth *et al.* [SHS03] report that IRoTS (a SLS algorithm) has difficulties in finding good solutions on structured MAX-SAT instances, while it performs exceptionally well on random MAX-SAT instances.

Table 4.2: Evaluations (maximization) of solutions on *non-uniform* random instances found by exact-Walsh-LS and walk-Walsh-LS. Mean values and standard deviations over 10 independent runs are reported. P-values calculated from Wilcoxon rank-sum test are also presented. Statistical significantly better solutions and the related p-values are marked in **bold** with Bonferroni adjusted significance level $\frac{0.05}{44} = 0.0011$.

	K=2			K=4		
n	exact-Walsh-LS	walk-Walsh-LS	p-value	exact-Walsh-LS	walk-Walsh-LS	p-value
20	14.17±0.00	14.17±0.00	NaN	14.83±0.00	14.83±0.00	NaN
50	35.78±0.00	35.78±0.00	NaN	37.02±0.00	37.02±0.00	NaN
100	72.73±0.00	72.73±0.00	NaN	75.40±0.00	74.92±0.41	0.0050
150	107.46±0.00	106.90±0.54	0.0022	109.98±0.14	109.02±0.99	0.0080
200	**143.63±0.16**	143.25±0.58	0.0933	**146.08±0.41**	142.79±1.53	**< 0.0001**
250	175.84±0.34	175.00±1.05	0.1304	**183.46±0.72**	177.55±3.51	**< 0.0001**
300	214.73±0.40	214.04±1.66	0.5437	**216.60±0.50**	209.97±3.56	**< 0.0001**
350	**249.84±0.58**	246.62±2.12	**< 0.0001**	**251.82±0.99**	243.14±3.11	**< 0.0001**
400	286.95±0.31	284.01±3.15	0.0892	**285.51±0.86**	278.37±3.13	**< 0.0001**
450	**319.91±0.51**	317.06±1.89	**< 0.0001**	**321.63±1.47**	311.60±4.23	**< 0.0001**
500	**352.24±1.13**	348.20±2.08	**0.0003**	**354.69±1.07**	346.62±3.60	**0.0005**

In order to shed some light on the underlying reason for the flip in comparison between exact-Walsh-LS and walk-Walsh-LS on solution quality, we pose hypothesis 9.

Hypothesis 9. *On non-uniform random NK-landscape instances, walk-Walsh-LS with* $\lambda = 10$ *finds worse solutions than exact-Walsh-LS due to the fact that* $\lambda = 10$ *is insufficient to escape local optima on non-uniform instances.*

Upon reaching a local optimum LO, if the walk length for a random walk is insufficient, applying local search undoes the effect of the random walk and the search falls back to LO. Motivated by this conjecture, we define an *unsuccessful* restart as in definition 4.1.1. Similarly, we can define a *successful* restart as in definition 4.1.2.

Definition 4.1.1. A restart (either soft or hard) upon a local optimum LO is *unsuccessful*, if the effect of the restart is undone after applying local search and the search falls back to the same local optimum LO.

Definition 4.1.2. A restart (either soft or hard) upon a local optimum LO is *successful*, if applying local search after a restart leads to a local optimum other than LO.

We revise experiment 6 to highlight the effectiveness of restart in escaping local optima. The revised experiment is described in experiment 7.

Table 4.3: Number of successful restarts over 100,000 moves by exact-Walsh-LS and walk-Walsh-LS on *uniform* random instances. Mean values and standard deviations over 10 independent runs are reported.

	K=2		K=4	
n	exact-Walsh-LS	walk-Walsh-LS	exact-Walsh-LS	walk-Walsh-LS
20	12614.70±67.96	12486.10±46.02	17523.10±54.95	17540.90±38.76
50	5254.00±10.54	6436.60±65.38	7139.30±15.33	8863.80±39.70
100	2781.30±7.97	4435.20±63.60	3641.70±14.10	5519.00±59.84
150	1814.20±4.57	3197.70±105.66	2373.40±5.08	3090.00±147.34
200	1323.10±3.60	2303.20±65.42	1774.40±6.70	2192.40±171.49
250	1102.30±1.89	1957.90±69.32	1404.60±3.47	1526.30±185.25
300	906.80±2.53	1643.10±44.84	1162.10±3.25	1298.50±165.34
350	767.50±2.88	1164.20±50.18	991.80±2.66	1067.00±84.26
400	694.50±1.96	1304.10±57.44	880.00±2.49	1022.80±85.56
450	609.50±1.84	1170.90±84.48	778.50±2.68	936.70±47.54
500	550.20±1.23	983.60±43.70	686.10±2.23	818.60±17.16

Experiment 7. We run exact-Walsh-LS and walk-Walsh-LS with $\lambda = 10$ on both uniform and non-uniform random NK-landscapes instances with $n = \{20, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500\}$ and $K = \{2, 4\}$. MaxBitFlip is set to 100000 for each algorithm on every instance. The trials for each configuration are repeated 10 times independently. The number of *successful restarts* and the number of restarts for each run are recorded.

The numbers of successful restarts on uniform random instances and non-uniform random instances are respectively reported in table 4.3 and table 4.4. We observe that on uniform random instances, walk-Walsh-LS consistently performs more successful restarts than exact-Walsh-LS across all tested instances. It suggests walk-Walsh-LS is able to more effectively sample distinct local optima in the search space, which in turn leads to better solutions. On non-uniform random instances, however, the numbers of successful restarts by walk-Walsh-LS are typically much smaller than those by exact-Walsh-LS. The trend is more evident on instances with large n. This explains why walk-Walsh-LS finds inferior solutions compared with exact-Walsh-LS on non-uniform instances.

Table 4.4: Number of successful restarts over 100,000 moves by exact-Walsh-LS and walk-Walsh-LS on *non-uniform* random instances. Mean values and standard deviations over 10 independent runs are reported.

	K=2		K=4	
n	exact-Walsh-LS	walk-Walsh-LS	exact-Walsh-LS	walk-Walsh-LS
20	12665.70±48.90	12883.20±41.69	15548.90±33.93	15767.00±32.68
50	5083.20±9.02	4963.00±62.30	5722.90±16.65	4852.00±52.57
100	2427.90±6.54	947.90±51.68	2719.80±5.71	941.00±73.39
150	1598.80±3.19	552.20±112.88	1744.30±4.64	439.00±88.12
200	1175.20±1.40	293.60±95.63	1266.40±2.63	253.10±107.41
250	955.90±1.85	255.80±67.46	1006.80±2.25	151.30±65.42
300	785.20±1.48	169.10±48.33	829.80±2.62	140.50±44.83
350	677.40±1.71	132.50±39.71	706.20±1.14	94.00±34.13
400	579.90±0.99	79.80±37.06	610.20±1.23	115.30±51.47
450	516.40±1.07	111.60±33.31	542.00±0.94	89.50±25.91
500	465.80±0.92	75.60±18.22	484.50±1.08	67.70±24.67

To validate hypothesis 9, the total number of restarts are further presented in table 4.5 and table 4.6. It is clear that exact-Walsh-LS with hard restarts almost never visits the same local optimum twice in a row, since most restarts are *successful*. walk-Walsh-LS with $\lambda = 10$ on the other hand has a high percentage of *unsuccessful* restarts, especially on non-uniform instances. Take non-uniform instance with $n = 500$ and $K = 4$ for example, on average only 67 out of 9980 restarts (0.067%) are successful. It suggests $\lambda = 10$ may be insufficient for walk-Walsh-LS to escape local optima, particular on non-uniform instances. Hypothesis 9 is hence empirically verified.

To show the potential of walk-Walsh-LS on non-uniform instances, we pick the non-uniform instance with $n = 500$ and $K = 4$ where the number of successful restarts for exact-Walsh-LS is especially low and adjust λ in hopes of improving the performance and possibly surpassing exact-Walsh-LS.

Experiment 8. We run walk-Walsh-LS with λ ranging from 10 to 500 with an interval 10 on non-uniform random NK-landscapes instances with $n = 500$ and $K = 4$. MaxBitFlip is

Table 4.5: Number of restarts over 100,000 moves by exact-Walsh-LS and walk-Walsh-LS on *uniform* random instances. Mean values and standard deviations over 10 independent runs are reported.

	K=2		K=4	
n	exact-Walsh-LS	walk-Walsh-LS	exact-Walsh-LS	walk-Walsh-LS
20	13348.50±52.24	12889.30±41.67	17707.30±50.24	17558.40±39.49
50	5263.90±11.10	10209.40±11.86	7139.30±15.33	9615.50±25.30
100	2781.30±7.97	9999.20±8.38	3641.70±14.10	9456.30±13.37
150	1814.20±4.57	9936.50±8.73	2373.40±5.08	9742.20±39.84
200	1323.10±3.60	9928.30±10.85	1774.40±6.70	9821.70±22.26
250	1102.30±1.89	9974.50±8.34	1404.60±3.47	9886.50±22.93
300	906.80±2.53	9957.00±6.32	1162.10±3.25	9911.70±13.85
350	767.50±2.88	9956.80±3.77	991.80±2.66	9921.00±12.99
400	694.50±1.96	9972.30±5.36	880.00±2.49	9924.60±5.34
450	609.50±1.84	9971.70±4.22	778.50±2.68	9928.60±9.47
500	550.20±1.23	9972.30±5.27	686.10±2.23	9937.50±4.65

Table 4.6: Number of restarts over 100,000 moves by exact-Walsh-LS and walk-Walsh-LS on *non-uniform* random instances. Mean values and standard deviations over 10 independent runs are reported.

	K=2		K=4	
n	exact-Walsh-LS	walk-Walsh-LS	exact-Walsh-LS	walk-Walsh-LS
20	13643.60±39.23	13323.30±32.21	16050.90±21.64	15886.60±26.89
50	5120.00±7.97	9928.30±12.37	5733.20±15.00	9030.30±33.32
100	2430.50±5.50	10018.80±8.34	2719.80±5.71	9859.70±20.56
150	1599.10±3.45	10002.30±8.58	1744.30±4.64	9994.40±10.66
200	1175.20±1.40	9998.90±6.74	1266.40±2.63	9987.20±7.58
250	955.90±1.85	9995.90±5.53	1006.80±2.25	9994.00±3.13
300	785.20±1.48	9986.90±3.00	829.80±2.62	9987.80±7.33
350	677.40±1.71	9987.00±3.80	706.20±1.14	9990.00±3.20
400	579.90±0.99	9984.30±1.64	610.20±1.23	9987.40±2.63
450	516.40±1.07	9981.50±2.64	542.00±0.94	9984.60±1.65
500	465.80±0.92	9979.60±1.96	484.50±1.08	9980.70±3.86

set to 100000 for each algorithm on every instance. Each configuration is repeated 10 times independently. Best-so-far solutions are recorded.

The impact of λ on the performance of walk-Walsh-LS is illustrated in figure 4.8. It can be concluded that by carefully choosing λ, walk-Walsh-LS can find better solutions than exact-Walsh-LS. We also observe that in the interval of $\lambda = [10, 120]$, increasing λ generally enhances the performance of walk-Walsh-LS. This shows that the potential of walk-Walsh-LS can be better exploited with a suitable setting of λ.

Summing up the studies in subsection 4.1.2 and subsection 4.1.3, walk-Walsh-LS clearly dominates exact-Walsh-LS in terms of efficiency. From the perspective of effectiveness, however, the λ parameter should be carefully chosen for walk-Walsh-LS to discover better solutions than exact-Walsh-LS. We also found λ appears to rely on the characteristics of the problem instances under investigation, since $\lambda = 10$ seems to be insufficient for non-uniform instances while it may be large enough for uniform instances. The optimal values of λ on non-uniform instances seem higher than that on uniform instances.

4.2 Best-Improvement vs. First-Improvement

We now consider how to achieve $O(1)$ complexity per move on uniform random instances by *only* breaking the second condition listed at the beginning of this chapter: determining the true best-improvement move requires a complete scan over **Impr**, which takes $O(n)$. Instead of taking the move yielding the highest improvement in evaluation, we can flip the first encountered bit that leads to improvement in evaluation. In such a way, the $O(n)$ cost of scanning the complete **Impr** list can be avoided. We refer to the SLS with this heuristic for selecting improving moves as *first-improvement local search (FILS)*. Specifically, Walsh-LS that takes first improving moves is called *Walsh-FILS*.

4.2.1 Runtime

Experiment 9. We run Walsh-FILS on both uniform and non-uniform random NK-landscapes instances with $n = \{20, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500\}$ and $K = \{2, 4\}$. MaxBit-Flip is set to $100,000$ for each algorithm on every instance. The trials for each configuration are repeated 10 times independently. Best-so-far solutions, overall runtime, and time spent on updating data structures are recorded.

