Efficient Hill Climber for Constrained Pseudo-Boolean Optimization Problems

Francisco Chicano
University of Málaga
Andalucía Tech, Spain
chicano@lcc.uma.es

Darrell Whitley
Colorado State University
Fort Collins, CO, USA
whitley@cs.colostate.edu

Renato Tinós
University of São Paulo
Riberão Preto, SP, Brazil
rtinos@uspl.br

ABSTRACT

Efficient hill climbers have been recently proposed for single- and multi-objective pseudo-Boolean optimization problems. For k-bounded pseudo-Boolean functions where each variable appears in at most a constant number of subfunctions, it has been theoretically proven that the neighborhood of a solution can be explored in constant time. These hill climbers, combined with a high-level exploration strategy, have shown to improve state of the art methods in experimental studies and open the door to the so-called Gray Box Optimization, where part, but not all, of the details of the objective functions are used to better explore the search space. One important limitation of all the previous proposals is that they can only be applied to unconstrained pseudo-Boolean optimization problems. In this work, we address the constrained case for multi-objective k-bounded pseudo-Boolean optimization problems. We find that adding constraints to the pseudo-Boolean problem has a linear computational cost in the hill climber.

Keywords

Hamming Ball Hill Climber; Local Search; Constraint Handling; Vector Mk Landscapes; Multi-Objective Optimization

1. INTRODUCTION

Hill climbing is one base technique used as a component of higher-level search algorithms like Iterated Local Search or Variable Neighborhood Search [7]. A hill climber starts at an initial solution and then searches for an improving move based on a notion of a neighborhood of solutions that are adjacent to the current solution.

We focus in this work on *Mk landscapes*. Mk landscapes, introduced by Whitley [12], are k-bounded pseudo-Boolean optimization problems composed of a linear combination of M subfunctions, where each subfunction is a pseudo-Boolean optimization problem defined over k variables. This definition is general enough to include NK landscapes, MAX-kSAT, as well as spin glass problems. For Mk landscapes, Whitley and Chen [13] proved that the location of improving moves can be determined in constant time for the Hamming distance 1 neighborhood. Two solutions are neighboring in this neighborhood if they differ in one bit. This result was later generalized by Chicano et al. [3], who proposed a hill climber that explores the solutions contained in a Hamming ball of radius r around a solution in constant time. Goldman et al. [6] combined this so-called *Hamming Ball Hill Climber* with recombination to achieve globally optimal results on relatively large Adjacent NK Landscape problems (e.g. 10,000 variables). The work of Goldman et al. shows that evolutionary algorithms do not need mutation to find improving moves, although mutation could probably be used to enable a form of restart. In a recent paper [4], the Hamming Ball Hill Climber proposed in [3] was extended to deal with multi-objective problems, where the objective function is given by the so-called *vector Mk landscape*s.

One aspect that is missing in all the previous work related to efficient hill climbers is the constraint handling. None of the previous hill climbers can be applied to constrained optimization problems. However, many optimization problems have constraints. One example that will be revisited later is the knapsack problem [8], whose objective function is an Mk landscape, since it is a linear pseudo-Boolean function in the form $\sum_{i=1}^{n} v_i x_i$, and its constraint is also based on an Mk landscape (in the form $\sum_{i=1}^{n} w_i x_i \leq W$). It can be argued that constraints can be handled using penalties in the objective function. In the case of (vector) Mk landscapes, however, the non-linearities introduced by the penalties could increase the epistasis degree and the number of variables of the problem, making hill climbers less efficient.

In this work, we provide some theoretical results that limit the performance of efficient hill climbers over constrained problems. In particular, it is found that the constant time per move of the previous hill climbers cannot be achieved when constraints are added to the problem. Instead, a linear time per move is possible. We use these theoretical results to propose an efficient hill climber for multi-objective constrained pseudo-Boolean problems.

The rest of the paper is organized as follows. In the next section we introduce some background and review the previous results. Section 3 provides the new theoretical results regarding hill climbing in constrained pseudo-Boolean problems and Section 4 proposes a hill climber based on these results. We also analyze the hill climber using some experiments whose results are presented in Section 5. Finally, Section 6 outlines some conclusions and future work.

