Understanding the Cost of Fitness Evaluation for Subset Selection: Markov Chain Analysis of Stochastic Local Search

Ole Jakob Mengshoel
ole.j.mengshoel@ntnu.no
Norwegian University of Science and Technology
Trondheim, Norway

Eirik Lund Flogard
eirik.flogard@arbeidstilsynet.no
Norwegian Labour Inspection Authority
Trondheim, Norway

Tong Yu
worktongyu@gmail.com
Carnegie Mellon University
Pittsburgh, Pennsylvania

Jon Riege
Riege.Jon@bcg.com
Boston Consulting Group
Oslo, Norway

ABSTRACT

With a focus on both the fitness and cost of subset selection, we study stochastic local search (SLS) heuristics in this paper. In particular, we consider subset selection problems where the cost of fitness function evaluation needs to be accounted for. Here, cost can be fitness evaluation's computation time or energy cost. We propose and study an SLS method, SLS4CFF, tailored to such problems. SLS4CFF ("SLS for costly fitness functions") is an amalgamation of several existing heuristics. We develop a homogeneous Markov chain model that explicitly represents both fitness and cost of subset selection with SLS4CFF. This Markov chain, which can be lumped or compressed for certain fitness and cost functions, enables us to better understand and analyze hyperparameter optimization in a principled manner, via expected hitting times. Studies with synthetic and real-world problems improve the understanding of SLS and demonstrate the importance of cost-awareness.

CCS CONCEPTS

• **Computing methodologies → Randomized search**; *Classification and regression trees*; • **Mathematics of computing → Markov processes**.

KEYWORDS

Pseudo-Boolean functions, costly fitness functions, synthetic problems, optimization, stochastic local search, Markov chain analysis, expected hitting time, machine learning, classification

ACM Reference Format:
Ole Jakob Mengshoel, Eirik Lund Flogard, Tong Yu, and Jon Riege. 2022. Understanding the Cost of Fitness Evaluation for Subset Selection: Markov Chain Analysis of Stochastic Local Search. In *Genetic and Evolutionary Computation Conference (GECCO '22), July 9–13, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3512290.3528689

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
GECCO '22, July 9–13, 2022, Boston, MA, USA
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9237-2/22/07…$15.00
https://doi.org/10.1145/3512290.3528689

1 INTRODUCTION

Context. Stochastic local search (SLS) has many applications in artificial intelligence (AI) and machine learning (ML): model finding [17, 46, 62], neural architecture search [60], sentence summarization [47], subset selection [3, 50], compressed sensing [42], and computing most probably explanations in Bayesian networks [20, 30, 33, 34]. SLS, similar to evolutionary algorithms (EAs), typically rely on the evaluation of a fitness function to drive search forward.

The cost of fitness function evaluation continues to receive attention in the literature [25, 29, 52]. Here, "cost" may refer to computational cost, energy cost, or some other cost. Reasons to study costly fitness functions (CFFs) include the following. First, in certain important applications, fitness function evaluation has very high computational cost. Feature selection using wrappers [22] is an application that may have a CFF. Other potential CFF applications include black-box topology optimization of mechanical structures [12, 67] and neural architecture search in ML [60]. As a concrete example, the amount of compute used in the largest ML training runs has recently increased exponentially with a 3.4-month doubling time.[1] Second, fitness function evaluation may vary dramatically over the search space, and may have high cost only in certain parts of the search space [25, 29, 52]. In ML, for example, complex models using many features may or may not give better accuracy than a simple model using few features, but the computational cost of using many features will very likely be higher.

Cost-related studies from the literature fall into different categories. One can directly minimize fitness evaluation cost by using approximate or surrogate fitness functions [51] or non-revisiting methods [27, 66]. Alternatively, one can minimize fitness evaluation cost by optimizing algorithms' heuristics and hyperparameters [5, 19, 34, 52]. Our work falls into this latter category, with a focus on SLS for subset selection and motivated by the problem of feature selection [13, 22, 31, 32, 35, 63].

Problems. We study a certain class of CFF subset selection problems, namely pseudo-Boolean functions where fitness function evaluation is costly and often varying. We study SLS4CFF, a simple SLS algorithm that nevertheless integrates several heuristics while being amendable to Markov chain analysis. A potential benefit of Markov chain analysis is the formulation of expected hitting time results for SLS; expected hitting times are analytical counterparts to

[1]https://openai.com/blog/ai-and-compute/

average runtimes in experiments. Unfortunately, due to a previous lack of focus on the cost of fitness function evaluation, existing expected hitting time models for SLS [8, 30, 31, 38, 40, 64] do not reflect CFFs well. Thus, we consider this question: what is the impact of CFFs on SLS for subset selection, especially when it comes to tuning and optimizing SLS4CFF's hyperparameters?

Contributions. To address CFFs in the context of SLS for subset selection, we consider in SLS4CFF the integration of multiple SLS heuristics and make contributions in the following areas. *First*, we carefully study the behavior of SLS4CFF by means of Markov chains and synthetic problems. The Markov chain formalizes the SLS4CFF search process, given a problem instance and hyperparameter settings, capturing both varying fitness and varying computational cost. Our Markov chain models enable, via expected hitting times, an integrated study of fitness and cost. *Second*, we study cost models both for synthetic problems and real-world ML problems. Our experiments with real-world datasets and ML classifiers (Decision Tree, Naive Bayes, and Support Vector Machine) suggest that a linear cost model approximates the computational cost well for these ML methods. *Third*, we observe that a naive Markov chain representation of SLS4CFF's behavior, while exponential in size, can under certain conditions be lumped or compressed. Experimentally, the compressed Markov chain model scales exceptionally well with problem size, thus enabling improved analysis opportunities compared to the naive Markov chain.

2 PROBLEM STATEMENT

We here discuss two goals that we seek to achieve when addressing CFF problems by means of SLS when applied to subset selection.
Maximizing Fitness. We study search spaces consisting of bitstrings $\mathbb{B} = \{0, 1\}^n$. In *subset selection*, fitness f is a pseudo-boolean function (PBF) that maps from $\mathbb{B}$ to the non-negative real numbers $\mathbb{R}_{\geq 0}$. Our focus is to optimize (without loss of generality, maximize) the fitness function f:

$$\boldsymbol{b}^* = \arg\max_{\boldsymbol{b} \in \mathbb{B}} f(\boldsymbol{b}) . \tag{1}$$

For simplicity, we assume in (1) a single global optimum $\boldsymbol{b}^*$ rather than multiple global optima $\{\boldsymbol{b}_1^*, \boldsymbol{b}_2^*, \ldots\}$.
Analyzing Cost. While we are interested in cost generally, we consider computational cost as a concrete example. What is the time it takes for an SLS algorithm to search for and find $\boldsymbol{b}^*$? There are several factors, let us highlight two. First, it depends on the time it takes to evaluate a state $\boldsymbol{b} \in \mathbb{B}$: $g(\boldsymbol{b})$, where g is a cost function that maps from $\mathbb{B}$ to $\mathbb{R}_{\geq 0}$. Second, an SLS algorithm will in general evaluate $g(\boldsymbol{b})$ for many $\boldsymbol{b} \in \mathbb{B}$ when running. Since SLS is randomized, run time is a random variable Q. Clearly, Q depends on the problem instance being optimized, the specific SLS algorithm, and its hyperparameter and heuristic settings. Of interest are the distribution of Q and its expectation $E(Q)$ which in the Markov chain setting is denoted the expected hitting time h. Expected hitting time is crucial since it corresponds to average run time as measured empirically in experiments [30, 31]; clearly it provides a desireable nonmyopic measure [25, 29].