The median runtimes of Walsh-FILS on uniform instances and non-uniform instances are reported in figure 4.9 and figure 4.10. The two figures suggest that the update time is bounded by some constant. We also find such a trend holds for both uniform and non-uniform instances. For example, top-right subfigure of figure 4.9 indicates that the update time only raises from 0.45 seconds to 0.55 seconds when n increases from 20 to 500 on random instances. We even observe a declining trend of the update time on non-uniform instances when n grows.

Even though Walsh-FILS has to pay a $O(n)$ initialization cost, it still achieves a great scalability in the overall runtime. Raising n from 20 to 500 only increases the overall runtime from 1.35 seconds to 1.5 seconds on uniform instances with $K = 2$ for instance.

4.2.2 Solution Quality

Our next concern is whether the improvement in runtime by Walsh-FILS is at the cost of sacrificing the solution quality. The comparisons between Walsh-FILS and exact-Walsh-LS in terms of quality of solutions are summarized in table 4.7 and table 4.8.

On uniform instances with $K = 2$, Walsh-FILS finds worse solutions than exact-Walsh-LS on uniform random instances in cases where n is sufficiently large to differentiate Walsh-FILS and exact-Walsh-LS. Wilcoxon rank-sum tests indicate that 6 out of 9 such cases are statistically significant. When K increases to 4 on uniform instances, however, the advantage of exact-Walsh-LS over Walsh-FILS is not statistically significant. When distribution of variables over subfunctions changes from uniform to non-uniform (binomial), exact-Walsh-LS still returns better solutions than Walsh-FILS on not-too-small instances (starting from $n = 100$). Nevertheless, the superiority of exact-Walsh-LS over Walsh-FILS in quality of

Table 4.7: Evaluations (maximization) of solutions on *uniform* random instances found by Walsh-FILS and exact-Walsh-LS. Mean values and standard deviations over 10 independent runs are reported. P-values calculated from Wilcoxon rank-sum test are also presented. Statistical significantly better solutions and the related p-values are marked in **bold** with Bonferroni adjusted significance level $\frac{0.05}{44} = 0.0011$.

	K=2			K=4		
n	Walsh-FILS	exact-Walsh-LS	p-value	Walsh-FILS	exact-Walsh-LS	p-value
20	14.36±0.00	14.36±0.00	NaN	15.43±0.00	15.43±0.00	NaN
50	38.65±0.00	38.65±0.00	NaN	39.38±0.13	39.38±0.13	1.0000
100	72.66±0.20	**73.07±0.09**	**0.0002**	77.74±0.30	77.81±0.43	0.9705
150	111.59±0.19	**111.99±0.22**	**0.0007**	114.64±0.60	114.92±0.57	0.3642
200	147.71±0.37	147.88±0.37	0.2176	151.90±0.67	152.03±0.95	1.0000
250	183.08±0.44	**184.71±0.49**	**< 0.0001**	189.10±0.79	189.66±0.75	0.0753
300	222.20±0.61	223.02±0.65	0.0191	224.49±0.66	225.02±0.93	0.2799
350	261.44±0.90	262.39±0.57	0.0185	262.34±1.66	262.20±1.35	0.9705
400	294.01±0.69	**296.20±0.50**	**< 0.0001**	298.08±1.34	297.58±1.14	0.3930
450	326.84±0.60	**328.42±0.60**	**< 0.0001**	335.87±1.11	335.04±0.51	0.0630
500	366.05±0.57	**368.12±0.71**	**< 0.0001**	374.48±1.15	374.28±1.15	0.7394

Table 4.8: Evaluations (maximization) of solutions on *non-uniform* random instances found by Walsh-FILS and exact-Walsh-LS. Mean values and standard deviations over 10 independent runs are reported. P-values calculated from Wilcoxon rank-sum test are also presented. Statistical significantly better solutions and the related p-values are marked in **bold** with Bonferroni adjusted significance level $\frac{0.05}{44} = 0.0011$.

	K=2			K=4		
n	Walsh-FILS	exact-Walsh-LS	p-value	Walsh-FILS	exact-Walsh-LS	p-value
20	14.17±0.00	14.17±0.00	NaN	14.83±0.00	14.83±0.00	NaN
50	35.78±0.00	35.78±0.00	NaN	37.02±0.00	37.02±0.00	NaN
100	72.60±0.16	72.73±0.00	0.0336	74.97±0.32	**75.40±0.00**	**0.0007**
150	107.15±0.22	107.46±0.00	0.0020	109.68±0.24	109.98±0.14	0.0088
200	143.42±0.21	143.63±0.16	0.0534	144.93±1.06	146.08±0.41	0.0068
250	174.96±0.49	**175.84±0.34**	**0.0002**	181.02±1.05	**183.46±0.72**	**0.0002**
300	213.67±0.73	214.73±0.40	0.0021	214.11±0.74	**216.60±0.50**	**< 0.0001**
350	247.95±0.71	**249.84±0.58**	**< 0.0001**	248.70±1.50	**251.82±0.99**	**< 0.0001**
400	284.57±0.39	**286.95±0.31**	**< 0.0001**	282.58±1.26	**285.51±0.86**	**< 0.0001**
450	317.91±0.88	**319.91±0.51**	**< 0.0001**	315.64±1.07	**321.63±1.47**	**< 0.0001**
500	349.85±0.87	**352.24±1.13**	**< 0.0001**	349.64±1.90	**354.69±1.07**	**< 0.0001**

solutions becomes more pronounced on non-uniform instances with $K = 4$, in 7 out of 9 cases the differences are statistically significant. In short, exact-Walsh-LS finds equally good or better solutions than Walsh-FILS on both uniform and non-uniform NK-Landscapes instances.

We conjecture that it is because that first-improvement local search is typically less greedy than best-improvement local search. Given the same number of moves and the same restart strategy, first-improvement local search requires more moves to hit a local optimum which results in visiting less local optima than best-improvement local search.

Hypothesis 10. *On both uniform random and non-uniform random NK-landscape instances, Walsh-FILS takes more moves to reach a local optimum and visits less local optima when compared with exact-Walsh-LS, given the same number of moves and the same restart strategy.*

Experiment 10. We run Walsh-FILS on both uniform and non-uniform random NK-landscapes instances with $n = \{20, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500\}$ and $K = \{2, 4\}$. MaxBitFlip is set to 100000 for each algorithm on every instance. The trials for each configuration are repeated 10 times independently. The number of *successful restarts* and the number of restarts for each run are recorded.

We compare the number of restarts issued by Walsh-FILS and exact-Walsh-LS on uniform and non-uniform instances separately in table 4.9 and table 4.10.

As expected, exact-Walsh-LS visits remarkably more local optima than Walsh-FILS, which in return leads to more restarts. This suggests that starting from a random point in the search space, Walsh-FILS indeed requires more moves to reach a local optimum. This trend is consistent across all tested uniform and non-uniform instances.

Nonetheless, visiting more local optima does not necessarily lead to more effective sampling of search space. As shown previously, some restarts can be unsuccessful and can be undone by applying local search after restarts. Hereby, we report the number of successful restarts in table 4.11 and table 4.12. We find the hard random restart employed by Walsh-FILS and exact-Walsh-LS avoids the same local optimum being visited successively. Except on

Table 4.9: Number of restarts over 100,000 moves by Walsh-FILS and exact-Walsh-LS on uniform random instances. Mean values and standard deviations over 10 independent runs are reported.

	K=2		K=4	
n	Walsh-FILS	exact-Walsh-LS	Walsh-FILS	exact-Walsh-LS
20	10246.20±39.30	13348.50±52.24	11398.90±44.57	17707.30±50.24
50	4032.10±11.00	5263.90±11.10	4522.70±13.72	7139.30±15.33
100	2073.50±10.14	2781.30±7.97	2266.10±6.72	3641.70±14.10
150	1325.20±5.12	1814.20±4.57	1484.60±4.45	2373.40±5.08
200	994.80±3.36	1323.10±3.60	1115.40±4.01	1774.40±6.70
250	802.90±2.77	1102.30±1.89	887.00±2.58	1404.60±3.47
300	680.80±1.75	906.80±2.53	732.50±2.42	1162.10±3.25
350	575.20±2.30	767.50±2.88	622.00±2.16	991.80±2.66
400	524.40±1.84	694.50±1.96	549.90±1.91	880.00±2.49
450	461.50±1.18	609.50±1.84	486.40±1.26	778.50±2.68
500	409.30±1.34	550.20±1.23	431.30±1.83	686.10±2.23

Table 4.10: Number of restarts over 100,000 moves by Walsh-FILS and exact-Walsh-LS on non-uniform random instances. Mean values and standard deviations over 10 independent runs are reported.

	K=2		K=4	
n	Walsh-FILS	exact-Walsh-LS	Walsh-FILS	exact-Walsh-LS
20	10575.80±22.73	13643.60±39.23	11044.00±33.57	16050.90±21.64
50	4024.90±13.88	5120.00±7.97	4385.20±16.85	5733.20±15.00
100	2040.90±6.71	2430.50±5.50	2119.20±8.72	2719.80±5.71
150	1335.50±3.84	1599.10±3.45	1385.00±5.25	1744.30±4.64
200	981.70±3.06	1175.20±1.40	1046.00±4.92	1266.40±2.63
250	815.00±2.49	955.90±1.85	829.60±3.31	1006.80±2.25
300	682.30±1.83	785.20±1.48	697.10±1.60	829.80±2.62
350	588.20±1.32	677.40±1.71	589.20±2.35	706.20±1.14
400	499.30±1.34	579.90±0.99	519.40±1.35	610.20±1.23
450	446.60±1.35	516.40±1.07	457.10±1.37	542.00±0.94
500	403.40±0.84	465.80±0.92	411.80±1.14	484.50±1.08

Table 4.11: Number of SUCCESSFUL restarts over 100,000 moves by Walsh-FILS and exact-Walsh-LS on uniform random instances. Mean values and standard deviations over 10 independent runs are reported.

	K=2		K=4	
n	Walsh-FILS	exact-Walsh-LS	Walsh-FILS	exact-Walsh-LS
20	10069.00±46.58	12614.70±67.96	11353.80±45.35	17523.10±54.95
50	4031.40±10.97	5254.00±10.54	4522.70±13.72	7139.30±15.33
100	2073.50±10.14	2781.30±7.97	2266.10±6.72	3641.70±14.10
150	1325.20±5.12	1814.20±4.57	1484.60±4.45	2373.40±5.08
200	994.80±3.36	1323.10±3.60	1115.40±4.01	1774.40±6.70
250	802.90±2.77	1102.30±1.89	887.00±2.58	1404.60±3.47
300	680.80±1.75	906.80±2.53	732.50±2.42	1162.10±3.25
350	575.20±2.30	767.50±2.88	622.00±2.16	991.80±2.66
400	524.40±1.84	694.50±1.96	549.90±1.91	880.00±2.49
450	461.50±1.18	609.50±1.84	486.40±1.26	778.50±2.68
500	409.30±1.34	550.20±1.23	431.30±1.83	686.10±2.23

small instances where n is less than or equal to 100, all restarts performed by Walsh-FILS and exact-Walsh-LS are actually *successful* by definition 4.1.2. Moreover, exact-Walsh-LS consistently issues more successful restarts than Walsh-FILS on uniform and non-uniform restarts, which validates hypothesis 10.

The choice between best-improvement local search and first-improvement local search for Walsh-LS is clearly a trade-off between runtime and quality of solutions. While Walsh-LS with first-improvement local search achieves constant complexity per move, it sacrifices the quality of solutions when compared with exact-Walsh-LS with best-improvement local search.

Table 4.12: Number of SUCCESSFUL restarts over 100,000 moves by Walsh-FILS and exact-Walsh-LS on nonuniform random instances. Mean values and standard deviations over 10 independent runs are reported.

