Steepest ascent can be exponential in bounded treewidth problems

David A. Cohen[1], Martin C. Cooper[2], Artem Kaznatcheev[3,4], and Mark Wallace[5]

[1]Department of Computer Science, Royal Holloway University of London, Egham, UK
[2]IRIT, University of Toulouse III, Toulouse, France
[3]Department of Computer Science, University of Oxford, Oxford, UK
[4]Department of Translational Hematology & Oncology Research, Cleveland Clinic, Cleveland, USA
[5]Faculty of Information Technology, Monash University, Melbourne, Australia

Abstract

We investigate the complexity of local search based on steepest ascent. We show that even when all variables have domains of size two and the underlying constraint graph of variable interactions has bounded treewidth (in our construction, treewidth 7), there are fitness landscapes for which an exponential number of steps may be required to reach a local optimum. This is an improvement on prior recursive constructions of long steepest ascents, which we prove to need constraint graphs of unbounded treewidth.

1 Introduction

We are interested in the long-term behaviour of steepest-ascent local search in optimisation problems over finite domains. Local search is an important tool in solving optimisation problems when exhaustive search algorithms, such as branch-and-bound, are not a practical possibility due to the size of the search space.

It is therefore important to gain understanding of the behaviour of local search and, in particular, the number of steps required to reach a local optimum. If local search can continue for a number of steps which is exponential in the number of variables, then in practice we can consider that it may never reach a stable state within a reasonable timescale. This failure to equilibrate is especially important to understand for commonly used greedy local search heuristics like steepest ascent.

Algorithmically, steepest ascent proceeds by a sequence of steps towards a local maximum of the objective function. At each step, a best move is chosen to transform the current state into a better neighbouring state. In this paper we restrict our attention to the simplest case in which all variables are Boolean and the set of possible moves from a given state is just the flipping of one variable.

Throughout the paper, we will refer to the objective function being maximized as the fitness function. We choose this terminology due to the central role that local search has played in the study of biological evolution. This has been the case ever since Wright [12] introduced the idea of viewing the evolution of populations of organisms as a local search process over a space of possible genotypes with associated fitness values that became known as a "fitness landscape." For over 80 years since Wright's introduction of fitness landscapes, it has been assumed that if the fitness function is not time-varying (i.e. if we have a static fitness landscape) then typical populations will quickly evolve to a local fitness peak. Recently, Kaznatcheev [4] has challenged this assumption by showing biologically reasonable families of landscapes where a local peak cannot be found in time polynomial in the number of genes (variables). On such "hard" fitness landscapes, we can consider local peaks as practically unreachable. This situation can be seen as open-ended evolution [11].

In the case of biological evolution, the particular local search algorithm that is implemented can depend on the details of the particular population structure, genetic architecture, and mutation rate [4]. However, in the limit of large populations with small mutations rates, the evolutionary dynamics are often modeled as steepest ascent. Thus, rigorously analyzing steepest ascent can help us better understand biological evolution. Of course, biology is not the only domain where local search matters. Similar models are used in fields like business operation & innovation theory [6, 8], physics, and economics [9].

1.1 Soft constraints and fitness functions

A discrete optimisation problem can be formalised as a set of functions on subsets of its variables. The value of the objective function is the sum of the values of these functions. In the generic framework known as the VCSP (Valued Constraint Satisfaction Problem), each function is known as a "soft constraint" on its variables [7]. The "scope" of a soft constraint is its set of variables; each combination of values assigned to these variables is mapped to a value by the soft constraint. The "arity" of a constraint is the number of variables in its scope. The "degree" of a variable is the number of other variables that occur with it in some constraint scope. In the formulation of the VCSP considered in this paper the objective function is to be maximised.

Biologically a combination of gene expressions underlies a phenotypic trait, and a combination of its phenotypic traits underlie the fitness. Put very simply, fitness is the sum of the values of its phenotypic traits. We can assign this same value to the combination of gene expressions which underly the phenotype traits. The number of genes involved in a single phenotypic trait is assumed to be reasonably small – say less than some fixed number $\mathtt{K}+1$ (the amount of epistasis), and any one gene is involved in a reasonably small number of phenotypic traits – say some fixed number $\mathtt{P}$ (the amount of pleiotropy).

A VCSP is a model of fitness if we associate a variable with each gene, which takes the value 1 if the gene is expressed. Each soft constraint over a set of variables, models the set of genes underlying a phenotypic trait, and the value of the soft constraint represents the value of the phenotypic trait. The bound $\mathtt{K}+1$ on the number of genes underlying a phenotypic trait is modelled by a bound on the arity of the soft constraints. The bound $\mathtt{P}$ on the number of different phenotypes involving one gene is modelled by a bound on the degree of any variable in the VCSP.

In this paper we only consider functions expressible as a set of soft constraints with bounded arity (our examples all have arity at most eight). The *constraint graph* is the graph whose vertices are the variables and with an edge between two variables if there is a soft constraint between them (i.e. they both belong to the scope of some soft constraint). Given a constraint graph $\mathtt{G}$, and a variable index $\mathtt{i}$, we will let $\mathtt{d_G(i)}$ be the *degree of* $\mathtt{i}$ *in* $\mathtt{G}$: i.e. the total number of other variables that co-occur with $\mathtt{x_i}$ in the scope of some constraint in the VCSP.

1.2 Bounded treewidth and local search

We are particularly interested in the behaviour of local search when the constraint graph has bounded treewidth. When treewidth is bounded, it is well known that it is possible to find a global optimum of a VCSP in polynomial time [1], but this does not inform us on the complexity of local search even for the simpler problem of reaching a local maximum. A major drawback of local search as a technique for solving optimisation problems is the possibility of there being an exponential number of local maxima whose value may even be exponentially smaller than the value of the global maximum. It is not difficult to show that this phenomenon can occur when treewidth is bounded. Consider, for example, the global objective function

$$F(x) \;=\; \sum_{i=1}^{N/2} f(x_{2i-1}, x_{2i}) \qquad (1)$$

where

$$f(a,b) = \begin{cases} \mathtt{1} & \text{if } \mathtt{a}=\mathtt{b}=\mathtt{0} \\ \alpha & \text{if } \mathtt{a}=\mathtt{b}=\mathtt{1} \\ \mathtt{0} & \text{otherwise} \end{cases} \qquad (2)$$

where $\mathtt{x} = \langle \mathtt{x_1}, \ldots, \mathtt{x_N} \rangle \in \{\mathtt{0},\mathtt{1}\}^{\mathtt{N}}$, and $\mathtt{N}$ is even. Each of the tuples consisting of repeated bits (i.e. $\mathtt{x_{2i-1}} = \mathtt{x_{2i}}$ for $\mathtt{i} = \mathtt{1}, \ldots, \mathtt{N/2}$) is a local maximum. Furthermore, the ratio between the value of the global maximum and the value of the worst local maximum is α and hence is exponential if $\alpha = 2^{\mathtt{N}}$. The constraint graph is a forest (consisting of $\mathtt{N/2}$ unconnected edges) and hence has treewidth 1.

Thus bounded treewidth does not alleviate the phenomenon of multi-modality. The possibility of exponential complexity of local search to reach a local maximum, which is the topic of this paper, is an independent phenomenon which, as we have seen, is of interest in genetics as well as computer science.