3 RELATED RESEARCH

Stochastic Local Search (SLS). Several SLS techniques and results [3, 8, 17, 30, 32, 34, 48, 49] are studied in or can be adapted to a subset selection setting. This includes algorithmic techniques such as noisy search, restart, and initialization. Randomized restart combined with hillclimbing (or greedy steps) are the key operators in GSAT, Selman et al.'s seminal local search algorithm for solving satisfiability problems in propositional logic [49]. GSAT has two important hyperparameters, optimized empirically, namely MAX_TRIES (controlling termination) and MAX_FLIPS (controlling the number of flips before restart). Several extensions to GSAT are proposed, among them a form of random walk [48]. The random walk, either in its pure or hybridized form, also plays a role in other applications [11, 61]. Randomization interleaved with hillclimbing steps has also proven useful in SLS for sparse signal recovery. Specifically, the SLS algorithm StoCoSaMP [42] substantially improves the performance of a well-established greedy pursuit algorithm without randomization, CoSaMP [37].

Pan et al.'s work on automatic image captioning [43] uses random walk with restart (RWR). RWR differs from GSAT in several ways, including RWR's use of probabilistic restart and a fixed node (state) for initialization. To compute most probable explanations in Bayesian networks (BNs), it has also been demonstrated experimentally that SLS can be highly competitive with clique tree clustering [34] while supporting formal analysis [30, 33, 34].

Another SLS technique is randomized neighborhood selection, where local search is conducted on a λ-sized random sample of neighbors in a solution space [57]. Randomized neighborhood selection is related to random walk; they are equivalent if $\lambda = 1$. Using randomized neighborhood selection can reduce the size of the neighborhood in local search and thus improve search efficiency in both SAT and project scheduling problems [1, 36].

We note two limitations of much previous SLS research. First, many algorithms integrate multiple heuristics [42, 57] but are too complex for theoretical analysis with homogeneous Markov chains. Second, although different heuristics are known, they have not yet been thoroughly analyzed in terms of computational costs. In contrast, we attack these two limitations by integrating fitness and cost considerations into a Markov chain framework for SLS.
Cost of Fitness Evaluation. The computation time for each state of traditional SLS [15, 30, 48] is fast, often on the order of microseconds or milliseconds. Thus, the computational cost of different SLS search steps has been abstracted away without much harm in many previous theoretical studies of SLS [30, 31, 38, 40, 64]. However, fitness evaluation cost varies from application to application and is important in the CFF setting. Examples of CFF problems include black-box topology optimization of mechanical structures [12, 67] and ML in the form of feature selection or deep learning, sometimes with massive datasets [24, 26, 28, 35, 35]. Further, fitness function evaluation cost may vary dramatically over the search space [25, 29, 52]. How can such problems be attacked? Among several approaches, our focus here is to minimize fitness evaluation cost via hyperparameter tuning [19, 31, 34], using Markov chain analysis as discussed in more detail below.
Markov Chain Analysis. Markov chain analysis, in particular expected hitting time analysis [8, 10, 14, 30, 38, 40, 56, 64], is used

in both EA and SLS settings. To complement experimental results, parametric Markov chain models of SLS have been formulated, in which SLS hyperparameters show up as parameters [30, 31]. Using these parametric Markov chains, we can compute expected hitting times, which turn out to be rational functions (ratios of polynomials) for individual problem instances as well as for their mixtures. Expected hitting time functions [30] correspond to noise response curves often reported in experimental literature. These functions can aid in SLS performance optimization [30, 31].

Existing expected hitting time results for SLS [8, 10, 30, 38, 40, 64] have some limitations when applied to the analysis of CFF problems. In particular, existing Markov chain models and results do not incorporate cost, which make them limited when it comes to CFF problems such as feature selection with wrappers [13, 22, 32, 63]. With SLS4CFF we seek to address this problem.

Hyperparameter and Bayesian Optimization (BO). SLS algorithms typically have multiple hyperparameters that determine their performance. Thus, setting or optimizing such hyperparameters is essential. Karafotias et al. identify two ways of setting hyperparameters[2] and say that [19]: "the tuning problem is the stationary side, while the control problem is the nonstationary side of the same coin." Reflecting this distinction, two SLS variants, SoftSLS and AdaptiveSLS [31], provide different ways of setting hyperparameters [19]. In SoftSLS, hyperparameters are tuned (offline), while in AdaptiveSLS they are controlled (online).

To tune SLS4CFF's hyperparameters one could adopt general hyperparameter optimization methods such as Bayesian optimization (BO) [9, 52, 55]. BO has recently seen an increased focus on cost-aware and -constrained computation [25, 29]. However, by directly applying BO to optimize SLS, SLS is treated as a black box, thus the algorithm design may not be optimal. Instead, our approach considers the unique characteristics of SLS4CFF, and hyperparameter tuning is guided by cost-aware Markov chain analysis.

Multi-Objective Optimization (MOO). Multi-objective optimization (MOO) algorithms, implemented for example by means of evolutionary algorithms [4, 6], may seem suitable for computing f and g as their two objectives f_1 and f_2. An MOO algorithm would compute a Pareto front that approximates Pareto optimality for $f_1 = f$ and $f_2 = g$, thus trading off these two objectives.

However, the purpose of SLS4CFF and its associated Markov chain analysis (see Section 6) is not to use both fitness and computational cost as objectives f_1 and f_2 for MOO. Our primary goal is maximization of fitness f, but the analysis framework enables us to also represent and analyze the cost, namely g, of doing so. Thus we treat minimization of g as a secondary goal, not as an objective on par with optimization of f in the MOO sense.

4 SLS FOR MARKOV CHAINS WITH COST

SLS4CFF builds on existing research on SLS [8, 17, 30–32, 34, 58, 59, 62] and is most closely related to SoftSLS [31] and SLS4FS [32, 35]. Compared to SoftSLS [31], an existing SLS algorithm with Markov chain analysis, SLS4CFF puts emphasis on integration of cost into the Markov chain analysis. And in contrast to our Markov chain analysis of SLS4CFF, results for SLS4FS [32, 35] are strictly experimental and concern feature selection.

Algorithm 1: SLS4CFF Algorithm.

```
input  : Probability $P_r$ of restart step, probability $P_n$ of
         noise step, filter $F$, fitness function $f$ for computing
         $f(\boldsymbol{b})$ for subset $\boldsymbol{b}$, termination threshold $\tau$, Greedy
         flag $X$ (strict $X = 0$ or soft $X = 1$).
output : Optimized subset $\boldsymbol{b}^+$
1  $\boldsymbol{b} \leftarrow$ Restart($F$), $c \leftarrow 0$, $f^+ \leftarrow 0$, $\boldsymbol{b}^+ \leftarrow \boldsymbol{b}$
2  while $f^+ < \tau$ do
3      if Rand(0,1) $< P_r$ then
4          $\boldsymbol{b} \leftarrow$ Restart($F$)
5      else
6          if rand(0,1) $< P_n$ then
7              $\boldsymbol{b} \leftarrow$ Noise($\boldsymbol{b}$)
8          else
9              $\boldsymbol{b} \leftarrow$ Greedy($\boldsymbol{b}, f, X$)
10     if $f(\boldsymbol{b}) > f^+$ then
11         { Update the current best subset $\boldsymbol{b}^+$ }
12         $f^+ \leftarrow f(\boldsymbol{b})$
13         $\boldsymbol{b}^+ \leftarrow \boldsymbol{b}$
14 return $\boldsymbol{b}^+$
```

4.1 SLS Algorithm

Local search takes place in the proximity, or in the neighborhood, of a current state $\boldsymbol{b} \in \mathbb{B}$.

*Definition 4.1 (**Neighborhood**).* Let $\boldsymbol{b} = (b_1 \dots b_i \dots b_n) \in \{0,1\}^n$, $\boldsymbol{b}' = (b'_1 \dots b'_i \dots b'_n) \in \{0,1\}^n$, and define Hamming distance H as $H(\boldsymbol{b}', \boldsymbol{b}) = \sum_{i=1}^{n} (b'_i \oplus b_i)$. The neighborhood $N(\boldsymbol{b}) \subseteq \{0,1\}^n$ of $\boldsymbol{b} \in \mathbb{B}$ is defined as all bitstrings with a Hamming distance H of one to $\boldsymbol{b}$: $N(\boldsymbol{b}) = \{\boldsymbol{b}' \in \{0,1\}^n \mid H(\boldsymbol{b}', \boldsymbol{b}) = 1\}$.