	K=2		K=4	
n	Walsh-FILS	exact-Walsh-LS	Walsh-FILS	exact-Walsh-LS
20	10215.70±27.85	12665.70±48.90	10934.40±32.27	15548.90±33.93
50	4021.10±14.42	5083.20±9.02	4384.50±17.00	5722.90±16.65
100	2040.90±6.71	2427.90±6.54	2119.20±8.72	2719.80±5.71
150	1335.50±3.84	1598.80±3.19	1385.00±5.25	1744.30±4.64
200	981.70±3.06	1175.20±1.40	1046.00±4.92	1266.40±2.63
250	815.00±2.49	955.90±1.85	829.60±3.31	1006.80±2.25
300	682.30±1.83	785.20±1.48	697.10±1.60	829.80±2.62
350	588.20±1.32	677.40±1.71	589.20±2.35	706.20±1.14
400	499.30±1.34	579.90±0.99	519.40±1.35	610.20±1.23
450	446.60±1.35	516.40±1.07	457.10±1.37	542.00±0.94
500	403.40±0.84	465.80±0.92	411.80±1.14	484.50±1.08

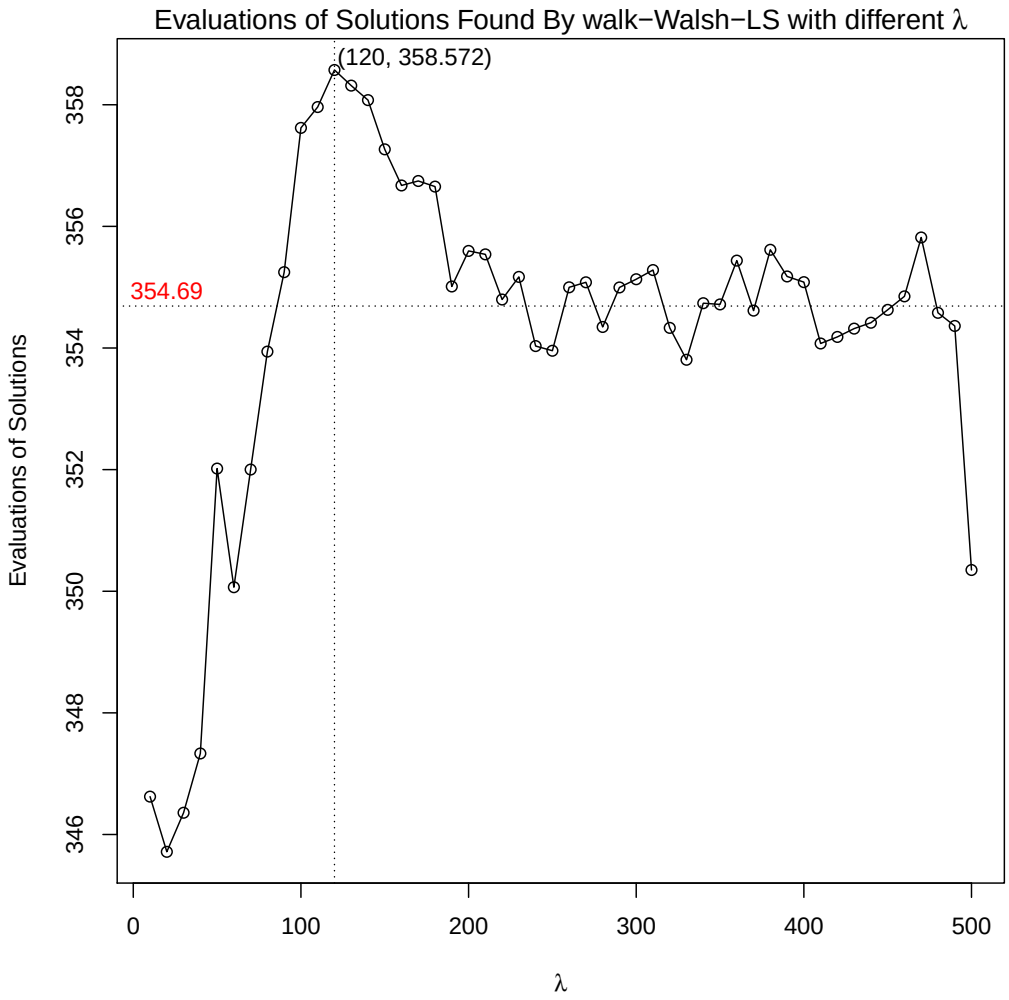

Figure 4.8: Evaluations of solutions found by walk-Walsh-LS with λ ranging from 10 to 500. Each point represents the mean evaluations over 10 independent runs. The highest mean of evaluations (358.572) is achieved by walk-Walsh-LS with $\lambda = 120$. The mean evaluations of solutions found by exact-Walsh-LS (354.69, which is colored in red) is reported as a baseline.

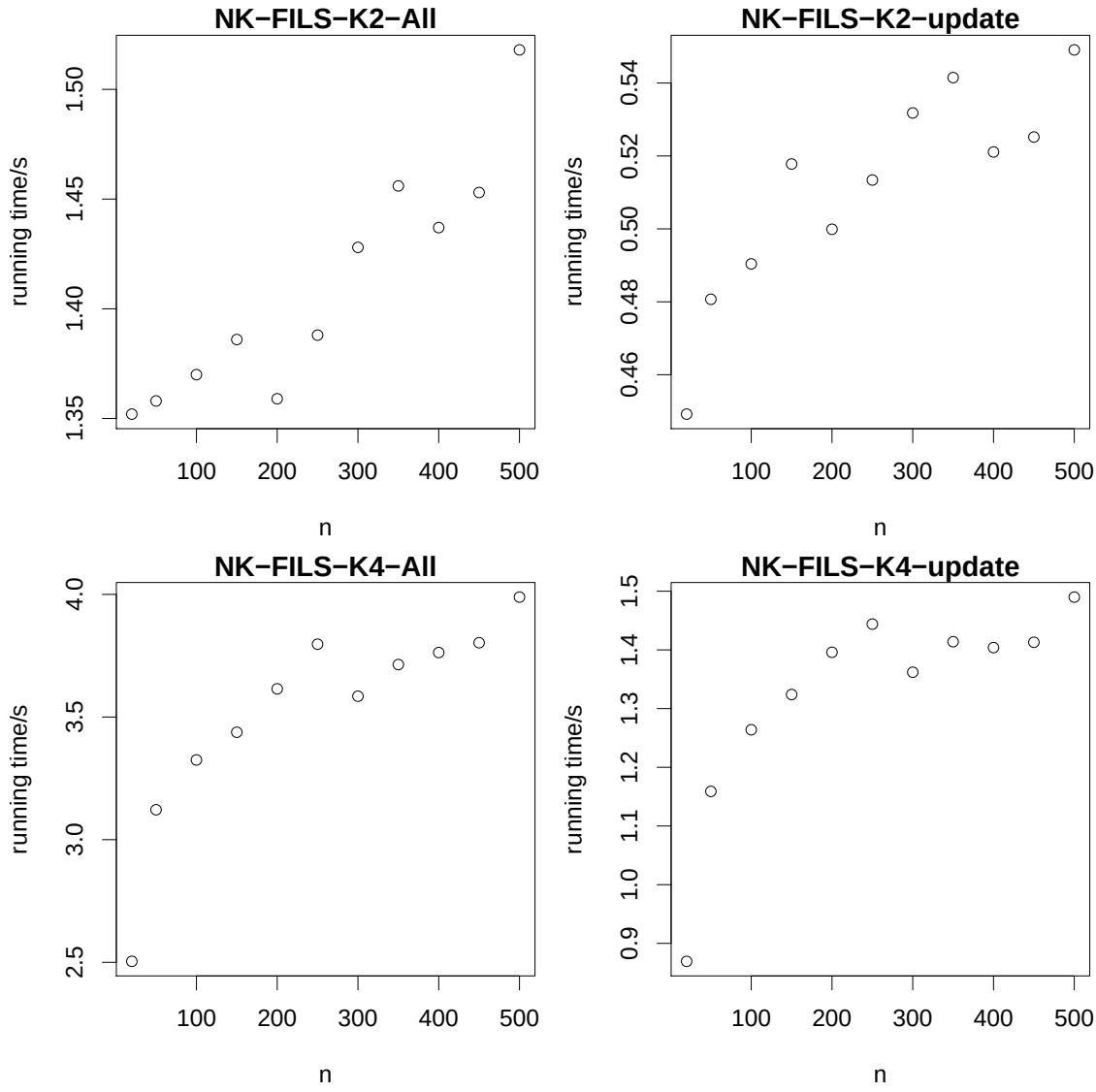

Figure 4.9: Median runtimes of Walsh-FILS in seconds on *uniform* instances over 10 runs. The left subfigures relate to the overall runnning time, while the right subfigures represent the time spent on updating data structures.

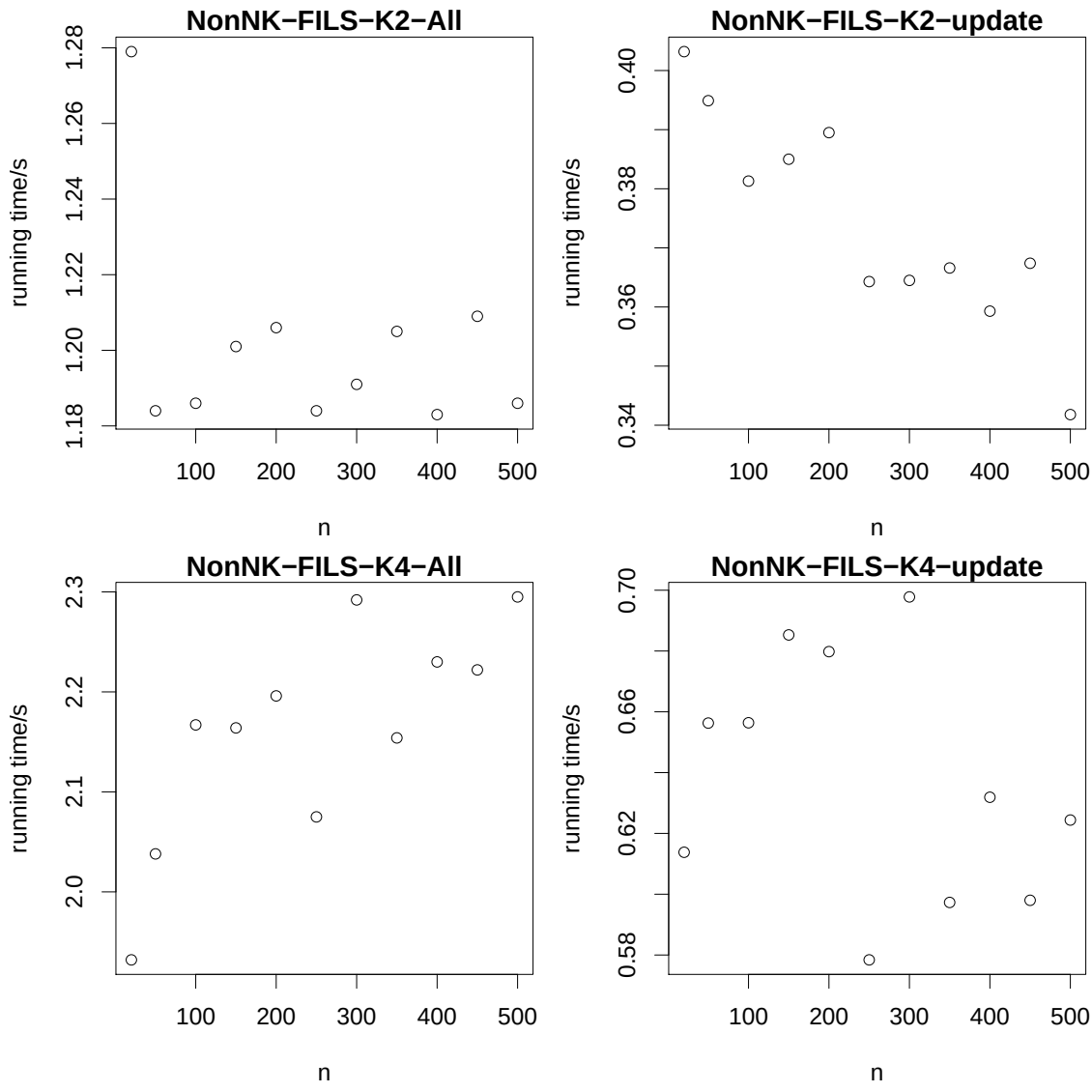

Figure 4.10: Median runtimes of Walsh-FILS in seconds on *non-uniform* instances. The left subfigures relate to the overall runnning time, while the right subfigures represent the time spent on updating data structures.

Chapter 5

Walsh-LS using Surrogate Function of Mean Value over Hamming Region

Plateaus are connected search regions in which all search positions have the same evaluation [HS04]. Since all positions on a plateau share the same evaluation, SLS has no gradient information to follow and thus gets stuck. Plateaus often occur in neutral landscapes, where neighboring positions in the search space are likely to have equal evaluation. The landscapes encountered for SAT instances are classic neutral landscapes [HK96] [FCS97] [SHW10]. This chapter studies how the mean value over a Hamming region can be used as surrogate fitness function to smooth the fitness landscape and to reduce the total number of plateaus. To study the effect of surrogate fitness function, we also consider NKq-landscapes ("quantized" NK [Gea01]) as an example of neutral landscapes in this chapter.

NKq-landscapes [GWH$^+$02] are almost identical to NK-Landscapes except that their subfunctions are lookup tables from uniform distribution of *integers* $[0, q)$, where q is the discretization level. The smaller q is, the less variance appears in the values of subfunctions. Hence there are more plateaus when q is small. We choose q = 2 for our empirical studies. Both NKq-Landscapes and SAT display plateaus and result in more equally good improving moves compared to the standard NK-Landscapes.