1.3 Summary

We show that local search following a steepest ascent can take an exponential number of steps to reach a local optimum even when (a) the variables are Boolean, (b) the problem is formalised using bounded arity soft constraints, and (c) the constraint graph has bounded treewidth. Previously, classes of objective functions have been described for which steepest ascent may require an exponential number of variable flips to reach a local maximum [3, 4] but the fitness functions were defined recursively and not expressed as the sum of bounded-arity functions. We first show in the next section that these examples cannot be expressed by soft constraints whose constraint graph has bounded treewidth. In related work, Kaznatcheev, Cohen, and Jeavons [5] have studied the complementary problem of guaranteeing short paths to a local optimum for local search which performs an arbitrary improving flip at each step rather than a steepest ascent. In the process, they have described some bounded treewidth examples where some exponentially long improving paths exist but these are not the paths followed by steepest ascent.

2 Earlier recursive construction of hard fitness landscapes

We are not the first to consider long steepest-ascent walks in fitness landscapes. Two constructions from the literature [3, 4] of fitness landscapes on $\mathtt{n}$ variables where steepest ascent requires of the order of $2^{\mathtt{n}}$ steps are particularly relevant. However, both these landscapes were specified recursively and not by concrete VCSP instances. In this section, we give a simpler presentation of the recursive definition of winding landscapes (Definition 1) that generalises both the construction in [3] and [4]. We then prove that if these winding landscapes were implemented by a VCSP then the fitness graph would have unbounded treewidth (Corollary 4). This will motivate consideration of a different form of fitness landscape in Section 3.

2.1 Recursive construction

Given a fitness landscape on $\{0,1\}^{2n}$, we refer to the hypercube $\{0,1\}^{2k}$ on the first $2k$ variables as a sub-cube.

Definition 1. *Suppose we are given a sequence of fittest steps $s_k^+ > 0$ and barrier steps s_k^- for $0 \leq k \leq n$ satisfying the conditions $s_k^+ > s_{k+1}^-$, $s_k^- < s_k^+ < s_{k+1}^+$. Consider the recursive construction of the functions $f^k\{0,1\}^{2k} \to \mathbb{R}$ with sub-cube optima $x_k^* = 0^{2(k-1)}11$ for $0 \leq k \leq n$ (where $x_0^* = \epsilon$, the empty string):*

$$f^0(\epsilon) = 0 \tag{3}$$

$$f^{k+1}(xab) = \begin{cases} f^k(x) & \text{if } a = b = 0 \\ f^k(x) + s_{k+1}^- & \text{if } a \neq b \ \& \ x \neq x_k^* \\ f^k(x_k^*) + s_{k+1}^- & \text{if } a = 0, b = 1 \ \& \ x = x_k^* \\ f^k(x_k^*) + s_{k+1}^+ & \text{if } a = 1, b = 0 \ \& \ x = x_k^* \\ f^k(x \oplus x_k^*) + f^k(x_k^*) + 2s_{k+1}^+ & \text{if } a = b = 1 \end{cases} \tag{4}$$

We define the (nth) winding landscape *as $f = f^n$.*

Here $x \oplus x_k^*$ means the bit-wise XOR of x and x_k^* (i.e. $[x \oplus x_k^*]_i = [x]_i + [x_k^*]_i \mod 2$).

Based on this definition, if the steepest-ascent path takes T_k steps to reach x_k^* from 0^{2k} in f^k then steepest-ascent will take the following path from 0^{2k} to x_{k+1}^* in f^{k+1} (see Appendix C.2 of [4] for the proof):

$$0^{2(k+1)} \to^{T_k} x_k^*00 \to x_k^*10 \to x_k^*11 \to^{T_k} 0^{2k}11 = x_{k+1}^* \tag{5}$$

which has a length of $T_{k+1} = 2T_k + 2$ steps. Solving this recurrence equation with $T_0 = 0$, we get $T_n = 2^{n+1} - 2$.

On the one hand, if $s_k^- \leq 0$ for all $0 \leq k \leq n$ then Definition 1 implements Horn, Goldberg, and Deb [3]'s *Root2path* construction and the long path from 0^{2n} is not only the steepest ascent but also the only ascent. On the other hand, if $s_k^- > 0$ for all $0 \leq k \leq n$ then Definition 1 implements Kaznatcheev [4]'s winding semismooth fitness landscape that has no reciprocal sign epistasis but still maintains the long path from 0^{2n} as the steepest ascent. This semismooth case is of interest because a short ascent exists from each variable assignment to the unique fitness peak but steepest ascent does not find this short ascent.

2.2 Lower bounds on the constraint-graph

The winding fitness landscape has the dramatic property of having drastic changes in the direction and magnitude of the gradient of the objective function between points which are very close to each other. Consider the sub-cube spanned by the first $2(k+1)$-variables. The sub-cube fitness maximum is at $x_k^* = 0^{2k}11$. If $s_{k+1}^- > 0$ (as in [4]) then the sub-cube fitness minimum is only Hamming-distance 2 away at $0^{2(k+1)}$: so the flows (i.e. differences in the objective function from the current point to its neighbouring points) change from all positive to all negative in just 2 steps. From this we can prove that the total scope size of any constraint graph implementing this fitness landscape must be high. The case in which we do not necessarily have $s_{k+1}^- > 0$ (as in [3]) is slightly more complicated since $0^{2(k+1)}$ is no longer a fitness minimum and mostly has negative flows. However, these negative flows have small magnitude compared to the very large magnitude negative flows at x_k^*, so a similar argument can be used. We formalise this argument below.

For convenience, let $||x||_0$ be the number of non-zero entries in x – also known as the Hamming weight or the zero-'norm' of the vector x. And let us use $x[i \to b]$ to mean a bit-string that is the same as x at every bit, except the i-th bit is set to b. Or, in symbols: $\forall j \neq i \quad [x[i \to b]]_j = [x]_j$ and $[x[i \to b]]_i = b$. This allows us to define the *gradient* ∇f of a fitness function f entry-wise as $[\nabla f(x)]_i = f(x[i \to 1]) - f(x[i \to 0])$ to state our degree lower-bound lemma:

Lemma 2. *Given a fitness function f implemented by a VCSP with constraint graph G and any two distinct variable assignments x and y that differ on a set of variables S, we have that the total degree $d_G(S) = \sum_{i \in S} d_G(i)$ of S in G is lower-bounded by the change in flow: $d_G(S) \geq ||\nabla f x - \nabla f y||_0$.*

Proof. If we look at a variable at position i and compare $\nabla f(x[i \to 1])$ to $\nabla f(x[i \to 0])$ then any differences in the gradient must have been due solely to the change in variable x_i. Thus, given any position j such that $[\nabla f(x[i \to 1])]_j \neq [\nabla f(x[i \to 0])]_j$ there must be a constraint that has both i and j (and maybe others) in its scope. Thus, by looking at the number of non-zero entries in $\nabla f(x[i \to 1]) - \nabla f(x[i \to 0])$, we get a lower bound on the number of other variables with which each variable i co-occurs in a constraint.

This reasoning can be extended over paths between non-adjacent states. Suppose we have two states x_1 and x_t at Hamming distance t from each other. Let $x_1 x_2 ... x_{t-1} x_k$ be any shortest path between them with the bits flipped at each step given by $i_1, i_2, ... i_{t-1}$. Notice the following:

$$\begin{aligned} ||\nabla f x_1 - \nabla f x_k||_0 &= ||(\nabla f x_1 - \nabla f x_2) + (\nabla f x_2 - \nabla f x_3) + \\ &\quad \cdots + (\nabla f x_{t-1} - \nabla f x_t)||_0 \qquad (6) \\ &\leq ||\nabla f x_1 - \nabla f x_2||_0 + ||\nabla f x_2 - \nabla f x_3||_0 + \\ &\quad \cdots + ||\nabla f x_{t-1} - \nabla f x_t)||_0 \qquad (7) \end{aligned}$$

In words: given two states x_1 and x_k that differ at a set of variables S, the total number of variables that the variables in S co-occur with is lower-bounded by $||\nabla f x_1 - \nabla f x_k||_0$. □

Now, we can apply this lower bound technique to the winding fitness landscape.