To appear in Genetic and Evolutionary Computation Conference 2016

2. BACKGROUND

In constrained multi-objective optimization, there is a vector function $\mathbf{f} : \mathbb{B}^n \to \mathbb{R}^d$ to optimize, called the *objective function*. We will assume, without loss of generality, that all the objectives (components of the vector function) are to be maximized. The constraints of the problem will be given in the form[1] $\mathbf{g}(x) \geq 0$, where $\mathbf{g} : \mathbb{B}^n \to \mathbb{R}^b$ is a vector function, that will be called *constraint function*[2]. That is, a solution is feasible if all the components of the vector function $\mathbf{g}$ are nonnegative when evaluated in that solution. This type of constraints does not represent a limitation. Any other equality or inequality constraint can be expressed in the form $g_i(x) \geq 0$, including those that use strict inequality constraints ($>$ and $<$)[3]. The set of feasible solutions of a problem will be denoted by $X_{\mathbf{g}} = \{x \in \mathbb{B}^n | \mathbf{g}(x) \geq 0\}$.

Given a vector function $\mathbf{f} : \mathbb{B}^n \to \mathbb{R}^d$, we say that solution $x \in \mathbb{B}^n$ *dominates* solution $y \in \mathbb{B}^n$, denoted with $x \succ_{\mathbf{f}} y$, if and only if $\mathbf{f}(x) \geq \mathbf{f}(y)$ and there exists $j \in \{1, 2, \ldots, d\}$ such that $f_j(x) > f_j(y)$. When the vector function is clear from the context, we will use $\succ$ instead of $\succ_{\mathbf{f}}$. Observe that we do not restrict the definition of dominance to feasible solutions of a problem, since we will need to consider dominance also between feasible and non-feasible solutions. Furthermore, we will use the concept of dominance using as vector function the constraint function.

The *Pareto Optimal Set* is the set of feasible solutions P that are not dominated by any other feasible solution in $X_{\mathbf{g}}$:

$$P = \{x \in X_{\mathbf{g}} | \nexists y \in X_{\mathbf{g}}, y \succ x\}\,. \tag{1}$$

The *Pareto Front* is the image by $\mathbf{f}$ of the Pareto Optimal Set: $PF = \mathbf{f}(P)$.

Definition 1 (Local Optimum [10]). *Given a vector function $\mathbf{f} : \mathbb{B}^n \to \mathbb{R}^d$, a constraint function $\mathbf{g} : \mathbb{B}^n \to \mathbb{R}^b$, and a neighborhood function $N : \mathbb{B}^n \to 2^{\mathbb{B}^n}$, we say that solution $x \in X_{\mathbf{g}}$ is a* local optimum *if it is not dominated by any other feasible solution in its neighborhood: $\nexists y \in N(x) \cap X_{\mathbf{g}}, y \succ x$.*

In this work we focus our attention to problems where the objective and constraint functions have k-bounded epistasis, that is, their components are *Mk Landscapes* [12]. These vector functions have been recently called *vector Mk landscapes* and are defined as follows.

Definition 2 (Vector Mk Landscape [4]). *Given two constants k and d, a* vector Mk Landscape $\mathbf{f} : \mathbb{B}^n \to \mathbb{R}^d$ *is a d-dimensional vector pseudo-Boolean function defined over $\mathbb{B}^n$ whose components are Mk Landscapes. That is, each component f_i can be written as a sum of m_i subfunctions, each one depending at most on k input variables:*

$$f_i(x) = \sum_{l=1}^{m_i} f_i^{(l)}(x) \quad \text{for } 1 \leq i \leq d, \tag{2}$$

where the subfunctions $f_i^{(l)}$ depend on k components of x.

Figure 1(a) shows a multi-objective constraint pseudo-Boolean problem with $d = 2$ objective functions and $b = 1$ constraint function. The first objective function, f_1, can be written as the sum of 3 subfunctions, $f_1^{(1)}$ to $f_1^{(3)}$, shown with white rectangles. The second component function, f_2, can be written as the sum of 2 subfunctions, $f_2^{(1)}$ and $f_2^{(2)}$, shown with blue (grey) rectangles. Finally, the constraint g_1 can be written as the sum of 3 subfunctions, $g_1^{(1)}$ to $g_1^{(3)}$, and is shown with circles in the figure. All the subfunctions depend at most on $k = 2$ variables.

Considering only vector Mk landscapes for the objective and constraint functions is not a limitation, since every pseudo-Boolean function that can be expressed in algebraic form using polynomial space can be transformed in polynomial time into a quadratic ($k = 2$) pseudo-Boolean function [11].