From the viewpoint of runtime, we show Walsh-LS using the mean value over a Hamming region as surrogate fitness function still achieves the same complexity per move as the original Walsh-LS. From the perspective of effectiveness, we demonstrate how Walsh-LS employs Walsh terms differently when surrogate fitness function is used. We will also present how the difference in using Walsh terms impacts the quality of solutions as the radius of Hamming regions being averaged varies.

5.1 Mean over Hamming Regions as Surrogate Fitness

5.1.1 Hamming Regions

SLS makes small changes to a candidate solution (e.g., by flipping a bit). The short-term dynamics of such algorithms are influenced by the statistical properties of the states that are mutually reachable by a small number of changes [SWH11]. Accordingly, we are interested in the mean of fitness values over regions that are *local* to a candidate solution. We formalize this as follows. Consider two candidate solutions $\mathbf{x}, \mathbf{y} \in \mathcal{B}^n$. The Hamming distance $\mathcal{D}(\mathbf{x}, \mathbf{y})$ is the number of positions in which the binary strings $\mathbf{x}$ and $\mathbf{y}$ differ. The set $\mathcal{B}^n$ together with the Hamming distance function form a metric space. Given a "centroid" $\mathbf{x}$, we can partition $\mathcal{B}^n$ into regions about $\mathbf{x}$.

A Hamming sphere of radius r around a point $\mathbf{x} \in \mathcal{B}^n$ is defined as the set

$$\mathsf{HS}^{(r)}(\mathbf{x}) = \{\mathbf{y} \in \mathcal{B}^n : \mathcal{D}(\mathbf{x}, \mathbf{y}) = r\}. \tag{5.1}$$

Similarly, we define a union of concentric spheres. A Hamming ball of radius r around a point $x \in \mathcal{B}^n$ is defined as the set

$$\mathsf{HB}^{(r)}(\mathbf{x}) = \{\mathbf{y} \in \mathcal{B}^n : \mathcal{D}(\mathbf{x}, \mathbf{y}) \leq r\}. \tag{5.2}$$

We refer to Hamming spheres and Hamming balls as *Hamming regions*.

5.1.2 Mean Values over Hamming Spheres

This subsection reviews an efficient method introduced by Sutton *et al.* [SWH12] to compute mean values over Hamming Regions from Walsh coefficients.

Suppose $Q = \{\mathbf{i} | \mathbf{i} \in \mathcal{B}^n,\ w_\mathbf{i} \neq 0\}$, then Q is the set of n-bit strings whose corresponding Walsh coefficients are non-zero. The mean values over a sphere of radius r can be computed using equation (5.3).

$$\overline{\mathsf{HS}^{(r)}}(\mathbf{x}) = \binom{n}{r}^{-1} \sum_{j=0}^{k} \gamma_j^{(r)} \varphi_{[j]}(\mathbf{x}) \quad \text{Equation 25 in [SWH12]}$$

$$= \binom{n}{r}^{-1} \sum_{j=0}^{k} \gamma_j^{(r)} \sum_{\mathbf{i} \in Q : \mathbf{bc}(\mathbf{i}) = j} w_{\mathbf{i}} \psi_{\mathbf{i}}(\mathbf{x}) \quad \text{Equation 9 in [SWH12]} \tag{5.3}$$

where

$$\gamma_j^{(r)} = \mathcal{K}_r(j, n) = \sum_{i=0}^{r} \binom{j}{i} \binom{n-j}{r-i} (-1)^i. \tag{5.4}$$

$\mathcal{K}_r(p, n)$ is the Krawtchouk polynomials [Kra29].

5.1.3 Walsh-LS with Surrogate Fitness

We consider how to execute Walsh-LS using surrogate fitness function. We first define the analogy of $\mathbf{S}(\mathbf{x})$ for Walsh-LS in equation (5.5).

$$S_p(\overline{\mathsf{HS}^{(r)}}(\mathbf{x})) = \binom{n}{r}^{-1} \sum_{j=0}^{k} \gamma_j^{(r)} \sum_{\mathbf{i} \in Q : \mathbf{bc}(\mathbf{i}) = j \wedge i_p = 1} w'_{\mathbf{i}}(x) \tag{5.5}$$

After flipping a bit p, the difference in the surrogate fitness can be computed using equation (5.6).

$$\begin{aligned}
&\overline{\mathsf{HS}^{(r)}}(\mathbf{x}^{(p)}) - \overline{\mathsf{HS}^{(r)}}(\mathbf{x}) \\
&= \binom{n}{r}^{-1} \sum_{j=0}^{k} \gamma_j^{(r)} \sum_{\mathbf{i} \in Q : \mathbf{bc}(i) = j} w_{\mathbf{i}} (\psi_{\mathbf{i}}(\mathbf{x}^{(p)}) - \psi_{\mathbf{i}}(\mathbf{x})) \\
&= \binom{n}{r}^{-1} \sum_{j=0}^{k} \gamma_j^{(r)} \sum_{\mathbf{i} \in Q : \mathbf{bc}(i) = j \wedge i_p = 1} w_{\mathbf{i}} (-2\psi_{\mathbf{i}}(\mathbf{x})) \\
&= -2 S_p(\overline{\mathsf{HS}^{(r)}}(\mathbf{x}))
\end{aligned} \tag{5.6}$$

Similar to equation (3.3) with the original fitness, $S_p(\overline{\mathsf{HS}^{(r)}}(\mathbf{x}))$ can be used as a proxy for the new surrogate fitness $\overline{\mathsf{HS}^{(r)}}(\mathbf{x}^{(p)})$, since $\overline{\mathsf{HS}^{(r)}}(\mathbf{x})$ is constant as p varies. $\mathbf{S}(\overline{\mathsf{HS}^{(r)}}(\mathbf{x}))$ is a proxy vector that reflects the changes in surrogate fitness when any one of n possible bits is flipped.

5.1.4 Update Proxy after a Single Bit Flip

Walsh-LS maintains three data structures, namely vector $\mathbf{w}'(\mathbf{x})$, vector $\mathbf{S}(\mathbf{x})$ and list $\mathbf{Impr}(\mathbf{x})$. With the surrogate fitness, only the vector $\mathbf{S}(\mathbf{x})$ is replaced with $\mathbf{S}(\overline{\mathsf{HS}^{(r)}(\mathbf{x})})$, while $\mathbf{w}'(\mathbf{x})$ and $\mathbf{Impr}(\mathbf{x})$ remain the same. We consider how to efficiently update the proxy vector $\mathbf{S}(\overline{\mathsf{HS}^{(r)}(\mathbf{x})})$ after flipping a bit.

Lemma 19. *Suppose $\mathbf{x} \in \mathcal{B}^n$ and $\mathbf{i} \in Q$. After flipping a bit q, the vector $\mathbf{S}(\overline{\mathsf{HS}^{(r)}(\mathbf{x})})$ can be computed using*

$$\begin{aligned}
&\forall p \in [1,n], \\
&S_p(\overline{\mathsf{HS}^{(r)}(\mathbf{x}^{(q)})}) \\
&= S_p(\overline{\mathsf{HS}^{(r)}(\mathbf{x})}) - 2\binom{n}{r}^{-1} \sum_{j=0}^{k} \gamma_j^{(r)} \sum_{\mathbf{i}\in Q:\mathbf{bc}(i)=j\wedge i_p=1\wedge i_q=1} w'_{\mathbf{i}}(\mathbf{x})
\end{aligned}$$

Proof. $\forall p \in [1,n]$, the difference between $S_p(\overline{\mathsf{HS}^{(r)}(\mathbf{x}^{(q)})})$ and $S_p(\overline{\mathsf{HS}^{(r)}(\mathbf{x})})$ can be calculated as follows:

$$\begin{aligned}
&S_p(\overline{\mathsf{HS}^{(r)}(\mathbf{x}^{(q)})}) - S_p(\overline{\mathsf{HS}^{(r)}(\mathbf{x})}) \\
&= \binom{n}{r}^{-1} \sum_{j=0}^{k} \gamma_j^{(r)} \sum_{\mathbf{i}\in Q:\mathbf{bc}(\mathbf{i})=j\wedge i_p=1} w_{\mathbf{i}}(x)(\psi_{\mathbf{i}}(\mathbf{x}^{(q)}) - \psi_{\mathbf{i}}(\mathbf{x})) \\
&= -2\binom{n}{r}^{-1} \sum_{j=0}^{k} \gamma_j^{(r)} \sum_{\mathbf{i}\in Q:\mathbf{bc}(i)=j\wedge i_p=1\wedge i_q=1} w'_{\mathbf{i}}(\mathbf{x}) \qquad (5.7)
\end{aligned}$$

□

Denote exact-Walsh-LS with the mean values over Hamming sphere of radius r as surrogate fitness as Walsh-LS-HS$^{(r)}$. We now demonstrate that Walsh-LS-HS$^{(r)}$ can be executed as fast as exact-Walsh-LS. Comparing equation (5.7) with lemma 5, we observe that the difference lies in the factor of Walsh term $w'_{\mathbf{i}}(\mathbf{x})$. Equation (5.8) how the extra factors from equation (5.7) are absorbed into Walsh terms.

$$wHS_{\mathbf{i}}^{(r)}(\mathbf{x}) = \binom{n}{r}^{-1} \gamma_{\mathbf{bc(i)}}^{(r)} w'_{\mathbf{i}}(\mathbf{x}) \qquad (5.8)$$

After defining $wHS_{\mathbf{i}}^{(r)}$, equation (5.7) can be rewritten as

$$S_p(\overline{\mathsf{HS}^{(r)}(\mathbf{x}^{(q)})}) - S_p(\overline{\mathsf{HS}^{(r)}(\mathbf{x})}) = -2 \sum_{\mathbf{i}\in Q:\mathbf{bc}(i)=j\wedge i_p=1\wedge i_q=1} wHS_{\mathbf{i}}^{(r)}(\mathbf{x}) \tag{5.9}$$

This suggests that except for substituting $w'_{\mathbf{i}}(\mathbf{x})$ in Walsh-LS with $wHS_{\mathbf{i}}^{(r)}(\mathbf{x})$ in the initialization stage, Walsh-LS-HS$^{(r)}$ can be executed in exactly the same manner as Walsh-LS. The runtime complexity of Walsh-LS-HS$^{(r)}$ is also therefore the same as Walsh-LS.

Our study extends Whitley and Chen's work [WC12]. They provide an efficient method for computing the mean value over a special Hamming sphere, HS$^{(1)}$, and use it as surrogate fitness. However, they execute Walsh-LS-HS$^{(1)}$ with surrogate fitness at the cost of doubling the runtime of exact-Walsh-LS with original fitness. In this chapter, we extend their approach to Hamming spheres of arbitrary radii, and we also show that using mean values over Hamming spheres of arbitrary fitness can be achieved *without any extra cost* in runtime.

5.1.5 Mean Values over Hamming Balls as Surrogate Fitness

A ball of radius r can be viewed as a superposition of r spheres of radii from 1 to r. We now show that the previously introduced techniques in this section also apply to Walsh-LS with mean values over Hamming balls of radius r as surrogate fitness (denoted as Walsh-LS-HB$^{(r)}$).

The mean values over a Hamming ball of radius r can be computed using equation (5.10).

$$\begin{aligned}\overline{\mathsf{HB}^{(r)}(\mathbf{x})} &= \Big(\sum_{s=0}^{r}\binom{n}{s}\Big)^{-1}\sum_{s=0}^{r}\sum_{j=0}^{k}\gamma_j^{(s)}\varphi_{[j]}(\mathbf{x}) \quad \text{Equation 25 in [SWH12]} \\ &= \Big(\sum_{s=0}^{r}\binom{n}{s}\Big)^{-1}\sum_{s=0}^{r}\sum_{j=0}^{k}\gamma_j^{(s)}\sum_{\mathbf{i}\in Q:\mathbf{bc}(\mathbf{i})=j} w_{\mathbf{i}}\psi_{\mathbf{i}}(\mathbf{x}) \quad \text{Equation 9 in [SWH12]} \end{aligned} \tag{5.10}$$

Similar to the definition of $wHS_{\mathbf{i}}^{(r)}(\mathbf{x})$, we can define $^{(r)}_{\mathbf{i}}(\mathbf{x})$ in equation (5.11).

$$wHB_{\mathbf{i}}^{(r)}(\mathbf{x}) = \Big(\sum_{s=0}^{r}\binom{n}{s}\Big)^{-1}\sum_{s=0}^{r}\gamma_{\mathbf{bc(i)}}^{(s)}w'_{\mathbf{i}}(\mathbf{x}) \tag{5.11}$$

Again, after initializing $wHB_{\mathbf{i}}^{(r)}(\mathbf{x})$, Walsh-LS-HB$^{(r)}$ can be executed as efficiently as Walsh-LS. While executing Walsh-LS-HB$^{(r)}$, $wHB_{\mathbf{i}}^{(r)}(\mathbf{x})$ is treated in the same way as $w'_{\mathbf{i}}(\mathbf{x})$.