For big or high-dimensional data, using $N(\boldsymbol{b})$ may be too compute-intensive and one can therefore use subsets of it. Tari et al. [57] call such neighborhoods partial, and find their use beneficial in several experiments, as have Mengshoel et al. [32] in feature selection. However, our focus here is on the complete case, $N(\boldsymbol{b})$.

Input. The SLS4CFF algorithm, see Algorithm 1, takes these *inputs*: probability of Restart P_r, probability of Noise P_n, filter F, fitness function f, termination threshold τ, and type of Greedy step X.[3]

Search. Given the current state $\boldsymbol{b}$, SLS4CFF searches for an optimal state $\boldsymbol{b}^*$ via repeated application of a greedy step Greedy, a noise step Noise, and a restart step Restart.[4] Let $\overline{P}_r = 1 - P_r$ and $\overline{P}_n = 1 - P_n$. In each iteration, SLS4CFF performs Greedy with probability $\overline{P}_r\overline{P}_n$ (line 9); Noise with probability $\overline{P}_r P_n$ (line 7); or Restart with probability P_r (line 4). These search steps are defined as follows.

[2]They in fact say parameter instead of hyperparameter, however we prefer the latter term since our work is motivated, in part, from ML where parameter learning is key and different from hyperparameter learning or optimization.

[3]The closely related algorithms MarkovSLS [31] and SLS4FS [32] include adaptation of P_r and P_n, using adaptation rates α_r and noise α_n respectively. Using $\alpha_r = \alpha_n = 0$ for MarkovSLS and SLS4FS gives no hyperparameter adaptation, in other words hyperparameter tuning and not control [19], as is hard-coded in SLS4CFF.

[4]The SLS4CFF pseudocode is not optimized for performance. For example, $f(\boldsymbol{b})$ in line 10 will in fact only require a new evaluation if a Noise or Restart step has been performed. After Greedy, $f(\boldsymbol{b})$ will already have been evaluated.

A greedy step Greedy($\boldsymbol{b}$, f, X) computes from bitstring $\boldsymbol{b}$ a bitstring $\boldsymbol{b'} \in N(\boldsymbol{b})$ while maximizing the objective function $f(\boldsymbol{b'})$. If there is a tie in $f(\boldsymbol{b'})$ among some neighbors in $N(\boldsymbol{b})$, one of these neighbors is picked uniformly at random. A *strict* greedy step ($X = 0$) stays with $\boldsymbol{b}$ if $f(\boldsymbol{b}) \geq f(\boldsymbol{b'})$ for all $\boldsymbol{b'} \in N(\boldsymbol{b})$, while a *soft* greedy step ($X = 1$) always moves to a best-fit neighbor.

There are different ways to randomize noise steps [15, 30, 33]. We study an easy-to-analyze method of picking a neighbor uniformly at random. A noise step Noise($\boldsymbol{b}$) thus randomly jumps from bitstring $\boldsymbol{b}$ to a neighbor $\boldsymbol{b'} \in N(\boldsymbol{b})$; $\boldsymbol{b'} \neq \boldsymbol{b}$.

Restart can positively impact SLS performance [33, 34, 46, 49]. An SLS4CFF Restart(F) step computes a bitstring $\boldsymbol{b} \in \{0, 1\}^n$, using a filter F. We use for F a uniform at random filter F_U: For each bit b_i, where $1 \leq i \leq n$, we flip an unbiased coin. If the coin comes up heads, $b_i = 1$, else $b_i = 0$.

SLS4CFF hyperparameters (P_r, P_n, F, and X) can be controlled online or tuned offline [19, 31]. We use in SLS4CFF offline tuning; hyperparameters are fixed while the algorithm is running. This enables analysis via homogeneous Markov chains [14, 30, 31].

Termination and Output. SLS4CFF terminates and *outputs* $\boldsymbol{b}^+$ as an approximation to $\boldsymbol{b}^*$ when reaching a threshold τ. To enable our analysis in Section 6, we put $\tau = f(\boldsymbol{b}^*)$, thus $\boldsymbol{b}^+ = \boldsymbol{b}^*$. Other termination critera can be used, including wall-clock time, a fixed number of iterations, poor improvement of f^+, and so on.

4.2 Cost of Fitness Function Evaluation

What is the connection between SLS4CFF (in Algorithm 1) and the cost of fitness function evaluation? SLS4CFF calls or evaluates the fitness function f in line 9 (inside Greedy) and in line 10. In fact, f is called n times in line 9 and as such this is the most computationally challenging part of SLS4CFF when n is high and computing f is costly, as formalized by g.

When SLS4CFF is applied to feature selection, $f(\boldsymbol{b})$ could represent the accuracy score of a classifier L using a subset $\boldsymbol{b}$ of features from a data set D [32]. Feature selection is a problem where cost g typically is important, as evaluating a specific feature subset $\boldsymbol{b}$ when learning L may take minutes or hours. There may also be differences in computational costs between different search steps, since L using few features is usually evaluated quicker than using many features.

We exemplify f and g in Section 5, and integrate them with SLS4CFF's operation via a Markov chain analysis in Section 6.

5 ENABLING COST-AWARE MARKOV CHAIN ANALYSIS: FITNESS AND COST FUNCTIONS

In general, SLS studies may have multiple goals. In this work we are focused on goals related to fitness f and cost g. To enable theoretical analysis of SLS4CFF, we introduce a few key distinctions and assumptions about f and g in this section.

5.1 Fitness Function f

The hurdle and cosine trap problems are examples of synthetic subset selection problems. These problems are interesting multi-modal optimization problems in that they contain multiple local but non-global optima. Escaping local optima and finding the global optimum $\boldsymbol{b}^*$ for such problems can be challenging, partly due to the large "valleys" of the local optima [41]. We use the problems in Section 6 to analyze the performance of SLS4CFF.

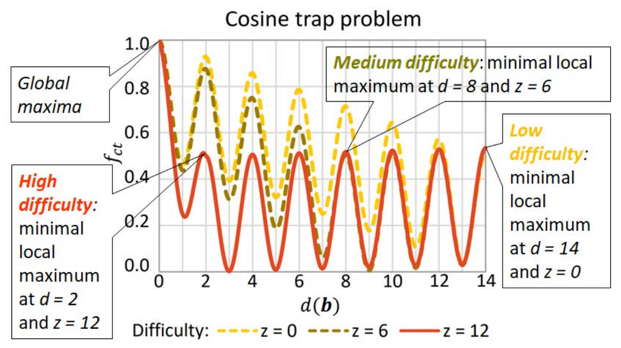

Figure 1: Three cosine trap problems with varying difficulty z. The graph for the high difficulty $z = 12$ problem partly hides the $z = 0$ and $z = 6$ graphs to the right in the plot.

5.1.1 Cosine Trap Problem. We now introduce the cosine trap problem as an example of a subset selection problem. As a problem with many local optima, one global optimum, and problem instances of varying hardness, we define a cosine trap problem (see [31, 63]):

*Definition 5.1 (**Cosine Trap Problem**).* The cosine trap problem f_{ct} maps the distance $d(\boldsymbol{b}) = H(\boldsymbol{b}^*, \boldsymbol{b})$ from optimum $\boldsymbol{b}^*$ and a difficulty setting $z \in [0, n)$ to a fitness value $f_{ct}(d, z) \in [0, 1]$.

$$f_{ct}(d, z) = \begin{cases} \frac{1}{4} + \frac{1}{2}(1 + \frac{d}{z-n}) + \frac{1}{4}\cos \pi d & \text{if} \quad d \leq n - z \\ \frac{1}{4} + \frac{9}{20}\frac{d-n+z}{(n-1)z} + \frac{1}{4}\cos \pi d & \text{otherwise.} \end{cases} \tag{2}$$

Here, z specifies the location of the minimal local maximum and controls the degree of deception. Intuitively, the location of the minimal local maximum (and the global minimum) of a cosine trap function is closer to $\boldsymbol{b}^*$ for higher values of z, thus increasing the problem's difficulty. In other words, the closer z is to n, the more difficult the problem is. The problem integrates ideas from deceptive trap [7] and hurdle [45] functions. These functions are motivated by search space structures and traps in real-world problems [16, 45].