An important tool for the forthcoming analysis is the *co-ocurrence graph.*

Definition 3 (Co-ocurrence Graph [5]). *Given an objective function $\mathbf{f} : \mathbb{B}^n \to \mathbb{R}^d$ and a constraint function $\mathbf{g} : \mathbb{B}^n \to \mathbb{R}^b$, the* co-occurrence graph *is $G = (V, E)$, where V is the set of Boolean variables and E contains all the pairs of variables (x_{j_1}, x_{j_2}) that* co-occur *(both variables are arguments) in a subfunction of the objective or the constraint function.*

In Figure 1(b) we show the variable co-occurrence graph of the vector Mk Landscape of Figure 1(a).

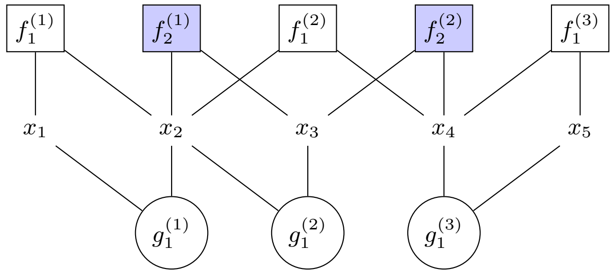
A *move* in $\mathbb{B}^n$ can be characterized by a binary string $v \in \mathbb{B}^n$ having 1 in all the bits that change in the solution. The *Score* of a move, has been previously defined as the increment in the objective function when that move is taken.

Definition 4 (Score). *For $v, x \in \mathbb{B}^n$, and a vector function $\mathbf{f} : \mathbb{B}^n \to \mathbb{R}^d$, we denote the* Score *of x with respect to* move *v for function $\mathbf{f}$ as $\mathbf{S}_v^{(\mathbf{f})}(x)$, defined as follows:*

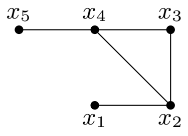
$$\mathbf{S}_v^{(\mathbf{f})}(x) = \mathbf{f}(x \oplus v) - \mathbf{f}(x), \tag{3}$$

where $\oplus$ is the exclusive OR bitwise operation.

When constraints are considered, we also need to track where feasible solutions are, and Scores of the contraint



(a) Vector Mk Landscape



(b) Co-occurrence graph

Figure 1: A multi-objective constrained pseudo-Boolean problem with $k = 2$, $n = 5$ variables, $d = 2$ objectives and $b = 1$ constraint (top) and its corresponding co-occurrence graph (bottom).

[1]The relational operators applied to vectors will mean that each component must fulfill the inequality.
[2]We will use **boldface** to denote vectors in $\mathbb{R}^d$ or $\mathbb{R}^b$, such as $\mathbf{f}$, but we will use normal weight for vectors in $\mathbb{B}^n$, like x.
[3]This is a consequence of the finite search space.

function will also be useful to efficiently track them. For this reason we decorate the notation for a Score with the function the Score is referred to (see Definition 4) and we distinguish between the *objective Score*, $\mathbf{S}^{(\mathbf{f})}$, and the *constraint Score*, $\mathbf{S}^{(\mathbf{g})}$. If a result holds for both we will omit the vector function.

2.1 Previous Results

There are some previous results for Multi-Objective pseudo-Boolean problems that can be easily adapted to the case of constrained multi-objective problems without providing a proof. We review these results in this subsection. In [4] it was found that some Scores can be written as a sum of other Scores.

Proposition 1 (from [4]). *Let $v_1, v_2 \in \mathbb{B}^n$ be two moves such that*[4] *$v_1 \cap v_2 = \emptyset$ and variables in v_1 do not co-occur with variables in v_2, that is, the subfunctions where variables in v_1 appear and the ones where variables in v_2 appear are disjoint. Then, the Score function $\mathbf{S}_{v_1 \cup v_2}(x)$ can be written as:*

$$\mathbf{S}_{v_1 \cup v_2}(x) = \mathbf{S}_{v_1}(x) + \mathbf{S}_{v_2}(x). \tag{4}$$

When $v_1, v_2 \in \mathbb{B}^n$ are two moves fulfilling the previous proposition, we will say that v_1 and v_2 *do not interact*. If such a decomposition is not possible we will say that v_1 and v_2 *interact*. For example, in the vector Mk Landscape of Figure 1 the Score $\mathbf{S}_{\underline{1,3,4}}$ can be written as the sum of the Score $\mathbf{S}_{\underline{1}}$ and $\mathbf{S}_{\underline{3,4}}$, where we used $\underline{i_1, i_2, ...}$ to denote the binary string having 1 in positions $i_1, i_2, \ldots$, and the rest set to 0. That is, $\underline{3,4}$ and $\underline{1}$ do not interact.