Proposition 3. *If the winding fitness landscape f on $2n$ variables from Definition 1 is implemented by a VCSP with constraint graph G then $d_G(2k+1) + d_G(2k+2) \geq k$ for each $0 \leq k < n$.*

Proof. Let us look at the gradients at the path's starting point:

$$\nabla f(0^{2n}) = [s_1^+, s_1^-, s_2^-, s_2^-, ..., s_i^-, s_i^-, ..., s_n^-, s_n^-] \quad (8)$$

and for each $\mathtt{1} \leq \mathtt{k} \leq \mathtt{n}$, look at the gradients at subcube peaks $\nabla \mathtt{f}(0^{2(\mathtt{k}-1)}(\mathtt{11})0^{2(\mathtt{n}-\mathtt{k})})$: they have a slightly more complicated form, so we define them point-wise for $\mathtt{i} \in [\mathtt{1}, \mathtt{n}]$, $\mathtt{b} \in \{\mathtt{0}, \mathtt{1}\}$, and $\mathtt{x} = 0^{2(\mathtt{k}-1)}(\mathtt{11})0^{2(\mathtt{n}-\mathtt{k})}$:

$$[\nabla f(x)]_{2i-b} = \begin{cases} -\mathtt{s}_\mathtt{i}^+ + \mathtt{b}(\mathtt{s}_\mathtt{i}^- - \mathtt{s}_\mathtt{i}^+) & \text{if } \mathtt{i} < \mathtt{k} \\ \mathtt{f}^\mathtt{k}(\mathtt{x}_\mathtt{k}^*) - \mathtt{s}_\mathtt{k}^- & \text{if } \mathtt{i} = \mathtt{k} \\ \mathtt{s}_{\mathtt{k}+\mathtt{1}}^+ & \text{if } \mathtt{i} = \mathtt{k}+\mathtt{1} \ \&\ \mathtt{b} = \mathtt{1} \\ \mathtt{s}_\mathtt{i}^- & \text{if } \mathtt{i} > \mathtt{k}+\mathtt{b} \end{cases} \quad (9)$$

Looking at the odd entries lower than $\mathtt{2k}$ (i.e. $\mathtt{i} < \mathtt{k}, \mathtt{b} = \mathtt{1}$ in equation 9), we have

$$[\nabla f(0^{2(k-1)}(11)0^{2(n-k)}) - \nabla f(0^{2n})]_{2i-1} = -2s_i^+ \neq 0 \quad .$$

Thus by equation 7, the variables at positions $\mathtt{2k}+\mathtt{1}$ and $\mathtt{2k}+\mathtt{2}$ together have degree of at least $\mathtt{k}$. □

Corollary 4. *Any VCSP implementing the winding fitness landscape from Definition 1 must have a constraint graph that is dense and with unbounded treewidth.*

Proof. Summing up over all $\mathtt{1} \leq \mathtt{k} \leq \mathtt{n}$, we get that any VCSP instance that implements $\mathtt{f}$ must have total degree of at least $(\mathtt{n}-\mathtt{1})\mathtt{n}/\mathtt{2}$ (i.e. quadratic in the number $\mathtt{2n}$ of variables).

In particular, this means that a constraint graph of bounded treewidth (which would have total degree linear in $\mathtt{2n}$) cannot implement the winding fitness landscape $\mathtt{f}$. □

In the following section we avoid the issue raised by Corollary 4 by using an explicit construction based on a low-treewidth constraint graph.

3 Exponential local search with bounded treewidth

We seek a simple model whose $\mathtt{n}$ variables are all Boolean (with domain $\{\mathtt{0}, \mathtt{1}\}$), for which steepest ascent local search, by flipping the value of a variable at each step, has an exponentially long ascending sequence. In the model we construct, the soft constraints will have scopes of arity eight, namely the variables with index $\mathtt{4i}, \ldots, \mathtt{4i}+\mathtt{7}$. The constructed model has a constraint graph of treewidth seven.

In order to better explain the judgments made in constructing our novel fitness landscape, we build the model in a series of stages. At each stage we modify or extend the set of domains, variables and soft constraints from the previous stage so as to reach the final model which satisfies the above conditions.

3.1 Stage 1: Counting

Our initial model will have $\mathtt{N}$ variables denoted $\mathtt{X}_\mathtt{N}, \ldots, \mathtt{X}_\mathtt{1}$, which will all have the same finite domain including the values 0 and 1.

We will denote a state, i.e. an assignment of the $\mathtt{N}$ variables, as an array of their values with the value of $\mathtt{X}_\mathtt{1}$ on the right. Thus the assignment $\mathtt{X}_\mathtt{1} = \mathtt{1}, \mathtt{X}_\mathtt{2} = \mathtt{0}, \mathtt{X}_\mathtt{3} = \mathtt{0}$ will be denoted by the array $\langle \mathtt{0}, \mathtt{0}, \mathtt{1} \rangle$. A local search path can thus be represented as a sequence of arrays. The final exponential-length ascending path will include as a subsequence an encoding of the standard Boolean *counting* sequence $\mathtt{0}, \mathtt{1}, \mathtt{10}, \mathtt{11}, \mathtt{100}, \mathtt{101}, \ldots$ or, more precisely:

$$0^N, 0^{N-1}1, 0^{N-2}10, 0^{N-2}11, 0^{N-3}100, 0^{N-3}101, \ldots \quad (10)$$

which is highlighted in all of the explicit examples.

Since the final model will have $\mathtt{O(N)}$ Boolean variables this will be a sub-sequence of exponential length.

3.2 Stage 2: Two new domain values

The values $\mathtt{0}$ and $\mathtt{1}$ alone do not enable us to write down a sequence of arrays of increasing value reached by local search. We only allow a local search step to flip a single variable value. Since many of the steps in the binary counting sequence above alter the value of many more than one variable, they cannot be steps in our final ascent.

To fix this we introduce a *carry* symbol, $\mathtt{C}$. The idea is that carry represents the fact that we have arrived at $\mathtt{S01}^\mathtt{i}$, where $\mathtt{S}$ is some arbitrary prefix of length $\mathtt{N}-\mathtt{i}-\mathtt{1}$. We first flip the rightmost $\mathtt{1}$ to be the new carry symbol. We then propagate this carry symbol to the left so long as it is preceded by a $\mathtt{1}$. Each propagation first yields two adjacent carry symbols, and then we replace the right hand carry with a $\mathtt{0}$. Eventually we have only one carry symbol and it has a $\mathtt{0}$ to its left. Now we flip this $\mathtt{0}$, replacing $\mathtt{0C}$ with a $\mathtt{1C}$, and we arrive at $\mathtt{S1C0}^{\mathtt{i}-\mathtt{1}}$. We can now flip the final $\mathtt{C}$ into a $\mathtt{0}$ and we have successfully performed a carry. For example, to count from $[\mathtt{15}]_2 = \mathtt{1111}$ to $[\mathtt{16}]_2 = \mathtt{10000}$:

1:	0	0	0	1	1	1	1
2:	0	0	0	1	1	1	C
3:	0	0	0	1	1	C	C
4:	0	0	0	1	1	C	0
5:	0	0	0	1	C	C	0
6:	0	0	0	1	C	0	0
7:	0	0	0	C	C	0	0
8:	0	0	0	C	0	0	0
9:	0	0	1	C	0	0	0
10:	0	0	1	0	0	0	0

In this sequence only one variable is flipped at each step. But note that the pair $\mathtt{1C}$ changes in different ways. At lines 2, 4 and 6 $\mathtt{1C}$ flips to $\mathtt{CC}$. However at line 9 $\mathtt{1C}$ flips to $\mathtt{10}$. This is because the carry symbol appears in two contexts. We introduce $\mathtt{C}$ to indicate a carry, but when its

job is done and the 0 on its left has become a 1, the carry symbol needs to be removed.