We study a 14-bit cosine trap problem, see Figure 1 and Figure 3, where $0 \leq z \leq 12$. In Figure 3 the optimal state has Hamming weight $w(\boldsymbol{b}^*) = 5$. Hamming weight w is the number of bits set to 1 in a bitstring $\boldsymbol{b}$.

5.1.2 Hurdle Problem. The hurdle problem was introduced by Prügel-Bennett [45] and recently used, for example, to analyze memetic algorithms [39].

*Definition 5.2 (**Hurdle Problem**).* The hurdle problem f_h maps the distance $d(\boldsymbol{b}) = H(\boldsymbol{b}^*, \boldsymbol{b})$ from optimum $\boldsymbol{b}^*$ and a hurdle width $w \in \{2, 3, \ldots, n\}$ to a fitness value $f_h(d, w)$.

$$f_h(d, w) = -\left\lceil \frac{d}{w} \right\rceil - \frac{\text{rem}(d, w)}{w}, \tag{3}$$

where the hurdle width $w = w(n)$ can be a function of n, d is the number of zero-bits of $\boldsymbol{b}$, and rem(d, w) is the remainder of d divided by w. By varying the hurdle width w, we can adjust the distance between the local optima.

As an example, we study a 14-bit hurdle problem, where $2 \leq w \leq 4$, see Figure 3. The local optima in the hurdle problem occur

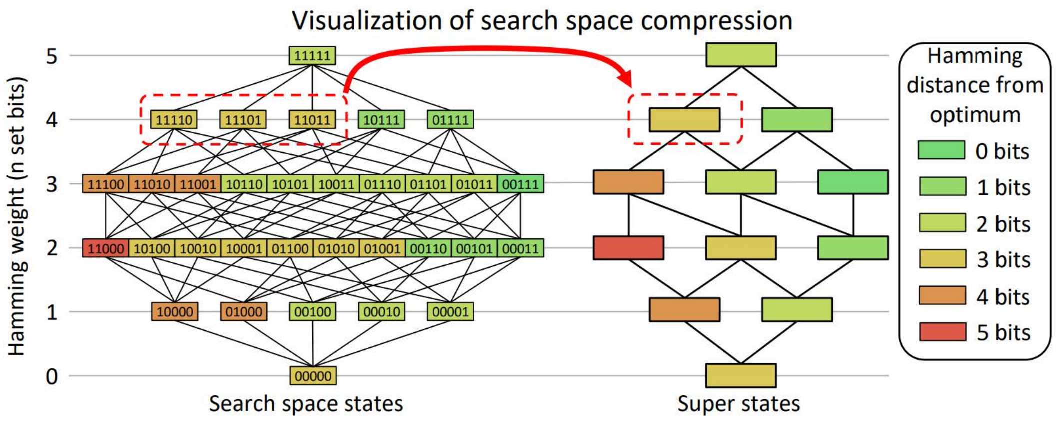

Figure 2: A small-scale illustration of search state compression. Optimum is $b^* = 00111$. The compressed search space (right) contains only 12 super states, down from 32 states in the original search space (left). On the left, encircled by red dashes, are three states with the same Hamming weight (number of set bits) and Hamming distance from the optimum (3 bits). These states form a super state, encircled by red dashes to the right. The degree of compression is much greater in our experiments.

whenever $\text{rem}(d, w) = 0$. The number of local optima for a 14-bit problem decreases as w increases.

The cosine trap and hurdle problems are similar in that they contain (i) multiple local but non-global optima and (ii) parameters to vary their difficulty. However, there are also differences. In the hurdle problem, w specifies distance between local optima. In the cosine trap problem, the width between local optima is fixed. In cosine trap problems the z-parameter specifies degree of deception, while there is no deception parameter in hurdle problems.

5.2 Cost Function g

For SLS4CFF, given the CFF assumption, substantial cost occurs as a result of performing search steps. The cost of a Greedy step is the sum of the costs of evaluating every neighbor while the cost of a Restart or Noise step is only the cost of evaluating one state (see Section 4.2). A frequent assumption in earlier work is that cost g is constant [30, 31]: $g(\boldsymbol{b}) = 1$. In reality, the cost tends to vary with $\boldsymbol{b}$. In this work we therefore also study linear cost functions: $g(\boldsymbol{b}) = \alpha w(\boldsymbol{b}) + \beta$, where $\alpha, \beta \in \mathbb{R}_{\geq 0}$ and $w(\boldsymbol{b})$ is the Hamming weight of $\boldsymbol{b}$. This linear model is suitable for simple ML tasks such as wrapper-based feature selection for classifiers.

Complex ML tasks, such as learning the structure of deep neural networks or Bayesian networks [24, 26, 28, 60], clearly have super-linear cost and are examples of CFF problems. Thus, it is of interest to consider cost functions that go beyond the linear case. A quadratic cost function of the form $g(\boldsymbol{b}) = w(\boldsymbol{b})^2$ is also studied, along with an exponential cost function $g(\boldsymbol{b}) = 2^{w(\boldsymbol{b})}$.

Cost g is integrated into the Markov chain model in Section 6. Numerical examples of expected hitting times of the linear, quadratic, and exponential cost functions are studied in Section 7.

6 MARKOV CHAIN MODEL WITH COST

In this section we consider SLS4CFF (see Section 4) when operating under certain fitness and cost functions, in particular the fitness and cost landscapes discussed in Section 5.

When applying SLS4CFF to CFF problems, a key difference from many other SLS applications are: (i) the varying computational cost of search steps and (ii) the integration of multiple heuristics. In this section we are introducing a lumpable Markov chain model to analyze the performance of SLS4CFF in terms of solving the cosine trap problem.[5] The lumped Markov model enables us to analyze significantly larger problems than would otherwise be tractable, and thus inform the practical application of SLS4CFF.

Markov Chain Models of SLS4CFF. SLS4CFF can be viewed from the perspective of stochastic processes. In fact, under the reasonable assumption that F is memoryless (as F_{U} is), SLS4CFF's application to a problem can be modelled as a discrete homogeneous Markov chain $\mathcal{M} = (\boldsymbol{T}, \boldsymbol{\pi})$. The initial vector $\boldsymbol{\pi}$ assigns, for SLS4CFF with $F = F_{\text{U}}$, equal probabilities to every state. For two states $\boldsymbol{b}, \boldsymbol{b'}$ an entry $T_{\boldsymbol{b},\boldsymbol{b'}}$ in the transition matrix $\boldsymbol{T}$ is [31]:

$$\begin{aligned} T_{\boldsymbol{b},\boldsymbol{b'}} &= \overline{P}_r\overline{P}_n P(\boldsymbol{b'}|\boldsymbol{b}, \text{Greedy}) + \overline{P}_r P_n P(\boldsymbol{b'}|\boldsymbol{b}, \text{Noise}) \\ &\quad + P_r P(\boldsymbol{b'}|\text{Restart}), \end{aligned} \tag{4}$$

where the three terms in (4) are induced, respectively, by the Greedy, Noise, and Restart steps in Algorithm 1.