One important consequence of this result is that there is no need to store all the Scores in memory to have complete information of the influence that the moves in a Hamming ball of radius r have. Only the Scores defined as

$$M^r = \{v \in \mathbb{B}^n | 1 \leq |v| \leq r \text{ and } G[v] \text{ is connected}\}, \tag{5}$$

need to be stored. In the previous expression, G is the co-occurrence graph of the problem and $G[v]$ is the subgraph of G induced by the set of vertices v. If all the variables appear in at most a constant number of subfunctions c, the number of Scores to store in memory was proven to be $\Theta(n(3ck)^r)$. Although the original proof is for unconstrained multi-objective problems, it is still true if we add constraints, since the only difference with the unconstrained case is that the variables could appear in more subfunctions (the ones of the constraint function). Another consequence of Proposition 1 is the constant time required to update the Scores in one move. This time is $O(b(k)|t|(3ck)^{r+1})$, where $b(k)$ is a bound on the time required to evaluate any subfunction and $|t|$ is the number of bits flipped in the move.

The efficient hill climbers in the previous work [13, 14, 3, 4] only explore the moves in M^r to identify improving moves. In the multi-objective case it was found that it is not always possible to deduce the presence or absence of a move in the Hamming ball of radius r dominating the current solution, unless one exists in M^r. These type of moves were called *strong improving* moves. The fact that the objective function is a vector and not a scalar has to do with this. Under some conditions, however, it is possible to certify that no strong improving move exists in the ball. In particular:

Proposition 2 (from [4]). *Let $\mathbf{f} : \mathbb{B}^n \to \mathbb{R}^d$ be a vector Mk Landscape, and $\mathbf{w} \in \mathbb{R}^d$ a d-dimensional weight vector with $\mathbf{w} > 0$. If there exists a strong improving move in a ball of radius r around solution x, then there exists $v \in M^r$ such that $\mathbf{w} \cdot \mathbf{S}_v(x) > 0$.*

The moves where $\mathbf{w} \cdot \mathbf{S}_v(x) > 0$ is true are called **w**-*improving* moves. The previous result was the basis of a hill climber that first takes all the strong improving moves in M^r and, if none exists, it takes the **w**-improving moves. If no **w**-improving move exists the algorithm stops, since there is no strong improving move in the Hamming ball. Taking **w**-improving moves does not guarantee that a strong improving move in the Hamming ball of the original solution will be reached. But probably it is the best strategy a hill climber can use to reach a strong improving move (if it exists) in constant time per move.

We will see in Section 3 that something similar occurs with the feasible moves. And we will need to define the concept of **w**-*feasible* move.

3. CONSTRAINED PROBLEMS

We have seen in the previous section that when the number of subfunctions each variable appears in is bounded by a constant, all the Scores (either objective or constraint) can be updated in constant time. The strong improving and **w**-improving moves can also be identified in constant time, since the value of an objective Score is enough to determine its category.

However, the classification of the moves as feasible or unfeasible, requires more than constant time. The feasibility of a move v does not only depend on the contraint Score $\mathbf{S}_v^{(\mathbf{g})}(x)$, but also on $\mathbf{g}(x)$. A move v is feasible in a solution x when $\mathbf{g}(x) + \mathbf{S}_v^{(\mathbf{g})}(x) \geq 0$. The value of $\mathbf{g}(x)$ could change, in general, after every move. Thus, even if the constraint Score $\mathbf{S}_v^{(\mathbf{g})}(x)$ does not change for move v, the feasibility of move v could change. This means that we need to re-evaluate the feasibility of each move every time $\mathbf{g}(x)$ changes. Since the number of moves whose Scores are stored in memory is $\Theta(n)$, this is also the worst case time required to re-classify the moves as feasible or unfeasible. We could use a complex data structure to keep track of which moves need to be re-classified and which not[5]. However, the next Theorem proves that, in the worst case, the time required to re-classify moves is $\Theta(n)$.

Theorem 1. *Let $x, y \in \mathbb{B}^n$ be two neighboring solutions of a multi-objective constrained Pseudo-Boolean problem and M^r the set of moves whose Scores are stored in memory. Let us assume that $|M^r| = \Theta(n)$. The number of moves feasible in x that are unfeasible in y could be $\Theta(n)$ in the worst case.*

Proof. In order to prove this, we only need to find a constrained pseudo-Boolean problem with $|M^r| = \Theta(n)$ where this could happen. Let's use the knapsack problem [8], which consists in maximizing the sum of the value of a set of items with the constraint that the sum of the weights of the items

[4]Abuse of notation: we will interpret moves as binary strings and sets of variables to flip.