So far, the construction is equivalent to Kaznatcheev, Cohen, and Jeavons [5]'s Example 2 and with the right choice of soft-constraints is sufficient to show that some fitness increasing path has exponential length. But this needs further elaboration to make the *steepest* ascent exponentially long and reduce domain size to 2.

To eliminate the dual role of carry as add and remove, we introduce a fourth symbol X and put extra steps to enable the transition from state 8 to state 9. When the 0C should become 1C, we first move to XC, then remove the carry symbol, and then flip the X to a 1.

8:	0	0	0	C	0	0	0
9a:	0	0	X	C	0	0	0
9b:	0	0	X	0	0	0	0
10:	0	0	1	0	0	0	0

The admissible configurations of these symbols can be summarised in as the six cases in equation 11:

$$\begin{array}{ccc} \{01\}+ & \{01\}^*1C\{01\}^*0 & \{01\}^*0C\{01\}^*0 \\ \{01\}^*CC\{01\}^*0 & 0^*XC\{01\}^*0 & 0^*X\{01\}^*0 \end{array} \quad (11)$$

A VCSP will be designed so that only admissible configurations of these symbols will be reached by steepest ascent from 0^N.

3.3 Stage 3: New intermediate symbols

Even with our new symbols C and X, the sequence we proposed in the previous section cannot correspond to a steepest ascent. Simply flipping the most significant variable from 0 to 1 (so 0^N becomes 10^{N-1}) jumps directly from the start the the end of the sequence.

It is necessary to ensure that a state s cannot be a neighbour of a previous state s' (apart from its immediate predecessor in the path) otherwise the steepest-ascent would take the short-cut from s' to s. In order to constrain and prioritise moves we introduce an intermediate symbol between each pair of symbols. It thus requires two steps to change any of the main symbols, $0, 1, C$ and X to another main symbol, via an intermediate symbol.

To encode all (4 main and 6 intermediate) symbols we will use four boolean variables. The encoding, and the definition of the soft constraints, will ensure that a flip of a boolean variable which increases the objective (a) can only yield boolean sequences that encode (main or intermediate) symbols; (b) cannot change a main to a main symbol; (c) cannot change an intermediate to an intermediate symbol; and, (d) can only change a main to an intermediate symbol if there currently are no intermediate symbols (i.e. all flips from intermediate symbols to main symbols are better than any move from a main symbol to an intermediate symbol)

Each intermediate symbol can be chosen to be one flip away from exactly two main symbols: it falls therefore "between" the two main symbols. Consequently, when an intermediate symbol is flipped during a sequence of local moves, in order to improve again, it must move forward to the other main symbol.

We build the set of (six) intermediate symbols using a function symbol i where $i_{ab} = i_{ba}$ is the intermediate symbol between main symbols a and b. We denote the set of symbols:

$$Sym = \{0, 1, C, X, i_{01}, i_{C,0}, i_{0,X}, i_{1,C}, i_{X,1}, i_{C,X}\} \quad (12)$$

Boolean encoding of main and intermediate symbols. The explicit encoding using the four Boolean variables $x_{1,i}, x_{2,i}, x_{3,i}, x_{4,i}$ is given in the following table.

X_i	$x_{1,i}$	$x_{2,i}$	$x_{3,i}$	$x_{4,i}$
0	1	0	0	0
1	0	1	0	0
C	0	0	1	0
X	0	0	0	1
i_{01}	1	1	0	0
i_{C0}	1	0	1	0
i_{0X}	1	0	0	1
i_{1C}	0	1	1	0
i_{X1}	0	1	0	1
i_{CX}	0	0	1	1

Each intermediate symbol has two Boolean variables set to 1, and each original symbol corresponds to a pattern with just one Boolean variable set to 1. The row $\langle 0, 0, 0, 0\rangle$ and any row with more than two 1's represents a non-symbol.

The intermediate symbols mean that the example of a steepest-ascent path will now take the following form (where each symbol would now be represented by four Boolean variables):

1:	0	0	0	1	1	1	1
	0	0	0	1	1	1	i_{1C}
2:	0	0	0	1	1	1	C
	0	0	0	1	1	i_{1C}	C
3:	0	0	0	1	1	C	C
	0	0	0	1	1	C	i_{C0}
4:	0	0	0	1	1	C	0
	0	0	0	1	i_{1C}	C	0
5:	0	0	0	1	C	C	0
	0	0	0	1	C	i_{C0}	0
6:	0	0	0	1	C	0	0
	0	0	0	i_{1C}	C	0	0
7:	0	0	0	C	C	0	0
	0	0	0	C	i_{C0}	0	0
8:	0	0	0	C	0	0	0
	0	0	i_{0X}	C	0	0	0
9a:	0	0	X	C	0	0	0
	0	0	X	i_{C0}	0	0	0
9b:	0	0	X	0	0	0	0
	0	0	i_{X1}	0	0	0	0
10:	0	0	1	0	0	0	0

There are now more possible configurations than those six that appear in equation 11. The additional admissible configurations that include intermediate symbols are $\{01\}^+i_{01}$, $\{01\}^+i_{1C}$, $\{01\}^+i_{1C}C\{01\}^*$, $0^*i_{0X}C\{01\}^*$, $\{01\}^+Ci_{C0}\{01\}^*$, $0^*Xi_{C0}\{01\}^*$. The VCSP soft constraints will be designed so that only these "intermediate" configurations will occur on the steepest ascent from 0^N.

3.4 Encoding using Soft Constraints

Ensuring that intermediate symbols only occur in admissible configurations The VCSP which assigns a value to each state comprises just two constraints, a unary constraint h which applies only to the last symbol, and a binary constraint f. The constraint f applies to each pair of neighbouring symbols. By assigning a value of 0 to each inadmissible pair of symbols it is simple to prevent an improving flip generating an intermediate symbol in an inadmissible configuration.

Indeed, for each intermediate symbol, we can list the non-zero rows of f that include that symbol:

i_{01}: None	i_{C0}: $f(C, i_{C0}), f(X, i_{C0})$	i_{0X}: $f(i_{0X}, C)$
i_{1C}: $f(i_{1C}, C)$	i_{X1}: $f(i_{X1}, 0)$	i_{CX}: None

Let us call the sequence (from 0^N to 01^{N-1}) via steepest ascent, a counting path. Each of these non-zero-valued rows lies between a predecessor and successor on a counting path:

$$CC \Rightarrow Ci_{C0} \Rightarrow C0 \qquad XC \Rightarrow Xi_{C0} \Rightarrow X0$$
$$0C \Rightarrow i_{0X}C \Rightarrow XC \qquad 1C \Rightarrow i_{1C}C \Rightarrow CC$$
$$X0 \Rightarrow i_{X1}0 \Rightarrow 10$$

The values of f for each row will be set so that both forward flips yield an increase in the valuation of the state. Consequently a backward flip, from the resulting configurations C0,X0,XC,CC,10, will always result in a lower valuation, and therefore never occur.