5.2 Empirical Studies

In this section, we first show that searching in the surrogate fitness space can improve the ability of Walsh-LS to find good solutions. We next empirically verify that using surrogate fitness over Hamming spheres comes at no additional cost of running time. Some in-depth empirical analysis is provided at the end of this section to provide insight on why the surrogate fitness can help discover better solutions.

Experiment 11 is conducted to evaluate the efficiency and effectiveness of Walsh-LS-HS$^{(r)}$. Exact-Walsh-LS is essentially Walsh-LS-HS$^{(0)}$, since Hamming Sphere with $r = 0$ shrinks to a single point. Also, Walsh-LS-HS$^{(r)}$ with $r > 0$ works entirely on the surrogate fitness, which means it has no access to the real fitness. We evaluate the best-so-far solution obtained in the surrogate fitness space using the original fitness function at the end of Walsh-LS-HS$^{(r)}$.

Experiment 11. We run Walsh-LS-HS$^{(r)}$ with $r = \{0, 1, 2, 3\}$ on both uniform and non-uniform random NK-landscapes and NKq-landscapes ($q = 2$) instances with $n = \{20, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500\}$ and $K = \{2, 4\}$. MaxBitFlip is set to $100,000$ for each algorithm on every instance. The trials for each configuration are repeated 10 times independently. Best-so-far solutions, their relative real fitnesses, the number of restarts, and overall runtime are recorded.

5.2.1 Solution Quality

We compare the real fitnesses returned by Walsh-LS-HS$^{(r)}$ with $r = \{0, 1, 2, 3\}$ on *uniform NK random instances* in table 5.1 and table 5.2. We are mainly interested in investigating the impact of using surrogate fitness in search. The comparison between exact-Walsh-LS and Walsh-LS-HS$^{(1)}$ highlights the difference, therefore p-value related to Wilcoxon rank-sum test comparing solutions by exact-Walsh-LS and Walsh-LS-HS$^{(1)}$ are presented in the last

columns of table 5.1 and table 5.2. As there are 88 possible comparisons[11] to perform, the significant level is adjusted to $\frac{0.05}{88} = 0.000568$ according to Bonferroni adjustment [Dun61].

As suggested by table 5.1 and table 5.2, applying the surrogate fitness makes no statistically significant difference in solution quality. It indicates that on the rugged uniform NK-landscapes with little inherent structure, searching in the surrogate fitness space makes no difference in solution quality.

Table 5.1: Evaluations of solutions (maximization) after 100,000 moves by Walsh-LS-HS on *uniform NK random instances with* $K = 2$. Mean values and standard deviations over 10 independent runs are reported. P-values calculated from Wilcoxon rank-sum tests comparing solutions by exact-Walsh-LS and Walsh-LS-HS$^{(1)}$ are also presented. Statistical significantly better solutions and the related p-values are marked in **bold** with Bonferroni adjusted significance level $\frac{0.05}{88} = 0.000568$.

n	exact-Walsh-LS	Walsh-LS-HS$^{(1)}$	Walsh-LS-HS$^{(2)}$	Walsh-LS-HS$^{(3)}$	p-value
20	14.362±0.000	14.362±0.000	14.362±0.000	14.041±0.000	NaN
50	38.648±0.000	38.648±0.000	38.648±0.000	38.587±0.000	NaN
100	73.073±0.095	73.033±0.160	72.980±0.179	73.103±0.079	0.9075
150	111.987±0.216	112.249±0.180	111.962±0.204	112.102±0.245	0.0140
200	147.879±0.369	147.837±0.166	148.226±0.490	147.920±0.213	0.6842
250	184.711±0.494	184.672±0.423	184.642±0.476	184.652±0.406	1.0000
300	223.021±0.648	222.960±0.349	223.252±0.740	223.407±0.979	0.9705
350	262.395±0.569	262.614±0.698	262.193±0.626	262.678±0.454	0.4813
400	296.195±0.497	296.278±1.172	296.318±0.821	295.991±0.613	0.8534
450	328.425±0.597	328.679±0.667	328.945±1.113	328.336±0.568	0.3930
500	368.116±0.705	367.590±0.843	367.449±0.831	367.875±1.128	0.1051

Compared to NK-landscapes, NKq-landscapes are neutral landscapes that are more similar to SAT instances. Evaluations of solutions by Walsh-LS-HS on *uniform NKq-landscapes* *($q = 2$)* are presented in table 5.3 and table 5.4. Walsh-LS-HS$^{(1)}$ consistently finds statistically significant better solutions than exact-Walsh-LS on sufficiently large instances (starting from the ones with $n = 100$).

[11] {11 levels for n} × {2 levels for "uniform" versus "non-uniform"} × {2 levels for "NK" versus "NKQ"} × 2 levels for K

Table 5.2: Evaluations of solutions (maximization) after 100,000 moves by Walsh-LS-HS on *uniform NK random instances with* $K = 4$. Mean values and standard deviations over 10 independent runs are reported. P-values calculated from Wilcoxon rank-sum tests comparing solutions by exact-Walsh-LS and Walsh-LS-HS$^{(1)}$ are also presented. Statistical significantly better solutions and the related p-values are marked in **bold** with Bonferroni adjusted significance level $\frac{0.05}{88} = 0.000568$.

n	exact-Walsh-LS	Walsh-LS-HS$^{(1)}$	Walsh-LS-HS$^{(2)}$	Walsh-LS-HS$^{(3)}$	p-value
20	15.427±0.000	15.427±0.000	15.204±0.000	15.204±0.000	NaN
50	39.382±0.126	39.409±0.104	39.112±0.000	39.112±0.000	0.6701
100	77.812±0.425	77.899±0.477	78.199±0.525	78.274±0.608	0.4359
150	114.919±0.566	114.730±0.480	114.832±0.638	115.326±0.762	0.4359
200	152.029±0.949	151.899±0.616	152.093±0.690	152.380±0.747	1.0000
250	189.661±0.748	189.368±0.463	189.589±0.779	190.090±1.137	0.2475
300	225.019±0.928	224.827±0.666	225.209±0.854	225.505±0.952	0.9118
350	262.196±1.354	263.126±1.129	262.372±0.826	262.922±1.069	0.1655
400	297.577±1.144	297.394±1.027	298.219±1.086	297.802±1.055	0.6842
450	335.036±0.507	335.671±0.617	336.055±1.071	336.941±1.940	0.0147
500	374.279±1.150	374.172±1.644	374.855±2.125	373.977±1.273	0.4813

We now study how the previously observed trends could change on non-uniform instances. Recall that the "non-uniform" distribution comes from the variables distribution over subfunctions. Non-uniform instances are generated using algorithm 6.

Evaluations of solutions on *non-uniform NK-landscapes instances* are summarized in table 5.5 and table 5.6. Applying the surrogate fitness appears to statistically significantly deteriorate the performance on small instances. While on larger instances (where $n \geq 200$), utilizing surrogate fitness makes no statistically significant difference in solution quality.

We next study the impact of variable distributions over subfunctions on non-uniform NKq-landscapes. Evaluations of solutions on *non-uniform NKq (q=2) landscapes instances* are summarized in table 5.7 and table 5.8. Walsh-LS-HS$^{(1)}$ finds statistically significant better solutions than exact-Walsh-LS on some instances, while on other instances Walsh-LS-HS$^{(1)}$ achieves comparable performance. Compared to the results on uniform NKq-landscapes, however, statistically significant better solutions are less commonly found. On uniform random NKq-landscapes, Walsh-LS-HS$^{(1)}$ discovers statistically significantly better solutions

Table 5.3: Evaluations of solutions (maximization) after 100,000 moves by Walsh-LS-HS on *uniform NKq (q=2) random instances with* $K = 2$. Mean values and standard deviations over 10 independent runs are reported. P-values calculated from Wilcoxon rank-sum tests comparing solutions by exact-Walsh-LS and Walsh-LS-HS$^{(1)}$ are also presented. Statistical significantly better solutions and the related p-values are marked in **bold** with Bonferroni adjusted significance level $\frac{0.05}{88} = 0.000568$.

n	exact-Walsh-LS	Walsh-LS-HS$^{(1)}$	Walsh-LS-HS$^{(2)}$	Walsh-LS-HS$^{(3)}$	p-value
20	18.000±0.000	18.000±0.000	18.000±0.000	17.000±0.000	NaN
50	45.000±0.000	45.000±0.000	45.000±0.000	45.000±0.000	NaN
100	93.000±0.000	93.900±0.316	94.000±0.000	93.900±0.316	$<$ **0.0001**
150	136.800±1.033	140.600±0.516	140.900±0.316	140.800±0.422	**0.0001**
200	180.700±1.160	186.200±0.789	187.000±0.943	186.600±0.843	**0.0001**
250	224.100±1.595	231.300±0.949	231.800±0.632	232.300±0.675	**0.0001**
300	268.500±0.850	277.300±1.059	277.200±0.632	277.300±1.160	**0.0001**
350	310.800±1.874	322.300±0.823	323.800±0.632	323.900±0.876	**0.0001**
400	355.300±1.829	372.500±0.850	373.700±1.160	373.700±1.337	**0.0002**
450	394.300±1.567	410.600±0.843	411.800±1.476	411.100±1.287	**0.0002**
500	444.800±1.874	462.500±1.434	464.100±1.370	463.700±1.636	**0.0002**

in 18 out of 22 cases, whereas it only returns statistically significantly better solutions on 9 out of 22 non-uniform NKq instances. The non-uniform variable distribution over subfunctions seems to make applying surrogate fitness less advantageous.

Table 5.4: Evaluations of solutions (maximization) after 100,000 moves by Walsh-LS-HS on *uniform NKq (q=2) random instances with* $K = 4$. Mean values and standard deviations over 10 independent runs are reported. P-values calculated from Wilcoxon rank-sum tests comparing solutions by exact-Walsh-LS and Walsh-LS-HS$^{(1)}$ are also presented. Statistical significantly better solutions and the related p-values are marked in **bold** with Bonferroni adjusted significance level $\frac{0.05}{88} = 0.000568$.

n	exact-Walsh-LS	Walsh-LS-HS$^{(1)}$	Walsh-LS-HS$^{(2)}$	Walsh-LS-HS$^{(3)}$	p-value
20	10.000±0.000	10.000±0.000	10.000±0.000	10.000±0.000	NaN
50	27.000±0.000	27.000±0.000	27.000±0.000	27.000±0.000	NaN
100	52.100±0.994	54.700±0.483	54.800±0.422	54.900±0.316	**0.0002**
150	77.700±1.829	83.300±0.483	83.600±0.516	84.400±0.843	**0.0001**
200	98.800±2.044	107.300±1.418	107.900±1.370	107.600±1.075	**0.0002**
250	121.900±1.370	133.900±1.197	134.900±0.994	135.100±1.370	**0.0002**
300	143.600±2.011	161.300±1.703	161.900±1.792	161.600±1.506	**0.0002**
350	165.200±2.394	187.400±2.413	187.000±1.491	187.200±2.098	**0.0002**
400	187.600±2.836	210.200±1.751	211.900±2.025	211.600±1.350	**0.0002**
450	210.600±2.757	236.900±0.738	238.700±1.889	239.400±1.578	**0.0001**
500	228.900±3.695	261.300±2.058	260.800±1.687	263.000±1.886	**0.0002**

Table 5.5: Evaluations of solutions (maximization) after 100,000 moves by Walsh-LS-HS on *non-uniform NK random instances with* $K = 2$. Mean values and standard deviations over 10 independent runs are reported. P-values calculated from Wilcoxon rank-sum tests comparing solutions by exact-Walsh-LS and Walsh-LS-HS$^{(1)}$ are also presented. Statistical significantly better solutions and the related p-values are marked in **bold** with Bonferroni adjusted significance level $\frac{0.05}{88} = 0.000568$.

n	exact-Walsh-LS	Walsh-LS-HS$^{(1)}$	Walsh-LS-HS$^{(2)}$	Walsh-LS-HS$^{(3)}$	p-value
20	14.172±0.000	14.126±0.000	14.126±0.000	14.126±0.000	$<$ **0.0001**
50	35.777±0.000	35.771±0.000	35.771±0.000	35.654±0.000	$<$ **0.0001**
100	72.727±0.000	72.727±0.000	72.727±0.000	72.727±0.000	NaN
150	107.462±0.000	107.461±0.000	107.461±0.000	107.461±0.000	$<$ **0.0001**
200	143.627±0.159	143.653±0.137	143.677±0.177	143.625±0.131	0.4388
250	175.841±0.341	175.876±0.177	175.923±0.211	175.804±0.267	0.8534
300	214.733±0.396	214.926±0.437	214.655±0.403	214.569±0.354	0.4495
350	249.842±0.584	249.938±0.530	249.808±0.636	250.062±0.672	0.7959
400	286.945±0.312	286.906±0.231	287.029±0.757	286.874±0.372	0.8534
450	319.910±0.511	319.766±0.471	319.963±0.527	319.780±0.357	0.4813
500	352.241±1.134	352.110±0.938	352.135±1.188	352.423±0.882	0.9118