[5]For example, if $\mathbf{g}$(x) has increased in all the components, then the constraint Scores with $\mathbf{S}_v^{(\mathbf{g})}(x) \geq 0$ stay in the same class.

does not exceed a maximum capacity (of the knapsack). Let us consider only the case when $r = 1$. In this case all the n moves have their Scores stored in memory. Let us assume that the weight of the heaviest item is the maximum capacity of the knapsack. Let solution $x = 0$ represent an empty knapsack. In x all the order-1 moves are feasible, since any item can be introduced in the knapsack without exceeding the capacity. Let us say that solution y is the one where the heaviest item is introduced in the knapsack. It is clear that y is a neighbor of x and the Hamming distance between both solutions is 1. In solution y, however, all moves are unfeasible, except the one that leads to solution $x = 0$ (empty knapsack). Thus, the number of moves feasible in x and unfeasible in the neighboring solution y is $n - 1$. $\square$

The previous result implies that hill climbers considering constraints require $O(n)$ time per move, in contrast to the $O(1)$ time per move when there is no constraint. Now we address the identification of feasible moves in the ball. The next theorem proves that an analysis of the moves in M^r provides some information about the existence of feasible moves in the ball.

THEOREM 2. *Let $x \in \mathbb{B}^n$ be a feasible solution of a multi-objective constrained pseudo-Boolean problem, that is, $x \in X_{\mathbf{g}}$. If there is a feasible move in the Hamming ball of radius r around x, then, there must exist a move v in M^r for which $\mathbf{w} \cdot \mathbf{g}(x \oplus v) \geq 0$, for any $\mathbf{w} > 0$.*

PROOF. Let us say that v is a feasible move in the Hamming ball of radius r. Then, by Proposition 1, there exist moves $v_1, v_2, \ldots v_j \in M^r$ such that $\mathbf{S}_v^{(\mathbf{g})}(x) = \sum_{l=1}^{j} \mathbf{S}_{v_l}^{(\mathbf{g})}(x)$. Since v is feasible and all $\mathbf{w} > 0$, we have:

$$\mathbf{w} \cdot (\mathbf{g}(x) + \mathbf{S}_v^{(\mathbf{g})}(x)) = \mathbf{w} \cdot \mathbf{g}(x) + \sum_{l=1}^{j} \mathbf{w} \cdot \mathbf{S}_{v_l}^{(\mathbf{g})}(x) \geq 0. \quad (6)$$

Due to the feasibility of x we have $\mathbf{w} \cdot \mathbf{g}(x) \geq 0$, and adding $j - 1$ times this value to (6) we obtain:

$$\sum_{l=1}^{j} \mathbf{w} \cdot \left(\mathbf{g}(x) + \mathbf{S}_{v_l}^{(\mathbf{g})}(x)\right) \geq 0. \quad (7)$$

For the previous expression to be nonnegative, there must be a v_l with $1 \leq l \leq j$ such that $\mathbf{w} \cdot (\mathbf{g}(x) + \mathbf{S}_{v_l}^{(\mathbf{g})}(x)) \geq 0$. $\square$

A move v with $\mathbf{w} \cdot \mathbf{g}(x \oplus v) \geq 0$ will be called **w**-*feasible*. The previous result allows us to efficiently identify the case in which no feasible solution exists around the current solution. In that case, none of the stored moves will be **w**-feasible. However, if there is a **w**-feasible move in M^r that is not feasible, we cannot ensure that a feasible move exists in the Hamming ball of radius r. In that case, the only computationally efficient decision a hill climber can take is to enter the unfeasible region taking an unfeasible, but **w**-feasible, move.

Unfortunately, when the current solution is unfeasible, identifying a feasible move or prove that none exists in the Hamming ball of radius r seems to be not much more efficient than an exhaustive exploration of the Hamming ball.

CONJECTURE 1. *Given a multi-objective constrained pseudo-Boolean optimization problem, and $x \in \mathbb{B}^n$ an unfeasible solution to the problem ($x \notin X_{\mathbf{g}}$), any algorithm to identify a feasible move in the Hamming ball of radius r around x or to confirm that such a move does not exist runs in $\Omega(n^r)$ time in the worst case, which is the time required to enumerate the Hamming ball.*

In order to support this conjecture, let us first introduce the *Move Interaction Graph*.