Note, in particular that the pair of main symbols CC, XC, 0C, 1C, X0 which yield these configurations by a flip can only occur once in an admissible configuration. Thus through these flips, two intermediate symbols will never appear in a configuration on a counting path.

To enable the symbols i_{01} and i_{1C} to occur as the last symbol, the soft constraint h, gives a non-zero value to $h(i_{01})$ and to $h(i_{1C})$. The values of h are small enough that on a counting path i_{01} and i_{1C} will never occur in a configuration that has any other intermediate symbols. The forward flips introduced above are always more valuable than h.

Ensuring the counting path only yields admissible configurations We show in this section that it is possible to make this counting path a steepest ascent by a judicious choice of soft constraints. We first identify a set of prioritised transition rules which if followed from the initial tuple 0^N produce exactly the counting path. We then construct soft constraints which define a fitness landscape on which the transition rules correspond to an ascending path. The list of transition rules is given below.

1. Change the last symbol from 0 to 1 (incrementing the counting number)
   (a) If $X_2 \in \{0, 1\}$ and $X_1 = 0$ then set $X_1 = i_{01}$.
   (b) If $X_2 \in \{0, 1\}$ and $X_1 = i_{01}$, then set $X_1 = 1$.
2. Change the last symbol from 1 to C
   (a) If $X_2 \in \{0, 1\}$ and $X_1 = 1$ then set $X_1 = i_{1C}$.
   (b) If $X_2 \in \{0, 1\}$ and $X_1 = i_{1C}$, then set $X_1 = C$.
3. Carry to 1
   (a) If $X_{i+1} = 1$ and $X_i = C$ then set $X_{i+1} = i_{1C}$.
   (b) If $X_{i+1} = i_{1C}$ and $X_i = C$ then set $X_{i+1} = C$.
4. Carry to 0
   (a) If $X_{i+1} = 0$ and $X_i = C$ then set $X_{i+1} = i_{0X}$.
   (b) If $X_{i+1} = i_{0X}$ and $X_i = C$ then set $X_{i+1} = X$.
5. Drop the carry after C
   (a) If $X_{i+1} = C$ and $X_i = C$ then set $X_i = i_{C0}$.
   (b) If $X_{i+1} = C$ and $X_i = i_{C0}$ then set $X_i = 0$.
6. Drop the carry after X
   (a) If $X_{i+1} = X$ and $X_i = C$ then set $X_i = i_{C0}$.
   (b) If $X_{i+1} = X$ and $X_i = i_{C0}$ then set $X_i = 0$.
7. Use the final carry X
   (a) If $X_{i+1} = X$ and $X_i = 0$ then set $X_{i+1} = i_{X1}$.
   (b) If $X_{i+1} = i_{X1}$ and $X_i = 0$ then set $X_{i+1} = 1$.

These rules are applicable in the following main symbol configurations:

$\{01\}^*0$	Rule1a
$\{01\}^*1$	Rule2a
$\{01\}^*1C\{01\}^*0$	Rule1a, Rule3a
$\{01\}^*0C\{01\}^*0$	Rule1a, Rule4a
$\{01\}^*CC\{01\}^*0$	Rule1a, Rule3a, Rule4a, Rule5a
$0^*XC\{01\}^*0$	Rule1a, Rule6a
$0^*X\{01\}^*0$	Rule1a, Rule7a

To maintain the counting property, Rule 1a must only fire in the first configuration. To keep to admissible configurations, Rule 5a must fire in the fifth configuration.

Thus the priorities we impose between these rules are as follows: (1) We prefer Rule 5a, over both Rules 3a, 4a; and (2) We prefer any of Rules 3a, 4a, 5a, 6a, 7a, over Rule 1a. The VCSP will ensure that other flips do not improve on the current valuation.

The rules applicable in the states which have an intermediate value are:

$\{01\}^+ i_{01}$	Rule1b
$\{01\}^+ i_{1C}$	Rule2b
$\{01\}^+ i_{1C} C \{01\}^*$	Rule3b
$0^* i_{0X} C$	Rule4b
$0^* i_{0X} C \{01\}^* 0$	Rule1a, Rule4b
$\{01\}^+ C i_{C0}$	Rule3a, Rule4a, Rule5b
$\{01\}^+ C i_{C0} \{01\}^* 0$	Rule1a, Rule3a, Rule4a, Rule5b
$0^* X i_{C0} \{01\}^*$	Rule6b

To ensure only admissible configurations Rule 4b must fire in the fifth configuration, and Rule 5b must fire in the sixth and seventh configurations. By applying the above rules according to these priorities we ensure that only admissible states appear on the steepest ascent from 0^N. We term this the Counting Path Property (CPP). It is straightforward to verify the following CPP lemma:

Lemma 5. *All flips when applied to an admissible state yield an admissible state by applying the above prioritised transition rules.*

The following table introduces the binary soft constraint f(a,b) which we use to construct a fitness landscape in which the counting path is a steepest ascent. We obtained these values by solving a system of linear equations (described in the appendix) which ensure that a steepest-ascent respects the transition rules and their relative priorities. Many solutions exist, but to simplify the proof, we chose a solution with integer values and with a large number of zeros.

f(a, b)		b									
		0	1	C	X	i_{01}	i_{C0}	i_{0X}	i_{1C}	i_{X1}	i_{CX}
a	0	0	4	6	0	0	0	0	0	0	0
	1	0	4	6	0	0	0	0	0	0	0
	C	13	0	0	0	0	8	0	0	0	0
	X	13	0	8	0	0	12	0	0	0	0
	i_{01}	0	0	0	0	0	0	0	0	0	0
	i_{C0}	0	0	0	0	0	0	0	0	0	0
	i_{0X}	0	0	7	0	0	0	0	0	0	0
	i_{1C}	0	0	23	0	0	0	0	0	0	0
	i_{X1}	14	0	0	0	0	0	0	0	0	0
	i_{CX}	0	0	0	0	0	0	0	0	0	0

We apply f to each successive pair X_{i+1}, X_i of variables, with increasing weight 4^{i-1}. We also apply the cost function h, whose sole purpose is to trigger rules 1a and 2a, with weight 1 to X_2, X_1. The function h is identically zero except for the values $h(i_{01}) = 1$ and $h(i_{1C}) = 5$. The value of a state is thus:

$$F(X) = h(X_1) + \sum_{i=1}^{N} 4^{i-1} f(X_{i+1}, X_i) \qquad (13)$$

It remains to be shown that a steepest-ascent algorithm on the fitness landscape defined by F, starting at the initial state 0^N respects the transition rules and their priorities. By Lemma 5, since 0^N clearly satisfies CPP, we only need to consider states satisfying CPP. It suffices to show that: (a) each transition triggered by the rules leads to an increase in F, (b) when conflicts arise the priorities between rules are respected by steepest ascent, and (c) no other transition leads to an increase in F.