Table 5.6: Evaluations of solutions (maximization) after 100,000 moves by Walsh-LS-HS on *non-uniform NK random instances with* $K = 4$. Mean values and standard deviations over 10 independent runs are reported. P-values calculated from Wilcoxon rank-sum tests comparing solutions by exact-Walsh-LS and Walsh-LS-HS$^{(1)}$ are also presented. Statistical significantly better solutions and the related p-values are marked in **bold** with Bonferroni adjusted significance level $\frac{0.05}{88} = 0.000568$.

n	exact-Walsh-LS	Walsh-LS-HS$^{(1)}$	Walsh-LS-HS$^{(2)}$	Walsh-LS-HS$^{(3)}$	p-value
20	14.832±0.000	14.771±0.000	14.771±0.000	14.361±0.000	< **0.0001**
50	37.015±0.000	37.006±0.000	37.006±0.000	37.006±0.000	< **0.0001**
100	75.400±0.000	75.208±0.329	75.352±0.147	75.247±0.271	< **0.0001**
150	109.983±0.139	109.927±0.136	109.973±0.195	109.950±0.169	0.3836
200	146.076±0.413	146.262±0.360	146.218±0.438	146.405±0.253	0.3829
250	183.456±0.716	183.624±0.806	183.329±0.427	183.346±0.545	0.6305
300	216.599±0.504	216.122±0.617	216.009±0.868	216.263±0.568	0.0524
350	251.822±0.986	252.452±0.888	252.836±1.573	252.415±1.210	0.2176
400	285.510±0.858	286.734±1.367	286.389±1.508	286.129±1.060	0.0355
450	321.627±1.469	321.262±1.375	320.755±1.128	320.943±1.556	0.7394
500	354.690±1.073	354.855±1.325	353.973±0.939	354.461±1.731	0.7959

Table 5.7: Evaluations of solutions (maximization) after 100,000 moves by Walsh-LS-HS on *non-uniform NKq (q=2) random instances with* $K = 2$. Mean values and standard deviations over 10 independent runs are reported. P-values calculated from Wilcoxon rank-sum tests comparing solutions by exact-Walsh-LS and Walsh-LS-HS$^{(1)}$ are also presented. Statistical significantly better solutions and the related p-values are marked in **bold** with Bonferroni adjusted significance level $\frac{0.05}{88} = 0.000568$.

n	exact-Walsh-LS	Walsh-LS-HS$^{(1)}$	Walsh-LS-HS$^{(2)}$	Walsh-LS-HS$^{(3)}$	p-value
20	18.000±0.000	18.000±0.000	18.000±0.000	18.000±0.000	NaN
50	46.000±0.000	46.000±0.000	46.000±0.000	46.000±0.000	NaN
100	91.000±0.000	91.000±0.000	91.000±0.000	91.000±0.000	NaN
150	135.100±0.316	136.000±0.000	136.000±0.000	136.000±0.000	< **0.0001**
200	175.600±0.516	176.000±0.000	176.000±0.000	176.000±0.000	0.0336
250	214.600±0.843	216.000±0.000	215.800±0.422	216.000±0.000	**0.0002**
300	261.900±0.738	262.700±0.483	262.900±0.316	262.800±0.422	0.0170
350	297.900±0.738	299.000±0.667	299.300±0.823	299.000±0.471	0.0057
400	347.300±1.494	348.500±0.707	349.000±0.943	349.100±0.738	0.0276
450	387.300±0.675	389.900±0.738	389.700±0.823	389.800±1.033	**0.0001**
500	430.600±0.966	432.600±0.699	432.600±0.699	432.600±0.699	**0.0002**

Table 5.8: Evaluations of solutions (maximization) after 100,000 moves by Walsh-LS-HS on *non-uniform NKq (q=2) random instances with* $K = 4$. Mean values and standard deviations over 10 independent runs are reported. P-values calculated from Wilcoxon rank-sum tests comparing solutions by exact-Walsh-LS and Walsh-LS-HS$^{(1)}$ are also presented. Statistical significantly better solutions and the related p-values are marked in **bold** with Bonferroni adjusted significance level $\frac{0.05}{88} = 0.000568$.

n	exact-Walsh-LS	Walsh-LS-HS$^{(1)}$	Walsh-LS-HS$^{(2)}$	Walsh-LS-HS$^{(3)}$	p-value
20	10.000±0.000	10.000±0.000	10.000±0.000	10.000±0.000	NaN
50	24.000±0.000	24.000±0.000	24.000±0.000	24.000±0.000	NaN
100	43.000±0.000	43.000±0.000	43.000±0.000	43.000±0.000	NaN
150	56.900±0.316	57.000±0.000	57.000±0.000	57.000±0.000	0.3681
200	81.300±0.823	82.000±0.000	82.000±0.000	82.000±0.000	0.0146
250	103.000±1.333	105.000±0.000	105.000±0.000	104.900±0.316	$<$ **0.0001**
300	116.800±1.033	119.000±0.000	119.000±0.000	119.000±0.000	$<$ **0.0001**
350	128.600±1.506	132.700±0.483	133.000±0.000	132.700±0.675	**0.0001**
400	142.800±0.789	148.100±1.101	148.300±0.949	148.100±0.876	**0.0001**
450	168.900±3.348	174.200±1.751	175.200±0.919	174.200±1.398	0.0016
500	178.700±2.452	185.900±1.969	183.900±2.378	184.800±1.549	**0.0002**

5.2.2 Runtime

We verify the previous claim that applying surrogate fitness incurs no extra cost of runtime. The runtime data collected from experiment 11 are presented in table 5.9 to table 5.16. Across all tested instances, surrogate fitness appears to be applied to Walsh-LS with no additional cost in runtime. In fact, exact-Walsh-LS with the original fitness takes even more time to complete 100000 moves than Walsh-LS-HS$^{(r)}$ with $r > 0$ on most tested cases. This suggests the performance gain in solution quality can be achieved at little extra runtime cost.

Table 5.9: Overall runtimes in seconds on *uniform NK instances with* $K = 2$. The median runtimes over 10 runs are presented, and the numbers after the "±" symbol are the corresponding standard deviations.

n	exact-Walsh-LS	Walsh-LS-HS$^{(1)}$	Walsh-LS-HS$^{(2)}$	Walsh-LS-HS$^{(3)}$
20	2.955±0.015	2.859±0.007	2.707±0.007	2.758±0.014
50	3.040±0.011	2.990±0.015	2.982±0.008	2.937±0.005
100	3.342±0.017	3.270±0.026	3.309±0.014	3.285±0.013
150	3.436±0.037	3.480±0.027	3.519±0.030	3.481±0.035
200	3.688±0.014	3.683±0.015	3.674±0.020	3.644±0.012
250	3.916±0.023	3.955±0.029	3.960±0.005	3.919±0.025
300	4.154±0.021	4.128±0.015	4.124±0.028	4.148±0.016
350	4.480±0.039	4.514±0.032	4.508±0.021	4.434±0.036
400	4.715±0.035	4.697±0.025	4.696±0.031	4.703±0.034
450	4.910±0.044	5.019±0.041	4.893±0.032	4.880±0.031
500	5.110±0.020	5.062±0.036	5.245±0.074	5.171±0.040

Table 5.10: Overall runtimes in seconds on *uniform NK instances with* $K = 4$. The mean runtimes over 10 runs are presented, and the numbers after the "±" symbol are the corresponding standard deviations.

n	exact-Walsh-LS	Walsh-LS-HS$^{(1)}$	Walsh-LS-HS$^{(2)}$	Walsh-LS-HS$^{(3)}$
20	6.514±0.056	6.123±0.031	5.815±0.031	5.611±0.026
50	8.182±0.076	8.092±0.062	7.798±0.091	7.726±0.053
100	8.887±0.099	9.075±0.270	9.066±0.107	8.754±0.133
150	9.264±0.228	9.186±0.092	9.160±0.113	9.071±0.169
200	9.612±0.188	10.004±0.270	9.855±0.229	9.680±0.163
250	10.374±0.232	10.457±0.341	10.524±0.200	10.448±0.228
300	11.302±0.206	11.605±0.269	11.471±0.318	11.738±0.427
350	13.185±0.348	12.911±0.270	13.061±0.341	12.862±0.187
400	14.158±0.414	14.036±0.253	14.107±0.273	14.122±0.355
450	15.129±0.441	14.849±0.221	14.970±0.304	15.084±0.408
500	15.819±0.344	15.514±0.172	15.678±0.250	15.547±0.164

Table 5.11: Overall runtimes in seconds on *uniform NKq (q=2) instances with* $K = 2$. The mean runtimes over 10 runs are presented, and the numbers after the "±" symbol are the corresponding standard deviations.

n	exact-Walsh-LS	Walsh-LS-HS$^{(1)}$	Walsh-LS-HS$^{(2)}$	Walsh-LS-HS$^{(3)}$
20	3.597±0.018	2.797±0.018	2.816±0.007	2.723±0.010
50	3.317±0.011	2.660±0.012	2.664±0.015	2.634±0.008
100	3.328±0.022	2.771±0.012	2.700±0.008	2.693±0.022
150	3.649±0.021	3.049±0.032	3.015±0.034	3.056±0.028
200	3.935±0.013	3.301±0.006	3.259±0.009	3.238±0.007
250	4.127±0.013	3.484±0.009	3.465±0.009	3.465±0.006
300	4.122±0.026	3.656±0.011	3.679±0.011	3.639±0.009
350	4.492±0.020	3.928±0.012	3.959±0.013	3.949±0.011
400	4.648±0.017	4.137±0.014	4.116±0.015	4.132±0.013
450	4.971±0.007	4.392±0.014	4.368±0.014	4.443±0.009
500	5.148±0.015	4.632±0.020	4.583±0.079	4.575±0.084

Table 5.12: Overall runtimes in seconds on *uniform NKq (q=2) instances with* $K = 4$. The mean runtimes over 10 runs are presented, and the numbers after the "±" symbol are the corresponding standard deviations.

n	exact-Walsh-LS	Walsh-LS-HS$^{(1)}$	Walsh-LS-HS$^{(2)}$	Walsh-LS-HS$^{(3)}$
20	6.009±0.068	4.974±0.024	4.712±0.023	4.496±0.026
50	7.953±0.051	6.508±0.038	6.197±0.075	6.448±0.036
100	8.911±0.078	7.274±0.044	6.995±0.050	6.905±0.064
150	9.228±0.182	7.103±0.060	6.956±0.025	6.837±0.102
200	9.514±0.123	7.680±0.072	7.505±0.068	7.370±0.049
250	9.951±0.109	7.969±0.164	7.604±0.134	7.536±0.086
300	10.473±0.117	7.974±0.100	8.100±0.076	7.942±0.178
350	10.908±0.163	8.529±0.099	8.468±0.235	8.449±0.137
400	11.954±0.128	9.065±0.224	8.884±0.132	8.886±0.136
450	12.541±0.309	9.883±0.248	9.237±0.176	9.268±0.134
500	13.095±0.260	10.380±0.294	10.109±0.263	10.144±0.297

Table 5.13: Overall runtimes in seconds on *non-uniform NK instances with* $K = 2$. The mean runtimes over 10 runs are presented, and the numbers after the "±" symbol are the corresponding standard deviations.

n	exact-Walsh-LS	Walsh-LS-HS$^{(1)}$	Walsh-LS-HS$^{(2)}$	Walsh-LS-HS$^{(3)}$
20	2.674±0.007	2.654±0.007	2.649±0.010	2.531±0.018
50	2.690±0.007	2.641±0.009	2.649±0.012	2.620±0.015
100	2.870±0.012	2.886±0.015	2.838±0.017	2.899±0.013
150	3.132±0.028	3.150±0.034	3.168±0.026	3.167±0.033
200	3.373±0.006	3.373±0.009	3.396±0.012	3.377±0.012
250	3.561±0.011	3.568±0.016	3.532±0.004	3.584±0.011
300	3.809±0.013	3.780±0.012	3.811±0.010	3.829±0.013
350	4.202±0.019	4.277±0.009	4.164±0.018	4.219±0.035
400	4.428±0.026	4.417±0.035	4.422±0.022	4.449±0.026
450	4.621±0.048	4.605±0.037	4.656±0.026	4.637±0.034
500	4.788±0.034	4.853±0.019	4.877±0.081	4.850±0.015