DEFINITION 5 (MOVE INTERACTION GRAPH). *Given a multi-objective constrained pseudo-Boolean optimization problem and a set of moves M, the Move Interaction Graph is defined as $G_M = (M, E_M)$, where the vertices are the moves in M and an edge exists between v_1 and v_2 if the moves v_1 and v_2 interact with each other, that is, they share a variable, or they have variables that interact.*

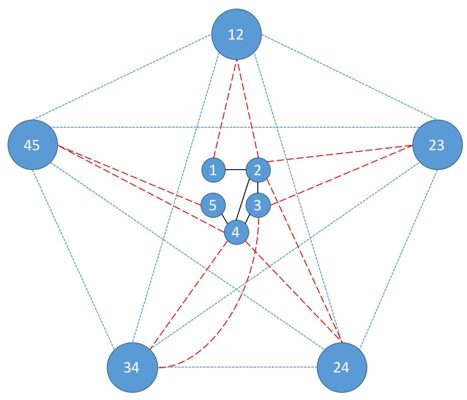


Figure 2: Move Interaction Graph of M^2 for the multi-objective constrained pseudo-Boolean problem of Figure 1.

.

Figure 2 shows the Move Interaction Graph for the set M^2 associated to the multi-objective constrained pseudo-Boolean problem of Figure 1. The numbers in the nodes are the bits flipped in each move. Observe that the five second order moves of M^2 form a complete graph (joined by dotted lines). The subgraph containing only the order-1 moves coincides with the variable co-occurence graph of Figure 1(b) (joined by solid lines in Figure 2). The rest of edges (dashed lines) correspond to interactions between order-1 and order-2 moves.

According to Proposition 1, the Score of a move can be decomposed if and only if the move can be described as the union of two moves that do not interact. We can apply the argument the other way around. Joining a set of moves that do not interact with each other, it is possible to create a higher-order move whose Score can be computed as the sum of the Scores of the component moves. Sets of moves that do not interact are independent sets[6] of a Move Interaction Graph.

By definition of M^r, the independent sets of G_{M^r} cover all the moves in the Hamming ball of radius r. In other words, any move in the Hamming ball can be expressed as an independent set of G_{M^r}. But there are also independent sets in G_{M^r} that correspond to moves outside the ball.

[6]In a graph, an *independent set* is a subset of vertices where no pair is adjacent.

Let us consider the case in which the constraint function $\mathbf{g}$ is 1-dimensional. Let us weigh each move v in G_{M^r} with its corresponding constraint Score $\mathbf{S}_v^{(\mathbf{g})}(x)$. If x is unfeasible, $\mathbf{g}(x) < 0$. A feasible move, v, in the Hamming ball must fulfill $\mathbf{g}(x) + \mathbf{S}_v^{(\mathbf{g})}(x) \geq 0$, or equivalently, $\mathbf{S}_v^{(\mathbf{g})}(x) \geq -\mathbf{g}(x)$. Observe that $-\mathbf{g}(x) > 0$. Finding such a move is equivalent to finding an independent set in G_{M^r} such that the sum of weights is greater than or equal to $-\mathbf{g}(x)$. Since an independent set of a graph is a clique in the complementary graph, the problem we are trying to solve is the decision problem related to the maximum weighted clique problem, which is NP-hard. If we assume that $\mathbf{g}$ can only take integer values and the constraint Score of all the moves are, by chance, 1, the problem is an instance of the k-clique problem, which is also NP-hard.

We know, however, that finding a feasible move can be solved in $\Theta(n^r)$ by exploring the Hamming ball. Thus, the particular instances of the maximum weighted clique problem we are solving are easier to solve than a general instance. An important question arises. Could we exploit some particular feature of the Move Interaction Graph to reduce the computation time below $\Theta(n^r)$? We have no answer to this question yet and we defer an answer to future work. But the previous argument makes us pessimistic.

4. A HILL CLIMBER FOR CONSTRAINED PROBLEMS

We have now all the theoretical ingredients to build an efficient hill climber for multi-objective constrained pseudo-Boolean problems. We assume that there is a high-level algorithm that runs the hill climber and provides the weight vector $\mathbf{w}$, which determines the direction of exploration in the objective space. We defer to future work the problem of finding an appropriate weight vector. We would like our hill climber to be able to identify feasible improving moves in the Hamming ball and stop only when no feasible improving move exists. Conjecture 1, however, prevents us from doing this efficiently. We will preserve efficiency and relax the identification requirement.

The hill climber is shown in Algorithm 1. Let us assume that the hill climber starts in a feasible solution. Then, it first explores the feasible region taking $\mathbf{w}$-improving moves while it is possible. This is the role of `exploreFeasibleRegion`, which is detailed in Algorithm 2. This procedure first takes a feasible strong improving move if one exist (Line 3) or a feasible $\mathbf{w}$-improving (and not strong improving) move otherwise (Line 5). In this case, it reports the solution to the high-level algorithm using the `report` procedure, since it could be one of the non-dominated solutions provided by the hill climber[7]. The procedure `report` should add the reported solution to an external set of non-dominated solutions. This set should be managed by the high-level algorithm invoking the hill climber.