Lemma 6. *In the fitness landscape defined by the global objective function* F*, the counting path is a steepest-ascent.*

A series of lemmas in appendix A contains a proof of lemma 6, and we have also verified the steepest ascent by coding in Python and Eclipse [2]. We can now state our main theorem:

Theorem 7. *Steepest ascent may take an exponential number of steps to reach a local optimum even when the fitness landscape is defined by soft constraints over Boolean variables with constraint graph of pathwidth and treewidth 7.*

Proof. Since the counting path is of exponential length, Lemma 6 tells us that steepest-ascent on the landscape defined by F requires an exponential number of steps to reach a local optimum starting from the initial state 0^N. The constraint graph of the X_i variables is clearly a chain. When we replace each X_i variable by the Boolean variables $x_{1,i}, x_{2,i}, x_{3,i}, x_{4,i}$, the constraint graph has treewidth 7. To see this, observe that when the variables are ordered in lexicographic order ($x_{a,i} < x_{b,j}$ if $i < j$ or ($i = j$ and $a < b$)), the only earlier variables that constraint a variable are among the 7 variables that immediately precede it in this order. If we consider the preceeding constraining variables along with the variable they constraint as an interval then we can see that our constraint graph is an interval graph with maximum clique size 8 – so it has pathwidth 7 and thus treewidth 7. □

4 Conclusion and Discussion

We have shown that steepest-ascent local search can require an exponential number of steps to reach a local maximum even when domains are boolean and the constraint graph has treewidth (in fact, the more restrictive pathwidth) bounded by a constant k. It is an open question to determine the smallest value of k for which this holds. We have given a proof for $k = 7$. Kaznatcheev, Cohen, and Jeavons [5] have shown that for $k = 1$ all ascending paths are bounded by a quadratic number of steps and that for $k = 2$ there exist some ascending paths that are exponentially long. But their long ascending paths are not the paths that steepest ascent would follow, so for values of k between 2 and 6 the question is still open for steepest ascent.

The fact that local search may require exponential time to converge has implications both in the performance analysis of local search algorithms and in the understanding of biological processes such as evolution. Of course, we are aware that the example we have constructed is pathological.

For example, experiments on computational protein design benchmark problems indicate that the average number of steps required by steepest ascent to reach a local optimum is proportional to the logarithm of the size of the attraction basin of the local optimum [10], and hence (sub)linear in the total number of variables. Further theoretical research is required to identify conditions under which exponentially-long steepest-ascent paths cannot occur.

Acknowledgements

We thank Chris Watkins for spotting a discrepancy between the table for `f(a, b)` and verification code in an earlier draft. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 714532). The paper reflects only the authors' views and not the views of the ERC or the European Commission. The European Union is not liable for any use that may be made of the information contained therein. David Cohen was supported by Leverhulme Trust Grant RPG-2018-161.

References

[1] Umberto Bertelé and Francesco Brioschi. *Nonserial dynamic programming*. Academic Press, 1972.

[2] David A. Cohen and Mark Wallace. Code for demonstrating correctness of longest descent (eclipse, python), November 2019. URL `https://github.com/cyclingProfessor/SteepestDescentPaper`.

[3] Jeffrey Horn, David E Goldberg, and Kalyanmoy Deb. Long path problems. In *International Conference on Parallel Problem Solving from Nature*, pages 149–158. Springer, 1994.

[4] Artem Kaznatcheev. Computational complexity as an ultimate constraint on evolution. *Genetics*, 212(1): 245–265, 2019. ISSN 0016-6731. doi: 10.1534/genetics.119.302000.

[5] Artem Kaznatcheev, David A Cohen, and Peter G Jeavons. Representing fitness landscapes by valued constraints to understand the complexity of local search. In *International Conference on Principles and Practice of Constraint Programming*, pages 300–316. Springer, 2019.

[6] Daniel A Levinthal. Adaptation on rugged landscapes. *Management science*, 43(7):934–950, 1997.

[7] Pedro Meseguer, Francesca Rossi, and Thomas Schiex. *Soft Constraints*, pages 281–328. Elsevier, 2006.

[8] Jan W Rivkin and Nicolaj Siggelkow. Patterned interactions in complex systems: Implications for exploration. *Management Science*, 53(7):1068–1085, 2007.

[9] T Roughgarden. Computing equilibria: A computational complexity perspective. *Economic Theory*, 42: 193–236, 2010.

[10] David Simoncini, Sophie Barbe, Thomas Schiex, and Sébastien Verel. Fitness landscape analysis around the optimum in computational protein design. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 355–362. ACM, 2018.

[11] Tim Taylor, Mark Bedau, Alastair Channon, David Ackley, Wolfgang Banzhaf, Guillaume Beslon, Emily Dolson, Tom Froese, Simon Hickinbotham, Takashi Ikegami, et al. Open-ended evolution: Perspectives from the oee workshop in york. *Artificial life*, 22(3): 408–423, 2016.

[12] S. Wright. The roles of mutation, inbreeding, crossbreeding, and selection in evolution. In *Proceedings of the Sixth International Congress on Genetics*, pages 355–366, 1932.

A Proofs of Lemmas

Lemma 8. *If the CPP is satisfied, then all transition rules lead to an increase in the objective function* $\mathtt{F}$.

Proof. In the following let $\mathtt{S}, \mathtt{S}'$ be any (possibly empty) string and let $\mathtt{a}$ be any symbol from $\{\mathtt{0}, \mathtt{1}\}$. To show that applying Rules 1a, 2a, 1b or 2b leads to an increase in $\mathtt{F}$, we only need to show that

$$F(S\ a\ 0) < F(S\ a\ i_{01}) < F(S\ a\ 1) < F(S\ a\ i_{1C}) < F(S\ a\ C)$$

This holds since

$$\begin{aligned}\mathtt{f(a,0)+h(0)=0} \;&<\; \mathtt{f(a,i_{01})+h(i_{01})=1} \;<\; \mathtt{f(a,1)+h(1)=4}\\ &<\; \mathtt{f(a,i_{1C})+h(i_{1C})=5} \;<\; \mathtt{f(a,C)+h(C)=6}\end{aligned}$$

For Rules 3a and 3b to increase $\mathtt{F}$, we require

$$F(S\ a\ 1\ C\ S') \;<\; F(S\ a\ i_{1C}\ C\ S') \;<\; F(S\ a\ C\ C\ S')$$

(where $\mathtt{a} \in \{\mathtt{0}, \mathtt{1}\}$ follows from the CPP). These inequalities hold since

$$\mathtt{4f(a,1)+f(1,C)=22} \;<\; \mathtt{4f(a,i_{1C})+f(i_{1C},C)=23} \;<\; \mathtt{4f(a,C)+f(C,C)=24}$$

For Rules 4a and 4b to increase $\mathtt{F}$, we require

$$F(S\ a\ 0\ C\ S') \;<\; F(S\ a\ i_{0X}\ C\ S') \;<\; F(S\ a\ X\ C\ S')$$

(where $\mathtt{a} \in \{\mathtt{0}, \mathtt{1}\}$ follows from the CPP). These inequalities hold since

$$\mathtt{4f(a,0)+f(0,C)=6} \;<\; \mathtt{4f(a,i_{0X})+f(i_{0X},C)=7} \;<\; \mathtt{4f(a,X)+f(X,C)=8}$$