Table 5.14: Overall runtimes in seconds on *non-uniform NK instances with* $K = 4$. The mean runtimes over 10 runs are presented, and the numbers after the "±" symbol are the corresponding standard deviations.

n	exact-Walsh-LS	Walsh-LS-HS$^{(1)}$	Walsh-LS-HS$^{(2)}$	Walsh-LS-HS$^{(3)}$
20	4.804±0.039	4.748±0.025	4.408±0.025	4.180±0.017
50	4.951±0.038	4.851±0.062	4.923±0.040	4.839±0.020
100	5.366±0.039	5.309±0.022	5.203±0.112	5.287±0.105
150	5.579±0.167	5.545±0.156	5.557±0.147	5.612±0.095
200	5.732±0.134	5.764±0.078	5.711±0.068	5.750±0.100
250	5.904±0.056	6.241±0.092	6.159±0.160	6.097±0.146
300	6.430±0.139	6.655±0.211	6.592±0.075	6.968±0.091
350	7.139±0.092	7.212±0.155	7.445±0.131	7.251±0.078
400	7.854±0.061	8.247±0.131	7.890±0.105	8.081±0.172
450	8.425±0.145	8.370±0.112	8.459±0.099	8.488±0.099
500	9.344±0.181	9.306±0.188	9.325±0.225	9.350±0.199

Table 5.15: Overall runtimes in seconds on *non-uniform NKq (q=2) instances with* $K = 2$. The mean runtimes over 10 runs are presented, and the numbers after the "±" symbol are the corresponding standard deviations.

n	exact-Walsh-LS	Walsh-LS-HS$^{(1)}$	Walsh-LS-HS$^{(2)}$	Walsh-LS-HS$^{(3)}$
20	2.755±0.017	2.198±0.013	2.278±0.013	2.228±0.018
50	3.032±0.021	2.471±0.019	2.471±0.021	2.501±0.017
100	3.252±0.007	2.616±0.013	2.630±0.007	2.661±0.010
150	3.314±0.010	2.856±0.038	2.824±0.032	2.849±0.010
200	3.510±0.015	3.046±0.009	3.035±0.009	3.041±0.006
250	3.813±0.012	3.336±0.015	3.236±0.006	3.230±0.009
300	3.955±0.021	3.599±0.006	3.571±0.007	3.521±0.016
350	4.165±0.014	3.749±0.010	3.704±0.011	3.716±0.008
400	4.463±0.012	4.152±0.012	4.123±0.015	4.060±0.009
450	4.567±0.016	4.314±0.010	4.317±0.016	4.264±0.013
500	4.675±0.014	4.533±0.030	4.525±0.021	4.532±0.016

Table 5.16: Overall runtimes in seconds on *non-uniform NKq (q=2) instances with* $K = 4$. The mean runtimes over 10 runs are presented, and the numbers after the "±" symbol are the corresponding standard deviations.

n	exact-Walsh-LS	Walsh-LS-HS$^{(1)}$	Walsh-LS-HS$^{(2)}$	Walsh-LS-HS$^{(3)}$
20	4.536±0.022	4.100±0.019	3.898±0.025	3.816±0.023
50	5.518±0.094	4.632±0.039	4.105±0.023	4.083±0.026
100	6.267±0.204	4.808±0.049	4.537±0.033	4.512±0.035
150	7.535±0.222	5.243±0.031	4.961±0.039	4.909±0.044
200	7.358±0.191	5.391±0.098	4.989±0.069	4.985±0.025
250	7.987±0.160	5.560±0.069	5.416±0.036	5.347±0.042
300	8.723±0.157	6.006±0.071	5.648±0.117	5.492±0.051
350	9.104±0.255	6.411±0.120	6.155±0.056	6.022±0.123
400	9.769±0.231	6.632±0.067	6.495±0.121	6.502±0.089
450	10.362±0.275	6.940±0.100	6.682±0.147	6.712±0.098
500	11.023±0.252	7.810±0.143	7.350±0.143	7.361±0.141

5.2.3 Why Can Surrogate Fitness Help Search?

Previous empirical studies suggest that applying surrogate fitness can help Walsh-LS discover better solutions on NKq-landscapes other than NK-landscapes. The key factor that distinguishes them is that NKq-landscapes are neutral landscapes, where equal moves occur a lot and there is no gradient to follow. In such cases, Walsh-LS issues a restart. We conjecture that surrogate fitness reflects the fitness of candidate solutions that are several moves away, and the number of equal moves is greatly reduced, which leads to less restarts given the same number of moves.

We report the number of restarts in figure 5.1. For neutral landscapes including uniform and non-uniform NKq-landscapes, increasing the radius from 0 to 1 greatly reduces the number of restarts. Further increase of radius from 1 does not seem to further reduce the number of restarts. Complete restarts void previous efforts on going deep in the search space and searching on promising regions. Too many restarts prevents exact-Walsh-LS from focusing on promising regions, which results in inferior solutions. This partially explains why Walsh-LS-HS$^{(1)}$ outperforms exact-Walsh-LS on uniform and non-uniform NKq-landscapes.

Meanwhile, we observe that varying the radius does not appear to affect the number of restarts on rugged landscapes including uniform and non-uniform NK-landscapes. It conforms with the fact that using the surrogate fitness does not impact the performance of Walsh-LS-HS$^{(r)}$ on uniform and non-uniform NK-landscapes.

We only report the number of restarts on instances with $n = \{300, 500\}$ and $K = \{2, 4\}$. Nonetheless, we find the reported trends hold consistently across all tested instances in experiment 11.

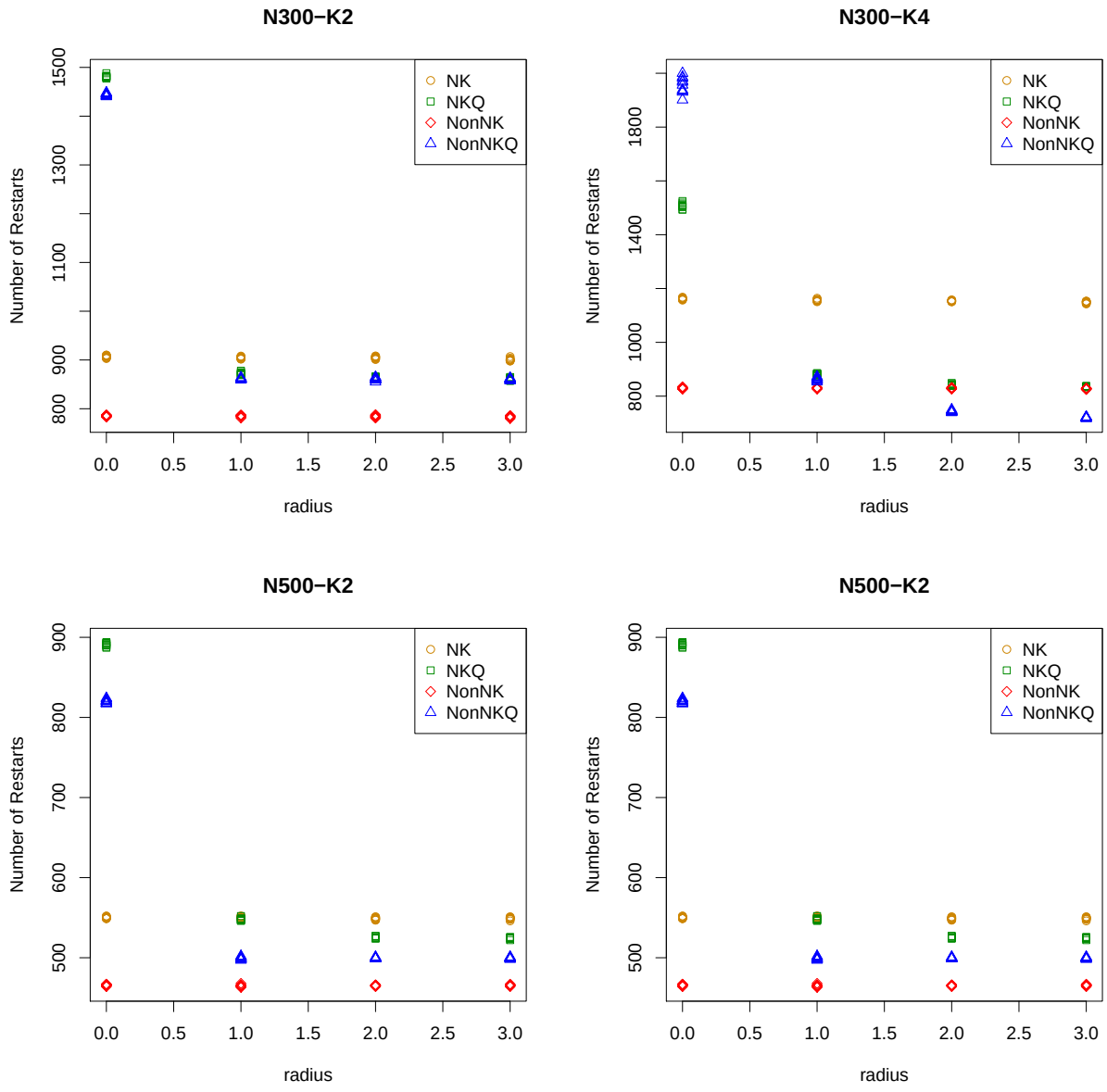

Figure 5.1: The number of restarts issued by Walsh-LS-HS$^{(r)}$ ($r = \{0, 1, 2, 3\}$) during 100000 moves.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Stochastic Local Search (SLS) is a class of simple yet effective algorithms in solving **NP**-Hard problems. The solution quality of SLS relies heavily on the number of iterations allowed for execution. Our goal is to utilize Walsh Analysis to improve both efficiency and effectiveness of SLS. Specifically, we algorithmically accelerate SLS without affecting the solution quality as described in chapter 3, and statistically significantly improve solution quality of SLS as described in chapter 5 with no extra cost in runtime.

Best-Improvement Local Search (BILS) is a common form of SLS, which always takes the move yielding the highest improvement in the objective function. Conventional BILS typically requires checking all n neighbors to determine which move to take.

With Walsh analysis, we propose a general approach, "Walsh-LS", that can determine which neighbors need to be examined. Given the objective function is epistatically bounded by a constant k, the number of neighbors that need to be checked is constant regardless of number of variables. We achieve an impressive speedup of runtime (up to 449×) versus a conventional implementation. Even though the number of neighbors that need to checked is constant, a full scan over the list of improving moves is required to determine the best improving move. In the worse case, the length of this list can be $O(n)$. To overcome this obstacle, Walsh-LS adopts approximation in selecting best improving moves, which leads to $O(1)$ complexity per move on average. To match the solution to the conventional implementation, we use Walsh-LS without approximation (denoted as *exact-Walsh-LS*) in empirical studies.

We also explore two other ways to achieve $O(1)$ complexity per move other than adopting approximation in selecting the best improving move: 1) using soft restarts (random walks) instead of random restarts upon hitting a local optimum; 2) using First-Improvement Local

Search (FILS) instead of BILS. We found that 1) exact-Walsh-LS with soft restarts indeed achieves $O(1)$ complexity per move. With carefully selected walk length, it can outperform the one using random restarts; 2) exact-Walsh-LS with FILS also achieves $O(1)$ complexity per move. However, it generally finds worse solution compared with BILS.

Finally, we attempt to improve the solution quality of exact-Walsh-LS. We conjecture that the information from candidate solutions that are several moves away can help search to determine where to focus when there is little gradient information in distance one neighbors on neutral landscapes. We utilize Walsh analysis to compute the mean values over Hamming regions of arbitrary radius and use these mean values as surrogate fitnesses to guide exact-Walsh-LS. This is achieved with no extra cost in runtime compared to the original exact-Walsh-LS. We empirically show that exact-Walsh-LS with surrogate fitness statistically significantly outperforms the original exact-Walsh-LS on neutral landscapes like NKq-landscapes, while it finds solutions of similar quality on rugged landscapes like NK-landscapes.

6.2 Future Work

Even though we have improved SLS in terms of both efficiency and effectiveness using Walsh analysis, there are still many ways to further explore along this direction.

First, while our proofs on runtime complexity should generalize to any problem that has a k-bounded pseudo-Boolean function as the objective function, the conclusion drawn from empirical studies on solution quality may not be generalized to other problem domains with k-bounded pseudo-Boolean function as objective function. Particularly, it would be interesting to examine if the improvement achieved by using surrogate fitness also holds for other neutral landscapes such as those in the Satisfiability problem. In addition, in many SAT solvers such as GSAT [SLM92] and WalkSAT [SKC94], simply the best move, regardless whether it is an improving one or not, is taken. Walsh-LS can also be extended to take such best moves.

Second, the quality of solutions returned by walk-Walsh-LS depends heavily on the setting of walk length λ. We also show the optimal setting of λ is problem-specific. It is worth

exploring some self-tuning mechanism on the setting of λ. Tabu mechanism [GL97] can also be applied to prevent local search from falling back to the same local optimum.