The `exploreFeasibleRegion` procedure finishes when there is no feasible $\mathbf{w}$-improving move in M^r. We can distinguish three possible scenarios here. It could happen that there is no $\mathbf{w}$-improving move (either feasible or unfeasible) in M^r. Then, by Proposition 2 we can certify that no improving solution exists in the Hamming ball, and, in particular, no feasible improving move. In that case, the hill climber stops in Line 5 of Algorithm 1. It can also happen that there is no feasible move in M^r. Since we are in a feasible solution, we can randomly select a $\mathbf{w}^*$ vector and apply Theorem 2 to check if we can discard the existence of feasible solutions in the Hamming ball. This is the goal of Line 8. Observe that it is very unlikely that this check is true, since it requires a solution to be feasible in the middle of a ball of unfeasible solutions. Anyway, with an appropriate structure this check requires only constant time. The third scenario is a mix of unfeasible and feasible but not $\mathbf{w}$-improving moves in M^r. In this case, feasible improving moves of higher order could exist in the ball and could be found joining several moves in M^r. Thus, we should continue with the exploration.

This exploration can be done in two ways: we could take feasible disimproving moves or unfeasible moves in M^r. We decided to explore the unfeasible moves. Two main reasons support our decision. First, it could happen that only unfeasible moves exist in M^r while feasible moves exist in the Hamming ball. Thus, exploring unfeasible moves is something the hill climber should be prepared for, anyway. Second, we do not want the hill climber to cycle (return to a previously explored solution). During the exploration of the feasible region, cycling is avoided because only strong improving or $\mathbf{w}$-improving moves are considered (notice the $>$ operator in Lines 3 and 5). If we allow the hill climber to take disimproving moves, there is no easy way to avoid cycling. Thus, if the hill climber finds that $\mathbf{w}^*$-feasible moves exist in M^r it selects one of such moves in Line 12. Otherwise, it selects an unfeasible (and not $\mathbf{w}^*$-feasible) move in Line 14 (at least one unfeasible $\mathbf{w}$-improving move must exist if the flow reaches this line). Then, it stores the last visited feasible move in variable y (Line 16) and continues the search in the unfeasible region calling the `exploreUnfeasibleRegion` procedure, which is detailed in Algorithm 3.

According to Conjecture 1, we cannot efficiently determine the presence or absence of feasible moves while the hill climber is exploring the unfeasible region. One efficient option, however, is to direct the search to find a feasible solution. This is what Algorithm 3 does. It tries to maximize the constraint function $\mathbf{g}$ in the direction of $\mathbf{w}^*$ with the hope that it will enter again the feasible region. This exploration is done taking first the strong improving moves for $\mathbf{g}$ (Line 6) and the $\mathbf{w}^*$-improving moves if no strong improving move exists (Line 8). When feasible moves appear in M^r during this exploration, only those that $\mathbf{w}$-improve the last feasible solution, y, are taken. The reason is to avoid cycling. Here again, the algorithm selects first a feasible move strong improving y (Line 13) if one exists, or a feasible move $\mathbf{w}$-improving y in second place (Line 15). Then, the algorithm enters the feasible region and runs again the `exploreFeasibleRegion` procedure. If no improving move exists for $\mathbf{g}$ in the Hamming ball of radius r around an unfeasible solution, the algorithm stops in Line 2. Observe that, in this case, a feasible move could exist in the Hamming ball. This solution, if it exists, does not $\mathbf{w}^*$-improves the current one and must be the union of several moves in M^r. This is the only scenario in which the hill climber could stop without identifying a feasible improving move in Hamming ball of radius r. But, as we announced above, we traded accuracy by efficiency in this case.

Although we assumed that a feasible solution is provided to the hill climber, it is not difficult to modify it to start in an unfeasible solution. The only thing to do is to run `explore-`

[7]If a strong improving move is taken, the starting solution is clearly a dominated solution.