For Rules 6a and 6b to increase $\mathtt{F}$, we require

$$F(S\ X\ C\ S') \;<\; F(S\ X\ i_{C0}\ S') \;<\; F(S\ X\ 0\ S')$$

where, by the CPP, $\mathtt{S}'$ is a string of zeros. This holds since

$$\mathtt{4f(X,C)+f(C,0)=45} \;<\; \mathtt{4f(X,i_{C0})+f(i_{C0},0)=48} \;<\; \mathtt{52=4f(X,0)+f(0,0)}$$

when $\mathtt{S}'$ is non-empty (and $\mathtt{f(X,C)=8} < \mathtt{f(X,i_{C0})=12} < \mathtt{f(X,0)=13}$ if $\mathtt{S}'$ is the empty string). For Rules 5a and 5b to increase $\mathtt{F}$, we require

$$F(S\ C\ C\ S') \;<\; F(S\ C\ i_{C0}\ S') \;<\; F(S\ C\ 0\ S')$$

where, again by the CPP, $\mathtt{S}'$ is a string of zeros. This holds since

$$\mathtt{4f(C,C)+f(C,0)=13} \;<\; \mathtt{4f(C,i_{C0})+f(i_{C0},0)=32} \;<\; \mathtt{52=4f(C,0)+f(0,0)}$$

when $\mathtt{S}'$ is non-empty (and $\mathtt{f(C,C)=0} < \mathtt{f(C,i_{C0})=8} < \mathtt{f(C,0)=13}$ if $\mathtt{S}'$ is the empty string). For Rules 7a and 7b to increase $\mathtt{F}$, we require

$$F(S\ a\ X\ 0\ S') \;<\; F(S\ a\ i_{X1}\ 0\ S') \;<\; F(S\ a\ 1\ 0\ S')$$

(where $\mathtt{a} \in \{\mathtt{0}, \mathtt{1}\}$ and $\mathtt{S}'$ is a string of zeros by the CPP). These inequalities hold since

$$\mathtt{4f(a,X)+f(X,0)=13} \;<\; \mathtt{4f(a,i_{X1})+f(i_{X1},0)=14} \;<\; \mathtt{4f(a,1)+f(1,0)} \leq \mathtt{16}$$

□

Lemma 9. *Assuming the CPP, when one of Rules 3a or 4a are in conflict with one of Rules 5a or 5b, the transitions triggered by the latter rules lead to a greater increase in* $\mathtt{F}$.

Proof. Given the limited form of states described by the CPP, there is only one type of conflict for each pair of rules. In the following calcualtions, $\mathtt{S}, \mathtt{S}'$ represent (possibly empty) strings and (since we are requiring the CPP) the symbol $\mathtt{a}$ will always stand for either 0 or 1.

Rules 3a and 5a are only in conflict in the state $\mathtt{Sa1CCS}'$. For the priority of Rule 5a to be respected, it suffices to show that

$$F(S\ a\ i_{1C}\ C\ C\ S') \ <\ F(S\ a\ 1\ C\ i_{C0}\ S')$$

where $\mathtt{S}'$ is a string of zeros. This holds since

$$\begin{aligned}&\mathtt{64f(a,i_{1C})+16f(i_{1C},C)+4f(C,C)+f(C,0)=381}\\ &<\ \mathtt{384=64f(a,1)+16f(1,C)+4f(C,i_{C0})+f(i_{C0},0)}\end{aligned}$$

when $\mathtt{S}'$ is non-empty (and $\mathtt{16f(a,i_{1C})+4f(i_{1C},C)+f(C,C)=92<96=16f(a,1)+4f(1,C)+f(C,i_{C0})}$ if $\mathtt{S}'$ is the empty string).

Rules 3a and 5b are only in conflict in the state $\mathtt{Sa1Ci_{C0}S}'$. For the priority of Rule 5b to be respected, it suffices to show that

$$F(S\ a\ i_{1C}\ C\ i_{C0}\ S') \ <\ F(S\ a\ 1\ C\ 0\ S')$$

where, by the CPP, $\mathtt{S}'$ is a string of zeros. This holds since

$$\begin{aligned}&\mathtt{64f(a,i_{1C})+16f(i_{1C},C)+4f(C,i_{C0})+f(i_{C0},0)=400}\\ &<\ \mathtt{404=64f(a,1)+16f(1,C)+4f(C,0)+f(0,0)}\end{aligned}$$

when $\mathtt{S}'$ is non-empty (and $\mathtt{16f(a,i_{1C})+4f(i_{1C},C)+f(C,i_{C0})=100<101=16f(a,1)+4f(1,C)+f(C,0)}$ if $\mathtt{S}'$ is the empty string).

Rules 4a and 5a are only in conflict in the state $\mathtt{Sa0CCS}'$. For the priority of Rule 5a to be respected, it suffices to show that

$$F(S\ a\ i_{0X}\ C\ C\ S') \ <\ F(S\ a\ 0\ C\ i_{C0}\ S')$$

where, by the CPP, $\mathtt{S}'$ is a string of zeros. This holds since

$$\begin{aligned}&\mathtt{64f(a,i_{0X})+16f(i_{0X},C)+4f(C,C)+f(C,0)=125}\\ &<\ \mathtt{128=64f(a,0)+16f(0,C)+4f(C,i_{C0})+f(i_{C0},0)}\end{aligned}$$

when $\mathtt{S}'$ is non-empty (and $\mathtt{16f(a,i_{0X})+4f(i_{0X},C)+f(C,C)=28<32=16f(a,0)+4f(0,C)+f(C,i_{C0})}$ if $\mathtt{S}'$ is the empty string).

Rules 4a and 5b are only in conflict in the state $\mathtt{Sa0Ci_{C0}S}'$. For the priority of Rule 5b to be respected, it suffices to show that

$$F(S\ a\ i_{0X}\ C\ i_{C0}\ S') \ <\ F(S\ a\ 0\ C\ 0\ S')$$

where, by the CPP, $\mathtt{S}'$ is a string of zeros. This holds since

$$\begin{aligned} &64\mathtt{f}(\mathtt{a}, \mathtt{i_{0X}}) + 16\mathtt{f}(\mathtt{i_{0X}}, \mathtt{C}) + 4\mathtt{f}(\mathtt{C}, \mathtt{i_{C0}}) + \mathtt{f}(\mathtt{i_{C0}}, 0) = 144 \\ &< \ 148 = 64\mathtt{f}(\mathtt{a}, 0) + 16\mathtt{f}(0, \mathtt{C}) + 4\mathtt{f}(\mathtt{C}, 0) + \mathtt{f}(0, 0) \end{aligned}$$

when $\mathtt{S}'$ is non-empty (and $16\mathtt{f}(\mathtt{a}, \mathtt{i_{0X}}) + 4\mathtt{f}(\mathtt{i_{0X}}, \mathtt{C}) + \mathtt{f}(\mathtt{C}, \mathtt{i_{C0}}) = 36 < 37 = 16\mathtt{f}(\mathtt{a}, 0) + 4\mathtt{f}(0, \mathtt{C}) + \mathtt{f}(\mathtt{C}, 0)$ if $\mathtt{S}'$ is the empty string). □

Lemma 10. *Assuming the CPP, when Rule 1a is in conflict with one of the other rules, the transition triggered by the other rule always leads to a greater increase in* F *then Rule 1a.*

Proof. By the CPP, conflicts between Rule 1a and another rule can only occur in a state ending with a string of at least two zeros, namely one of $\mathtt{S0C0}^{\mathtt{r}}$, $\mathtt{S1C0}^{\mathtt{r}}$, $\mathtt{SX0}^{\mathtt{r}}$, $\mathtt{Si_{X1}0}^{\mathtt{r}}$, $\mathtt{S0CC0}^{\mathtt{r}}$, $\mathtt{S1CC0}^{\mathtt{r}}$, $\mathtt{SXC0}^{\mathtt{r}}$, $\mathtt{S0Ci_{C0}0}^{\mathtt{r}}$, $\mathtt{S1Ci_{C0}0}^{\mathtt{r}}$, $\mathtt{Si_{1C}C0}^{\mathtt{r}}$, $\mathtt{Si_{0X}C0}^{\mathtt{r}}$, $\mathtt{SXi_{C0}0}^{\mathtt{r}}$ for some string S of zeros and ones, and with $\mathtt{r} \geq 2$. In each case, the application of Rule 1a leads to an increase of just 1 in the value of F, whereas the application of any other rule leads to an increase of at least $4^{\mathtt{r}-1} > 1$ (by the same exhaustive case analysis as given in the proof of Lemma 8). □

Lemma 11. *Assuming the CPP is satisfied, the only transitions which lead to an increase in* F *are those triggered by Rules 1-14.*

Proof. Recall that the only possible transitions are those between a main symbol (0, 1, C or X) and an intermediate symbol, since all other transitions change 2 bits.