In order to calculate the first passage time $m(\boldsymbol{b})$ of each state, including the cost of computation, we make two modifications to an existing model [31]. First, the row in the transition matrix with transitions from $\boldsymbol{b}$ now has a corresponding first passage time for $\boldsymbol{b}$ as a linear combination of those of other states. Based on this, the expected computational cost of a transition $\mathbb{E}[C_{\boldsymbol{b}}]$ from $\boldsymbol{b}$ can be expressed as:

$$\begin{aligned} \mathbb{E}[C_{\boldsymbol{b}}] &= \overline{P}_r\overline{P}_n \sum_{\boldsymbol{b}_n \in N(\boldsymbol{b})} g(\boldsymbol{b}_n) + \overline{P}_r P_n \frac{1}{n} \sum_{\boldsymbol{b}_n \in N(\boldsymbol{b})} g(\boldsymbol{b}_n) \\ &\quad + P_r \sum_i \pi_i g(\boldsymbol{b}_i). \end{aligned} \tag{5}$$

For Greedy (the $\overline{P}_r\overline{P}_n$-term in (5)) this is the cost of evaluating each state in the neighborhood $N(\boldsymbol{b})$. For Noise (the $\overline{P}_r P_n$-term) it is the average evaluation cost in $N(\boldsymbol{b})$. For Restart (the P_r-term) it is the average evaluation cost of all states weighted by $\boldsymbol{\pi}$.

[5]The use of both lumped and unlumped Markov chains for representing and analyzing the search space for optimization problems is well-established for genetic and SLS algorithms [30, 31, 53, 54, 56, 65]. A novel aspects of our work is the integration of both fitness and computational cost in a *lumped* Markov chain model, reflecting multiple SLS4CFF heuristic.

The second modification is the following. We assume a single optimal state $\boldsymbol{b}^*$, set $m(\boldsymbol{b}^*) = 0$, and express $m(\boldsymbol{b})$ as:

$$m(\boldsymbol{b}) = \begin{cases} 0 & \text{if} \quad \boldsymbol{b} = \boldsymbol{b}^* \\ \mathbb{E}[C_{\boldsymbol{b}}] + \sum_{\boldsymbol{b}'} T_{\boldsymbol{b},\boldsymbol{b}'} m(\boldsymbol{b}') & \text{otherwise.} \end{cases} \tag{6}$$

The expected hitting time $m(\boldsymbol{b})$ of the search is the sum of the first passage times weighted by $\boldsymbol{\pi}$. Here, (6) forms a system of linear equations that can easily be solved numerically, see Section 7.

Markov Chain Compression. The number of states n_S in the search space grows exponentially with n; $\boldsymbol{T}$'s size is n_S^2. Unless we lump or compress the search space, Markov chain analysis of even medium sized problems would be computationally infeasible. We accomplish this compression by making two assumptions in the synthetic problems we study, partly supported by experiments in Section 7.2. First, cost $g(\boldsymbol{b})$ depends only on the Hamming weight of the state $\boldsymbol{b}$. Second, fitness $f(\boldsymbol{b})$ depends only on $H(\boldsymbol{b}, \boldsymbol{b}^*)$. These assumptions let us lump states with the same Hamming weight and distance from $\boldsymbol{b}^*$ into "super states," and we have this result:

Proposition 6.1. *Let the $\mathcal{M}$ be the Markov chain defined above. Let $A = \{A_{x,y}\}$ be the super state partition of $\mathcal{M}$ where a super state $A_{x,y}$ contains all states $\boldsymbol{b}$ with Hamming weight $w(\boldsymbol{b}) = x$ and distance $H(\boldsymbol{b}^*, \boldsymbol{b}) = y$ from optimum. Consequently, $\mathcal{M}$ is lumpable with respect to A.*

Since $\mathcal{M}$ is lumpable with respect to the super state partition A, A produces a valid Markov chain that allows a coarser analysis, including a hitting time analysis, of SLS4CFF's search process [21]. In brief, we have the following:

Proof sketch. We first observe that it is sufficient that the search space G has lumpable neighborhoods with respect to A, to prove that $\mathcal{M}$ is lumpable with respect to A. Then we can show that G always has lumpable neighborhoods with respect to A.

The goal of lumpability or compression is to make Markov chain analysis tractable. A small visual example of how our compression works for a 5-bit problem is in Figure 2.

7 NUMERICAL EXAMPLES AND EXPERIMENTS

We provide in Section 7.1 and Section 7.3 numerical examples where we analyze how varying the hyperparameters of SLS4CFF affects the expected hitting time for various synthetic problems. We also conduct ML experiments, in Section 7.2, where we test our linear cost model from Section 6 using real-world datasets. Search space compression is investigated in Section 7.4.

The numerical examples and experiments are implemented in Python with libraries including NumPy, SciPy, and Scikit-learn.[6] Experiments in Section 7.2 are executed in NTNU's IDUN cluster environment,[7] using one CPU and 24 GB of memory per run.

7.1 Finding CFF Expected Hitting Times

Goal. How are CFF expected hitting times impacted when cost is integrated into the Markov chain analysis for SLS4CFF?

Method and Data. The expected hitting times for the hurdle and cosine trap problems are found via Equation 6 by means of Python's NumPy library,[8] as the optimal state $\boldsymbol{b}^*$ is known. Putting $F = F_{\text{U}}$, a commonly used approach [48, 49, 57], the expected hitting time curves are shown in Figure 3 for cosine trap problems and in Figure 4 for hurdle problems. The expected hitting time is recorded as the noise probability P_n is varied.

A cost model is represented by the function g in Equation 5, used in Equation 6. Figure 3(a) and Figure 4(a) use constant cost $g(\boldsymbol{b}) = 1$ [30, 31]. Linear cost models are used in Figure 3(b)-(d) and Figure 4(b)-(d). The cost of evaluating a state $\boldsymbol{b}$ is its Hamming weight $w(\boldsymbol{b})$, giving $g(\boldsymbol{b}) = \alpha w(\boldsymbol{b}) + \beta$, simplified to $g(\boldsymbol{b}) = w(\boldsymbol{b})$ for $\alpha = 1, \beta = 0$. Strict Greedy is used with SLS4CFF in (b), while soft Greedy is used in (c) and (d). The restart probability is $P_r = 0$ for (b) and (c); $P_r = 1/14$ for (d).

Results and Discussion. Both Figure 3 and Figure 4 show that the expected hitting times vary drastically with noise P_n, although P_n becomes less important when restart is added in (d). For cosine trap problems, the expected hitting times also vary much with problem difficulty z. The optimal noise P_n^* for the cosine trap problems increases with z and $P_n^* = 1$ for the hardest "needle in a haystack" problem $z = 12$. For the hurdle problems, $0.2 \leq P_n^* \leq 0.4$ in Figure 4(a)-(d) where restart is not used. Using soft instead of strict Greedy lowers the optimal expected hitting time for every hurdle problem. Doing so has the same effect on the cosine trap problems, except for $z = 12$.[9] Adding restart P_r to the cosine trap problems reduces the optimal expected hitting time for the more difficult problems, not including $z = 12$. Restart has less effect on the hurdle problems, but reduces the hitting times significantly when $P_n \approx 0$. For all the problems, $P_r = 1/14$ increases the range of P_n-values that produce near-optimal expected hitting times and removes the asymptotic behavior for small P_n. Using $P_r = 1/14$ increases the optimal expected hitting time of the easiest cosine trap problems somewhat. Overall, however, these results suggest that there is an advantage to using $P_r > 0$ when problem difficulty is unknown.

It would be interesting to study in more detail the reduced impact of varying noise when restart is introduced, i.e., Figure 3(c) relative to Figure 3(d). Note, however, that the scale on the y-axis is logarithmic in both figures, so there is still a clear impact of varying noise in Figure 3(d). To further study this phenomenon, one could reduce restart probability to take place around approximately every $3n$ steps, inspired by Schöning's WalkSAT analysis [46].

7.2 Linear Cost Models for CFF

Goal. To what extent does the linear cost function g correspond to experimental results for feature selection applied to real-world datasets? We now study this question, using several datasets.

Method and Data. To validate the linear cost model $g(\boldsymbol{b})$ introduced in Section 5.2, we study three ML classifiers, namely Decision Tree (DT), Naive Bayes (NB), and Support Vector Machine (SVM). The datasets used are CIFAR-10 [23], gas-drift [2], and bioresponse [18]. For each dataset, each method is trained on 100 different feature subsets and the training time for each feature subset is recorded.