Third, we have theoretically show both Walsh-LS-HS$^{(r)}$ and Walsh-LS-BS$^{(r)}$ with surrogate fitness ($r > 0$) can be executed as fast as exact-Walsh-LS with the original fitness. Empirical studies show that Walsh-LS-HS$^{(r)}$ can outperform exact-Walsh-LS on NKq-landscapes. It is unclear how Walsh-LS-BS$^{(r)}$ compares to Walsh-LS-HS$^{(r)}$ in terms of solution quality.

Lastly, we have extensively employed Walsh analysis to examine the runtime complexity per move of SLS. On solution quality of SLS, however, we resort to empirical studies. It would be even more interesting if Walsh analysis could be utilized to predict or estimate the solution quality of SLS. In that case, practitioners could have some idea about the performance of SLS before it is actually run.

References

[ABJ13] Marijn Heule Adrian Balint, Anton Belov and Matti J arvisalo. The International SAT Competitions Webpage. `http://www.satcompetition.org/`, 2013. (10)

[Bet81] Albert Donally Bethke. *Genetic Algorithms as Function Optimizers*. Ph.D. Dissertation, University of Michigan, 1981. (3)

[BF98] Brian Borchers and Judith Furman. A Two-Phase Exact Algorithm for MAX-SAT and Weighted MAX-SAT Problems. *Journal of Combinatorial Optimization*, 2:299–306, 1998. 10.1023/A:1009725216438. (10)

[BH02] Endre Boros and Peter L. Hammer. Pseudo-boolean optimization. *Discrete Applied Mathematics*, 123(1-3):155 – 225, 2002. (1, 6)

[BM67] E. O. Brigham and R. E. Morrow. The fast fourier transform. *Spectrum, IEEE*, 4(12):63 –70, dec. 1967. (9)

[BR03] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, September 2003. (1)

[CA11] Francisco Chicano and Enrique Alba. Exact computation of the expectation curves of the bit-flip mutation using landscapes theory. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 2027–2034, New York, NY, USA, 2011. ACM. (5)

[CA12] Francisco Chicano and Enrique Alba. Exact computation of the fitness-distance correlation for pseudoboolean functions with one global optimum. In Jin-Kao Hao and Martin Middendorf, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 7245 of *Lecture Notes in Computer Science*, pages 111–123. Springer Berlin / Heidelberg, 2012. (5)

[CKT91] Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the Really Hard Problems Are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI-91, Sydney, Australia*, pages 331–337, 1991. (7)

[CWA12] Francisco Chicano, Darrell Whitley, and Enrique Alba. Exact computation of the expectation curves for uniform crossover. In *Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference*, GECCO '12, pages 1301–1308, New York, NY, USA, 2012. ACM. (5)

[Dun61] Olive Jean Dunn. Multiple comparisons among means. *Journal of the American Statistical Association*, 56:52–64, 1961. (48, 72)

[FCS97] Jeremy D. Frank, Peter Cheeseman, and John Stutz. When gravity fails: Local search topology. *Journal of Artificial Intelligence Research*, 7:249–281, 1997. (66)

[For09] Lance Fortnow. The status of the P versus NP problem. *Communications of the ACM*, 52(9):78–86, September 2009. (7)

[Gea01] Nicholas Geard. *An exploration of NK landscapes with neutrality.* PhD thesis, The University of Queensland, 2001. (66)

[GL97] Fred Glover and Manuel Laguna. *Tabu Search.* Kluwer Academic Publishers, Norwell, MA, USA, 1997. (8, 87)

[Gol89a] David E. Goldberg. Genetic algorithms and Walsh functions: Part I, a gentle introduction. *Complex Systems*, 3(2):129–152, 1989. (2)

[Gol89b] David E. Goldberg. Genetic Algorithms and Walsh Functions: Part II, Deception and Its Analysis. *Complex Systems*, 3:153–171, 1989. (2)

[Gro92] Lov K. Grover. Local search and the local structure of NP-complete problems. *Operations Research Letters*, 12(4):235 – 243, 1992. (4)

[GWH+02] N Geard, J Wiles, J Hallinan, B Tonkes, and B Skellett. A comparison of neutral landscapes - NK, NKp and NKq. In D B Fogel, M A El-Sharkawi, X Yao, G Greenwood, H Iba, P Marrow, and M Shackleton, editors, *Congress on Evolutionary Computation (CEC2002)*, pages 205–210. IEEE Press, 2002. (66)

[Hal27] Philip Hall. The distribution of means for samples of size n drawn from a population in which the variate takes values between 0 and 1, all such values being equally probable. *Biometrika*, 19(3/4):pp. 240–245, 1927. (40)

[Hec02] Robert B. Heckendorn. Embedded landscapes. *Evolutionary Computation*, 10(4):345–369, 2002. (4)

[Hel00] Keld Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106 – 130, 2000. (8)

[HK96] Steven Hampson and Dennis Kibler. Large plateaus and plateau search in boolean satisfiability problems: When to give up searching and start again. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26:437–455, 1996. (66)

[Hol92] John H. Holland. *Adaptation in natural and artificial systems.* MIT Press, Cambridge, MA, USA, 1992. (3)

[HRW99] Robert B. Heckendorn, Soraya B. Rana, and L. Darrell Whitley. Polynomial Time Summary Statistics for a Generalization of MAXSAT. In *GECCO*, pages 281–288, 1999. (4)

[HS04] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations & Application*. Morgan Kaufmann, 1 edition, September 2004. (1, 8, 11, 66)

[JF95] Terry Jones and Stephanie Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In Larry J. Eshelman, editor, *International Conferecen on Genetic Algorithm (ICGA)*, pages 184–192. Morgan Kaufmann, 1995. (5)

[KL87] Stuart Kauffman and Simon Levin. Towards a general theory of adaptive walks on rugged landscapes. *Journal of Theoretical Biology*, 128(1):11–45, 1987. (7)

[Kra29] Mikhail Kravchuk. Sur une généralisation des polynomes d'Hermite. *Comptes rendus de l'Académie des sciences*, 189(17):620–622, 1929. (68)

[KS11] Fredrik Kahl and Petter Strandmark. Generalized roof duality for pseudo-boolean optimization. In Dimitris N. Metaxas, Long Quan, Alberto Sanfeliu, and Luc J. Van Gool, editors, *International Conference on Computer Vision (ICCV)*, pages 255–262. IEEE, 2011. (1)

[KW89] Stuart A. Kauffman and Edward D. Weinberger. 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. (1)

[LH05] Chu Min Li and Wen Qi Huang. Diversification and determinism in local search for satisfiability. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing*, SAT'05, pages 158–172, Berlin, Heidelberg, 2005. Springer-Verlag. (15)

[LMS03] Helena Lourenǫ, Olivier Martin, and Thomas Stützle. Iterated local search. In Fred Glover and Gary Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 320–353. Springer New York, 2003. 10.1007/0-306-48056-5_11. (xiv, 9)

[Pel10] Martin Pelikan. NK landscapes, problem difficulty, and hybrid evolutionary algorithms. In *GECCO '10: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, pages 665–672, New York, NY, USA, 2010. ACM. (3)

[PSG+09] Martin Pelikan, Kumara Sastry, David E. Goldberg, Martin V. Butz, and Mark Hauschild. Performance of evolutionary algorithms on NK landscapes with nearest neighbor interactions and tunable overlap. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pages 851–858, New York, NY, USA, 2009. ACM. (3)

[RHW98] Soraya Rana, Robert Heckendorn, and Darrell Whitley. A Tractable Walsh Analysis of SAT and its Implications for Genetic Algorithms. In Jack Mostow and Chuck Rich, editors, *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 392–397. American Association for Artificial Intelligence, 1998. (3, 10, 19)

[SHS03] Kevin Smyth, Holger H. Hoos, and Thomas Stützle. Iterated Robust Tabu Search for MAX-SAT. In Yang Xiang and Brahim Chaib-draa, editors, *Canadian Conference on AI*, volume 2671 of *Lecture Notes in Computer Science*, pages 129–144. Springer, 2003. (51)

[SHW09] Andrew M. Sutton, Adele E. Howe, and L. Darrell Whitley. A theoretical analysis of the k-satisfiability search space. In *Proceedings of the Second International Workshop on Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics*, SLS '09, pages 46–60, Berlin, Heidelberg, 2009. Springer-Verlag. (4)

[SHW10] Andrew M. Sutton, Adele E. Howe, and L. Darrell Whitley. Directed Plateau Search for MAX-k-SAT. In *Proceedings of the Third Annual Symposium on Combinatorial Search (SOCS)*, 2010. (2, 66)

[SK75] David Sherrington and Scott Kirkpatrick. Solvable Model of a Spin-Glass. *Phys. Rev. Lett.*, 35:1792–1796, Dec 1975. (1)

[SKC94] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (vol. 1)*, AAAI '94, pages 337–343, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence. (8, 86)

[SLM92] Bart Selman, Hector Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI'92, pages 440–446. AAAI Press, 1992. (15, 86)

[Sta95] Peter Stadler. Towards a theory of landscapes. In Ramn Lpez-Pea, Henri Waelbroeck, Riccardo Capovilla, Ricardo Garca-Pelayo, and Federico Zertuche, editors, *Complex Systems and Binary Networks*, volume 461-461 of *Lecture Notes in Physics*, pages 78–163. Springer Berlin / Heidelberg, 1995. 10.1007/BFb0103571. (4)

[Sta96] Peter F. Stadler. Landscapes and their correlation functions. *Journal of Mathematical Chemistry*, 20:1–45, 1996. 10.1007/BF01165154. (4)

[Sut11] Andrew M. Sutton. *An analysis of combinatorial search spaces for a class of NP-hard problems*. Ph.D. Dissertation, Colorado State University, Fort Collins, CO, USA, 2011. (1, 7)

[SWH09] Andrew M. Sutton, L. Darrell Whitley, and Adele E. Howe. A polynomial time computation of the exact correlation structure of k-satisfiability landscapes. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pages 365–372, New York, NY, USA, 2009. ACM. (4)

[SWH11] Andrew M. Sutton, Darrell Whitley, and Adele E. Howe. Approximating the distribution of fitness over hamming regions. In Hans-Georg Beyer and William B.

Langdon, editors, *Foundations of Genetic Algorithms (FOGA)*, pages 93–104. ACM, 2011. (67)

[SWH12] Andrew M. Sutton, L. Darrell Whitley, and Adele E. Howe. Computing the moments of k-bounded pseudo-Boolean functions over Hamming spheres of arbitrary radius in polynomial time. *Theoretical Computer Science*, 425(0):58 – 74, 2012. (4, 67, 68, 70)

[TH04] Dave A. D. Tompkins and Holger H. Hoos. UBCSAT: An Implementation and Experimentation Environment for SLS Algorithms for SAT and MAX-SAT. In Holger H. Hoos and David G. Mitchell, editors, *SAT (Selected Papers*, volume 3542 of *Lecture Notes in Computer Science*, pages 306–320. Springer, 2004. (26)

[Wal23] J. L. Walsh. A closed set of normal orthogonal functions. *American Journal of Mathematics*, 45(1):pp. 5–24, 1923. (9)

[WC12] Darrell Whitley and Wenxiang Chen. Constant time steepest descent local search with lookahead for NK-landscapes and MAX-kSAT. In *Proceedings of the Fourteenth International Conference on Genetic And Evolutionary Computation Conference*, GECCO '12, pages 1357–1364, New York, NY, USA, 2012. ACM. (2, 11, 15, 16, 17, 18, 26, 70)

[WCH12] Darrell Whitley, Wenxiang Chen, and Adele Howe. An Empirical Evaluation of O(1) Steepest Descent for NK-Landscapes. In Carlos Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone, editors, *Parallel Problem Solving from Nature - PPSN XII*, volume 7491 of *Lecture Notes in Computer Science*, pages 92–101. Springer Berlin / Heidelberg, 2012. (2)

[Wei96] Edward D. Weinberger. NP Completeness of Kauffman's N-K Model, A Tuneable Rugged Fitness Landscape. Working Papers 96-02-003, Santa Fe Institute, February 1996. (3)

[Whi11] Darrell Whitley. Defying Gravity: constant time steepest ascent for MAX-kSAT. Technical report, Colorado State University, December 2011. (11)

[Zha04] Weixiong Zhang. Configuration landscape analysis and backbone guided local search: part i: Satisfiability and maximum satisfiability. *Artificial Intelligence*, 158:1–26, September 2004. (1)

[ZHS08] Shude Zhou, Robert Heckendorn, and Zengqi Sun. Detecting the epistatic structure of generalized embedded landscape. *Genetic Programming and Evolvable Machines*, 9:125–155, 2008. 10.1007/s10710-007-9045-7. (4)