We first consider all possible transitions from a main symbol to one of the six intermediate symbols:

$\mathtt{i_{C0}}$: since the only non-zero values of f (or h) with an argument $\mathtt{i_{C0}}$ are $\mathtt{f}(\mathtt{X}, \mathtt{i_{C0}}) = 12$ and $\mathtt{f}(\mathtt{C}, \mathtt{i_{C0}}) = 8$, the only transitions we need to consider are $\mathtt{SX0S}' \to \mathtt{SXi_{C0}S}'$, $\mathtt{SC0S}' \to \mathtt{SCi_{C0}S}'$ and $\mathtt{SXCS}' \to \mathtt{SXi_{C0}S}'$, $\mathtt{SCCS}' \to \mathtt{SCi_{C0}S}'$. The first two transitions lead to a decrease in F since they correspond to the inverse transitions of Rules 6b and 5b. The latter two transition correspond to Rules 6a and 5a.

$\mathtt{i_{0X}}$: since the only non-zero value of f (or h) with an argument $\mathtt{i_{0X}}$ is $\mathtt{f}(\mathtt{i_{0X}}, \mathtt{C}) = 7$, the only transitions we need to consider are $\mathtt{S0CS}' \to \mathtt{Si_{0X}XS}'$ and $\mathtt{SXCS}' \to \mathtt{Si_{0X}XS}'$. The former corresponds to Rule 4a, whereas the latter leads to a decrease in F since it is the inverse transition of Rule 4b.

$\mathtt{i_{1C}}$: since the only non-zero values of f or h with an argument $\mathtt{i_{1C}}$ are $\mathtt{f}(\mathtt{i_{1C}}, \mathtt{C}) = 23$ and $\mathtt{h}(\mathtt{i_{1C}}) = 5$, the only transitions we need to consider are $\mathtt{S1CS}' \to \mathtt{Si_{1C}CS}'$, $\mathtt{SCCS}' \to \mathtt{Si_{1C}CS}'$, and $\mathtt{Sa1} \to \mathtt{Sai_{1C}}$ $\mathtt{SaC} \to \mathtt{Sai_{1C}}$ (where $\mathtt{a} \in \{0, 1\}$). Of these transitions, the first corresponds to Rule 3a, the second to the inverse of Rule 3b, the third to Rule 2a and the last to the inverse of Rule 2b.

$\mathtt{i_{X1}}$: since the only non-zero value of f (or h) with an argument $\mathtt{i_{X1}}$ is $\mathtt{f}(\mathtt{i_{X1}}, 0) = 14$, the only transitions we need to consider are $\mathtt{S10S}' \to \mathtt{Si_{X1}0S}'$ and $\mathtt{SX0S}' \to \mathtt{Si_{X1}0S}'$. The former corresponds to the inverse of Rule 7b and the latter corresponds to Rule 7a.

$\mathtt{i_{01}}$: since the only non-zero values of ($\mathtt{f}$ or) $\mathtt{h}$ with an argument $\mathtt{i_{01}}$ are $\mathtt{h(i_{01})} = \mathtt{1}$, the only transitions we need to consider are $\mathtt{Sa0} \to \mathtt{Sai_{01}}$ and $\mathtt{Sa1} \to \mathtt{Sai_{01}}$ (where $\mathtt{a} \in \{\mathtt{0}, \mathtt{1}\}$). The former corresponds to Rule 1a and the latter corresponds to the inverse of Rule 1b.

We now consider all possible transitions from one of the six intermediate symbols to a main symbol:

$\mathtt{i_{C0}}$: since, by the CPP, $\mathtt{i_{C0}}$ can only occur just after $\mathtt{X}$ or $\mathtt{C}$, the only transitions we need to consider are $\mathtt{SXi_{C0}S'} \to \mathtt{SX0S'}$, $\mathtt{SCi_{C0}S'} \to \mathtt{SC0S'}$, $\mathtt{SXi_{C0}S'} \to \mathtt{SXCS'}$, and $\mathtt{SCi_{C0}S'} \to \mathtt{SCCS'}$. Of these transitions, the first corresponds to Rule 6b, the second to Rule 5b, the third to the inverse of Rule 6a and the last to the inverse of Rule 5a.

$\mathtt{i_{0X}}$: since, by the CPP, $\mathtt{i_{0X}}$ can only occur just before $\mathtt{C}$, the only transitions we need to consider are $\mathtt{Si_{0X}CS'} \to \mathtt{S0CS'}$ and $\mathtt{Si_{0X}CS'} \to \mathtt{SXCS'}$. The first of these transitions corresponds to Rule 4a and the second corresponds to the inverse of Rule 4b.

$\mathtt{i_{1C}}$: since, by the CPP, $\mathtt{i_{1C}}$ can only occur at $\mathtt{X_1}$ (with $\mathtt{X_2} \in \{\mathtt{0}, \mathtt{1}\}$) or just before $\mathtt{C}$, the only transitions we need to consider are $\mathtt{Sai_{1C}} \to \mathtt{Sa1}$, $\mathtt{Sai_{1C}} \to \mathtt{SaC}$ (where $\mathtt{a} \in \{\mathtt{0}, \mathtt{1}\}$), and $\mathtt{Si_{1C}CS'} \to \mathtt{S1CS'}$, $\mathtt{Si_{1C}CS'} \to \mathtt{SCCS'}$. Of these transitions, the first is the inverse of Rule 1a, the second is Rule 2b, the third is the inverse of Rule 3a and the fourth is Rule 3b.

$\mathtt{i_{X1}}$: since, by the CPP, $\mathtt{i_{X1}}$ can only occur just before $\mathtt{0}$, the only transitions we need to consider are $\mathtt{Si_{X1}0S'} \to \mathtt{S10S'}$ and $\mathtt{Si_{X1}0S'} \to \mathtt{SX0S'}$. The first of these transitions corresponds to Rule 7b and the second to the inverse of Rule 7a.

$\mathtt{i_{01}}$: since, by the CPP, $\mathtt{i_{01}}$ can only occur at $\mathtt{X_1}$ (with $\mathtt{X_2} \in \{\mathtt{0}, \mathtt{1}\}$), the only transitions we need to consider are $\mathtt{Sai_{01}} \to \mathtt{Sa0}$ and $\mathtt{Sai_{01}} \to \mathtt{Sa1}$. The former corresponds to the inverse of Rule 1a and the latter to Rule 1b.

□

Lemma 6 now follows directly from Lemmas 8, 9, 10, 11, and the steepest ascent has been verified by coding in Python and Eclipse [2].