[6]https://scikit-learn.org/stable/

[7]https://www.hpc.ntnu.no/idun/

[8]Specifically, `Numpy.linalg.solve` was used here.

[9]However, it also increases P_n^* for the more difficult problems. Soft Greedy performs significantly worse than strict Greedy at lower values of P_n for problems where $z \geq 6$. We hypothesize that soft Greedy can escape the local maxima of the problem, but for the most difficult instances this makes SLS4CFF more likely to "climb" in the wrong direction, away from $\boldsymbol{b}^*$.

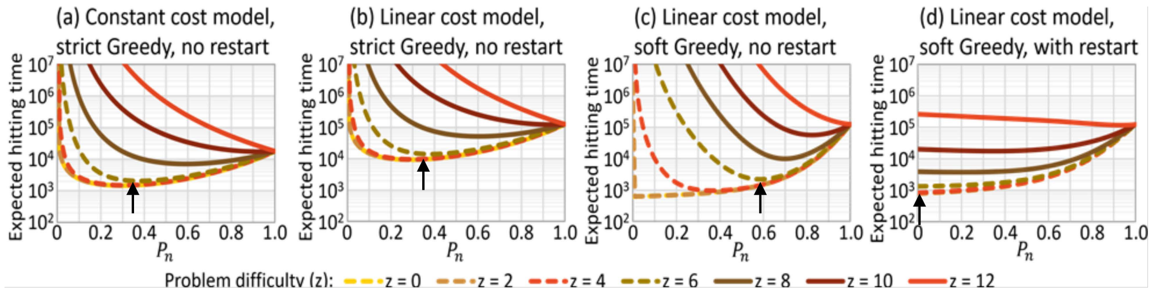

Figure 3: Expected hitting time (y-axis) as a function of noise probability P_n (x-axis) for four variants (a)–(d) of SLS4CFF on cosine trap problems of varying difficulty z. Optimum b^* has Hamming weight $w(b^*) = 5$. (a) is a baseline, namely the constant cost model of SoftSLS [31], while (b)–(d) show how the linear cost model interacts with different hyperparameters for SLS4CFF. The black arrows highlight, as an example, the optimal values for P_n for the graph where $z = 6$. Comparing (b) to (a) shows the impact of a constant [31] versus a linear cost model; (c) shows soft greedy's effect; and (d) shows restart's effect.

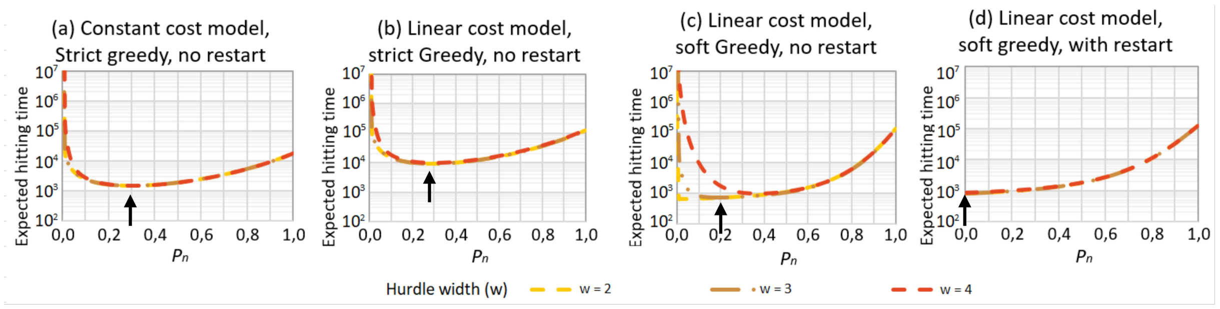

Figure 4: Expected hitting time (y-axis) as a function of noise probability P_n (x-axis) for four variants (a)–(d) of SLS4CFF for three hurdle problems with varying width w. The black arrows highlight, as an example, the optimal values for P_n for the graph where $w = 3$. In brief, (a) is a baseline, namely the constant cost model of SoftSLS [31], while (b)–(d) show how the linear cost model interacts with different hyperparameters for SLS4CFF. Other details are analogous to Figure 3.

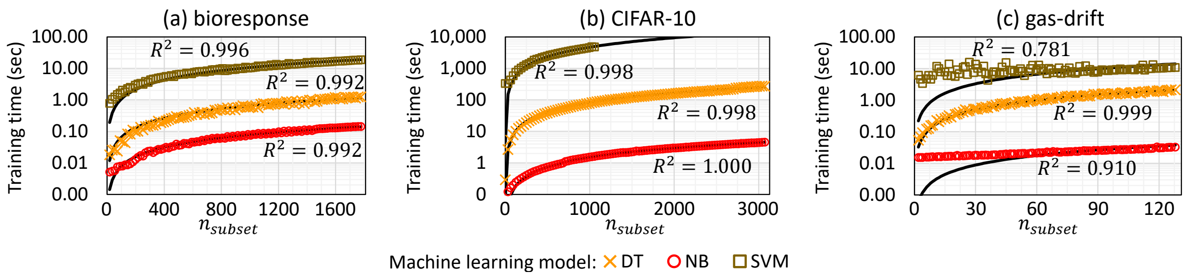

Figure 5: Training time (y-axis) as a function of the number of features (x-axis) for three ML models (DT, NB, and SVM) applied to three datasets. A linear regression model $y = ax$ (black solid line) has been fitted for each ML model for each dataset.

To select the feature subsets, we choose 100 values for the number of features n_{subset}, evenly distributed between 1 and the total number of features n. For each value, a subset of $n_{\text{subset}} = w(\boldsymbol{b})$ features is chosen uniformly at random, so that in $\boldsymbol{b}$ each feature b_i, where $1 \leq i \leq n$, is selected with probability n_{subset}/n.

Empirical cost models are created by using linear regression $y = ax + b$ with $b = 0$ to fit each dataset-method combination.[10] We then compare the empirical cost models to the actual training times of the ML models, which are implemented by scikit-learn [44] using the default parameter settings.

Results and Discussion. Figure 5 shows how training time increases with n_{subset}. The results show that 8 of the 9 dataset-model combinations (3 models, 3 datasets) have coefficients $R^2 \geq 0.91$. This suggests that the training time is well approximated by a linear regression model $y = ax + b$ with $a > 0, b = 0$, which corresponds to our linear model $g(\boldsymbol{b}) = \alpha w(\boldsymbol{b}) + \beta$ with $\alpha > 0, \beta = 0$. An exception is for SVM and gas-drift, where $R^2 = 0.781$, and a decent approximation of our linear model $y = ax$ only occurs as the number of included features increases. Overall, these results suggest that the linear cost function aligns well with experimental results for feature selection with real-world datasets.

[10] We used $b = 0$ for simplicity and consistency across these experiments; for the gas-drift dataset $b > 0$ may well improve the fit substantially.

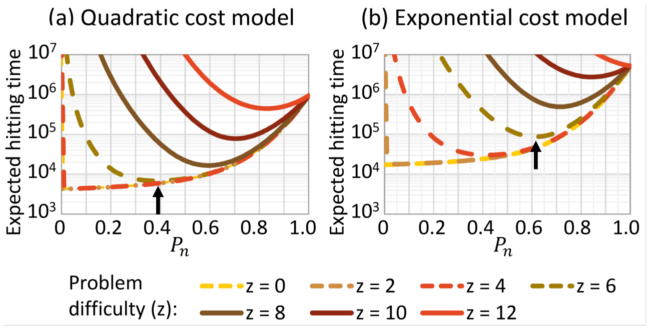

Figure 6: Expected hitting time (y-axis) as a function of noise probability P_n (x-axis) using (a) quadratic and (b) exponential cost functions g for SLS4CFF in cosine trap problems f of varying difficulty z. The black arrows show the optimal values for P_n for $z = 6$. SLS4CFF's hyperparameters and other settings are the same as in Figure 3(a) and Figure 3(b).

7.3 Varying Cost Models for CFF

Goal. Even though we used a linear cost function in Section 7.2, other cost functions g are of interest. How can expected hitting time for SLS4CFF be impacted when cost is higher?

Method and Data. The cosine trap problem from Section 5.1 is used for the analysis. We consider two complex cost functions for g, namely quadratic and exponential cost models, see Section 5.2. These cost functions are analyzed using the same parameters as in Figure 3(b), and in a similar fashion.

Results and Discussion. Figure 6 summarizes the results from the studies. As expected, expected hitting time increases, and it seems that optimal noise probability P_n^* increases sharply with the exponential cost model. As an example, for $z = 4$ the optimal noise level is $P_n^* \approx 0$ for the quadratic cost model, while it is $P_n^* \approx 0.4$ for the exponential cost model. For $z = 6$, which is highlighted in the figures, $P_n^* \approx 0.3$ for the linear case, $P_n^* \approx 0.4$ for the quadratic case, and $P_n^* \approx 0.6$ for the exponential case. Clearly, this illustrates how the optimal value P_n^* for P_n depends on the cost function g, and in particular how P_n^* increases to favor Noise over Greedy steps for a more extreme CFF.

7.4 The Compressed Markov Chain Model

Goal. The number of states n_S in the SLS4CFF search space grows exponentially with the bit-string length n so that $n_S = 2^n$. Unless we lump or compress the search space, Markov chain analysis of even medium sized problems would be computationally infeasible. How well does the proposed Markov chain compression method work for the cosine trap problem? How does it scale to models with $n \in \{8, 10, 12, 14, 16, \ldots\}$?

Method and Data. Let ω be the Hamming weight for the optimal state $\boldsymbol{b}^*$, $\omega = w(\boldsymbol{b}^*)$, and n_C the number of compressed states of the search space for a bit-string of length n. By varying n and ω we seek to construct compressed Markov chains. For each resulting Markov chain, we seek its size n_C.

Results and Discussion. The impact of compressing the search space into super states is illustrated in Figure 7. The table shows the number of compressed states and the compression (in %) for different combinations of bitstring lengths n and values for ω. The

n	n_S	$\omega = 3$		$\omega = 6$		$\omega = 9$		$\omega = 12$		$\omega = 15$		$\omega = 18$	
		n_C	%	n_C	%	n_C	%	n_C	%	n_C	%	n_C	%
5	32	12	37.50										
6	64	16	25.00	7	10.94								
8	256	24	9.38	21	8.20								
10	1024	32	3.13	35	3.42	20	1.95						
12	4096	40	0.98	49	1.20	40	0.98	13	0.32				
14	16384	48	0.29	63	0.38	60	0.37	39	0.24				
16	65536	56	0.09	77	0.12	80	0.12	65	0.10	32	0.05		
18	262144	64	0.02	91	0.03	100	0.04	91	0.03	64	0.02	19	0.01

Figure 7: How the number of super states n_C scales with the number of search space states n_S (or bitstring length n) and the number of set bits $\omega = w(\boldsymbol{b}^*)$ for optimum $\boldsymbol{b}^*$.

results are strong. For example, there are $n_C = 40$ compressed states at $n = 12$ bits (compared to $n_B = 4096$ regular states), so the compression is substantial. The high compression rate for larger problems enables us to analyze the most difficult cosine trap problems in Section 7.1 within a reasonable time.

8 DISCUSSION AND FUTURE WORK

Cost-aware and cost-constrained computation is becoming increasingly important [25, 29, 52], which is why we study costly fitness functions (CFFs) in this paper.

The critical reader may argue that hybrid methods similar to SLS4CFF are well-known, either in the form of memetic algorithms [39] or hybrid integrated filter and wrapper methods [32]. However, existing works do not provide an SLS method that is simultaneously analyzed by means of a lumped, scalable Markov chain approach that also considers computational cost. This coupling of an SLS algorithm with Markov chain analysis with computational cost integrated has, to our knowledge, not been performed prior to this paper. The lumped Markov chain model of SLS4CFF reduces the computation cost (time) for analyzing SLS4CFF's expected hitting time. In other words, the model makes it possible to analyze much larger problems in a reasonable amount of time, compared to directly running SLS4CFF simulations with costs.

These are a few areas for future work: First, it would be useful to further study the impact of varying a range of SLS4CFF hyperparameters, and include partial neighborhoods [32, 57]. Second, we plan to scale to larger problems, both in synthetic and natural settings. This would include investigation of other ML models (beyond Decision Tree, Naive Bayes, and Support Vector Machines), and even larger datasets with more features. Comprehensive studies can also be conducted to analyze the impact of varying SLS4CFF hyperparameters, when using these different models on real-world problems. Third, it would be interesting to automatically optimize hyperparameters of SLS4CFF [8, 31, 63] in a comprehensive way and for latent problem distributions, considering the varying computational costs of search steps for different computers and datasets. Ideally, in ML one would want to have a Markov chain hitting time analysis, developed along the lines suggested here, automatically and dynamically inform ML experiments in order to minimize cost while maximizing accuracy.

9 ACKNOWLEDGEMENTS

We would like to thank anonymous reviewers for their many helpful comments.

REFERENCES

[1] Alkasem, H. H. and Menai, M. E. B. (2021). A stochastic local search algorithm for the partial max-sat problem based on adaptive tuning and variable depth neighborhood search. *IEEE Access*, 9:49806–49843.

[2] Bache, K. and Lichman, M. (2013). UCI machine learning repository.

[3] Bian, C., Feng, C., Qian, C., and Yu, Y. (2020). An efficient evolutionary algorithm for subset selection with general cost constraints. In *Proc. AAAI*, pages 3267–3274.

[4] Coello, C. A. C., Lamont, G. B., and van Veldhuizen, D. A. (2006). *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation).* Springer-Verlag.

[5] DaCosta, L., Fialho, A., Schoenauer, M., and Sebag, M. (2008). Adaptive operator selection with dynamic multi-armed bandits. In *Proc. GECCO*, pages 913–920.

[6] Deb, K. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms.* John Wiley & Sons, Inc.

[7] Deb, K. and Goldberg, D. E. (1993). Analyzing deception in trap functions. In *Foundations of Genetic Algorithms*, pages 93 – 108.

[8] Ermon, S., Gomes, C. P., Sabharwal, A., and Selman, B. (2014). Designing fast absorbing Markov chains. In *Proc. AAAI*, pages 849–855.

[9] Falkner, S., Klein, A., and Hutter, F. (2018). BOHB: Robust and efficient hyperparameter optimization at scale. In *Proc. ICML*, pages 1436–1445.

[10] Gadat, S. and Younes, L. (2007). A stochastic algorithm for feature selection in pattern recognition. *JMLR*, 8:509–547.

[11] Grady, L. (2006). Random walks for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11):1768–1783.

[12] Guirguis, D., Aulig, N., Picelli, R., Zhu, B., Zhou, Y., Vicente, W., Iorio, F., Olhofer, M., Matusik, W., Coello Coello, C. A., and Saitou, K. (2020). Evolutionary black-box topology optimization: Challenges and promises. *IEEE Transactions on Evolutionary Computation*, 24(4):613–633.

[13] Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *JMLR*, 3:1157–1182.

[14] He, J. and Yao, X. (2003). Towards an analytic framework for analysing the computation time of evolutionary algorithms. *Artificial Intelligence*, 145(1-2):59–97.

[15] Hoos, H. H. (2002a). An adaptive noise mechanism for WalkSAT. In *Proc. AAAI*, pages 655–660.

[16] Hoos, H. H. (2002b). A mixture-model for the behaviour of SLS algorithms for SAT. In *Proc. AAAI*, pages 661–667.

[17] Hoos, H. H. and Stützle, T. (2005). *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, San Francisco.

[18] Ingelheim, B. (2012). Predicting a biological response.

[19] Karafotias, G., Hoogendoorn, M., and Eiben, A. E. (2015). Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, 19(2):167–187.

[20] Kask, K. and Dechter, R. (1999). Stochastic local search for Bayesian networks. In *Proc. AISTATS.*

[21] Kemeny, J. G. and Snell, J. L. (1976). *Finite Markov chains.* Springer-Verlag.

[22] Kohavi, R. and John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324.

[23] Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto.*

[24] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proc. NeurIPS*, pages 1097–1105.

[25] Lee, E. H., Eriksson, D., Perrone, V., and Seeger, M. W. (2021). A nonmyopic approach to cost-constrained Bayesian optimization. *CoRR*, abs/2106.06079.

[26] Liu, H., Simonyan, K., and Yang, Y. (2018). Darts: Differentiable architecture search. In *arXiv preprint arXiv:1806.09055.*

[27] Lou, Y., Yuen, S. Y., and Chen, G. (2021). Non-revisiting stochastic search revisited: Results, perspectives, and future directions. *Swarm and Evolutionary Computation*, 61.

[28] Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., and Banzhaf, W. (2019). NSGA-Net: Neural architecture search using multi-objective genetic algorithm. Proc. GECCO, pages 419-–427.

[29] Luong, P., Nguyen, D., Gupta, S., Rana, S., and Venkatesh, S. (2021). Adaptive cost-aware Bayesian optimization. *Knowledge-Based Systems*, 232:107481.

[30] Mengshoel, O. J. (2008). Understanding the role of noise in stochastic local search: Analysis and experiments. *Artificial Intelligence*, 172(8-9):955–990.

[31] Mengshoel, O. J., Ahres, Y., and Yu, T. (2016). Markov chain analysis of noise and restart in stochastic local search. In *Proc. IJCAI*, pages 639–646.

[32] Mengshoel, O. J., Flogard, E., Riege, J., and Yu, T. (2021a). Stochastic local search heuristics for efficient feature selection: An experimental study. In *Proc. NIKT*, pages 58–71.

[33] Mengshoel, O. J., Roth, D., and Wilkins, D. C. (2011a). Portfolios in stochastic local search: Efficiently computing most probable explanations in Bayesian networks. *Journal of Automated Reasoning*, 46(2):103–160.

[34] Mengshoel, O. J., Wilkins, D. C., and Roth, D. (2011b). Initialization and restart in stochastic local search: Computing a most probable explanation in Bayesian networks. *IEEE Transactions on Knowledge and Data Engineering*, 23(2):235–247.

[35] Mengshoel, O. J., Yu, T., Riege, J., and Flogard, E. (2021b). Stochastic local search for efficient hybrid feature selection. In *Proc. GECCO*, pages 133-–134.

[36] Nagata, Y. (2018). Random partial neighborhood search for the post-enrollment course timetabling problem. *Computers & Operations Research*, 90:84–96.

[37] Needell, D. and Tropp, J. A. (2009). CoSaMP: Iterative signal recovery from incomplete and inaccurate samples. *Applied and Computational Harmonic Analysis*, 26(3):301–321.

[38] Neumann, F. and Wegener, I. (2007). Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. *Theoretical Computer Science*, 378(1):32 – 40.

[39] Nguyen, P. T. H. and Sudholt, D. (2020). Memetic algorithms outperform evolutionary algorithms in multimodal optimisation. *Artificial Intelligence*, 287:103345.

[40] Nikolaev, A. G. and Jacobson, S. H. (2011). Using Markov chains to analyze the effectiveness of local search algorithms. *Discrete Optimization*, 8(2):160 – 173.

[41] Ochoa, G. and Veerapen, N. (2016). Deconstructing the big valley search space hypothesis. In *Evolutionary Computation in Combinatorial Optimization*, pages 58–73.

[42] Pal, D. K. and Mengshoel, O. J. (2016). Stochastic CoSaMP: Randomizing greedy pursuit for sparse signal recovery. In *Proc. ECML-PKDD*, pages 761–776.

[43] Pan, J.-Y., Yang, H.-J., Faloutsos, C., and Duygulu, P. (2004). Automatic multimedia cross-modal correlation discovery. In *Proc. KDD*, pages 653-–658.

[44] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *JMLR*, 12:2825–2830.

[45] Prügel-Bennett, A. (2004). When a genetic algorithm outperforms hill-climbing. *Theoretical Computer Science*, 320(1):135 – 153.

[46] Schoning, U. (1999). A probabilistic algorithm for k-sat and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science.*

[47] Schumann, R., Mou, L., Lu, Y., Vechtomova, O., and Markert, K. (2020). Discrete optimization for unsupervised sentence summarization with word-level extraction. In *Proc. ACL*, pages 5032–5042.

[48] Selman, B., Kautz, H., et al. (1993). Domain-independent extensions to gsat: Solving large structured satisfiability problems. In *Proc. IJCAI*, pages 290–295.

[49] Selman, B., Levesque, H., and Mitchell, D. (1992). A new method for solving hard satisfiability problems. In *Proc. AAAI*, pages 440–446.

[50] Shang, K., Ishibuchi, H., and Chen, W. (2021). Greedy approximated hypervolume subset selection for many-objective optimization. In *Proc. GECCO*, pages 448–456.

[51] Shi, L. and Rasheed, K. (2010). A survey of fitness approximation methods applied in evolutionary algorithms. In Tenne, Y. and Goh, C.-K., editors, *Computational Intelligence in Expensive Optimization Problems*, pages 3–28. Springer Verlag.

[52] Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. In *Proc. NeurIPS*, pages 2951–2959.

[53] Spears, W. M. (1999). Aggregating models of evolutionary algorithms. In *Proc. CEC*, volume 1, pages 631–638.

[54] Spears, W. M. and De Jong, K. A. (1996). Analyzing GAs using Markov models with semantically ordered and lumped states. In *Proc. FOGA*, pages 85–100.

[55] Srinivas, N., Krause, A., Kakade, S., and Seeger, M. (2010). Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proc. ICML*, pages 1015-–1022.

[56] Suzuki, J. (1995). A Markov chain analysis on simple genetic algorithms. *IEEE Trans. Systems, Man, and Cybernetics*, 25(4):655–659.

[57] Tari, S., Basseur, M., and Goëffon, A. (2021). Partial neighborhood local searches. *International Transactions in Operational Research*, pages 2761–2788.

[58] Wang, S., Ding, Z., and Fu, Y. (2017). Feature selection guided auto-encoder. In *Proc. AAAI*, pages 2725–2731.

[59] Weise, T., Wu, Z., and Wagner, M. (2019). An improved generic bet-and-run strategy for speeding up stochastic local search. In *Proc. AAAI*, pages 2395–2402.

[60] White, C., Nolen, S., and Savani, Y. (2021). Exploring the loss landscape in neural architecture search. In *Proc. UAI*, pages 654–664.

[61] Xia, F., Liu, J., Nie, H., Fu, Y., Wan, L., and Kong, X. (2020). Random walks: A review of algorithms and applications. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 4(2):95–107.

[62] Yolcu, E. and Póczos, B. (2019). Learning local search heuristics for boolean satisfiability. In *Proc. NeurIPS*, pages 7990–8001.

[63] Yu, T., Kveton, B., and Mengshoel, O. J. (2017). Thompson sampling for optimizing stochastic local search. In *Proc. ECML-PKDD*, pages 493–510.

[64] Yu, Y. and Zhou, Z.-H. (2008). A new approach to estimating the expected first hitting time of evolutionary algorithms. *Artificial Intelligence*, 172(15):1809 – 1832.

[65] Yuen, S. and Cheung, B. (2006). Bounds for probability of success of classical genetic algorithm based on Hamming distance. *IEEE Transactions on Evolutionary Computation*, 10:1–18.

[66] Yuen, S. Y. and Chow, C. K. (2007). A non-revisiting genetic algorithm. In *Proc. CEC*, pages 4583–4590.

[67] Zavala, G. R., Nebro, A. J., Luna, F., and Coello, C. A. C. (2014). A survey of multi-objective metaheuristics applied to structural optimization. *Structural and Multidisciplinary Optimization*, 49(4):537–558.