Algorithm 1 Hill Climber.

```
Input: Scores vector $\mathbf{S}$, weight vector $\mathbf{w}$, initial feasible solution $x$
 1: $\mathbf{S} \leftarrow$ computeScores($x$);
 2: while true do
 3:   exploreFeasibleRegion($\mathbf{S}$, $\mathbf{w}$, $x$)
 4:   if $\forall v \in M^r$, $\mathbf{w} \cdot \mathbf{S}_v^{(\mathbf{f})}(x) \leq 0$ then
 5:     stop algorithm
 6:   end if
 7:   chose a weight vector $\mathbf{w}^* > 0$ randomly
 8:   if $\forall v \in M^r$, $\mathbf{w}^* \cdot \mathbf{g}(x \oplus v) < 0$ then
 9:     stop algorithm
10:   end if
11:   if $\exists v \in M^r$, $\mathbf{w}^* \cdot \mathbf{g}(x \oplus v) \geq 0, x \oplus v \notin X_{\mathbf{g}}$ then
12:     $t \leftarrow$ pick($\{v \in M^r | \mathbf{w}^* \cdot \mathbf{g}(x \oplus v) \geq 0, x \oplus v \notin X_{\mathbf{g}}\}$)
13:   else
14:     $t \leftarrow$ pick($\{v \in M^r | x \oplus v \notin X_{\mathbf{g}}\}$)
15:   end if
16:   $y \leftarrow x$
17:   move($\mathbf{S}$, $x$, $t$)
18:   exploreUnfeasibleRegion($\mathbf{S}$, $\mathbf{w}$, $\mathbf{w}^*$, $x$, $y$)
19: end while
```

Algorithm 2 Procedure `exploreFeasibleRegion`.

```
Input: Scores vector $\mathbf{S}$, weight vector $\mathbf{w}$, initial solution $x$
 1: while $\exists v \in M^r$, $\mathbf{w} \cdot \mathbf{S}_v^{(\mathbf{f})}(x) > 0$, $x \oplus v \in X_{\mathbf{g}}$ do
 2:   if $\exists v \in M^r$, $\mathbf{S}_v^{(\mathbf{f})}(x) > 0$, $x \oplus v \in X_{\mathbf{g}}$ then
 3:     $t \leftarrow$ pick($\{v \in M^r | \mathbf{S}_v^{(\mathbf{f})}(x) > 0, x \oplus v \in X_{\mathbf{g}}\}$)
 4:   else
 5:     $t \leftarrow$ pick($\{v \in M^r | \mathbf{w} \cdot \mathbf{S}_v^{(\mathbf{f})}(x) > 0, x \oplus v \in X_{\mathbf{g}}\}$)
 6:     report($x$);
 7:   end if
 8:   move($\mathbf{S}$, $x$, $t$)
 9: end while
10: report($x$);
```

Algorithm 3 Procedure `exploreUnfeasibleRegion`.

```
Input: Scores vector $\mathbf{S}$, weight vector $\mathbf{w}$, weight vector $\mathbf{w}^*$, initial solution $x$, last feasible solution $y$
 1: while $\forall v \in M^r$, $x \oplus v \notin X_{\mathbf{g}} \vee \mathbf{w} \cdot \mathbf{f}(x \oplus v) \leq \mathbf{w} \cdot \mathbf{f}(y)$ do
 2:   if $\forall v \in M^r$, $\mathbf{w}^* \cdot \mathbf{S}_v^{(\mathbf{g})}(x) \leq 0$ then
 3:     stop algorithm
 4:   end if
 5:   if $\exists v \in M^r$, $\mathbf{S}_v^{(\mathbf{g})}(x) > 0$ then
 6:     $t \leftarrow$ pick($\{v \in M^r | \mathbf{S}_v^{(\mathbf{g})}(x) > 0\}$)
 7:   else
 8:     $t \leftarrow$ pick($\{v \in M^r | \mathbf{w}^* \cdot \mathbf{S}_v^{(\mathbf{g})}(x) > 0\}$)
 9:   end if
10:   move($\mathbf{S}$, $x$, $t$)
11: end while
12: if $\exists v \in M^r$, $x \oplus v \in X_{\mathbf{g}}$, $\mathbf{f}(x \oplus v) > \mathbf{f}(y)$ then
13:   $t \leftarrow$ pick($\{v \in M^r | \mathbf{f}(x \oplus v) > \mathbf{f}(y), x \oplus v \in X_{\mathbf{g}}\}$)
14: else
15:   $t \leftarrow$ pick($\{v \in M^r | \mathbf{w} \cdot \mathbf{f}(x \oplus v) > \mathbf{w} \cdot \mathbf{f}(y), x \oplus v \in X_{\mathbf{g}}\}$)
16: end if
17: move($\mathbf{S}$, $x$, $t$)
```

`UnfeasibleRegion` first with no previous feasible solution y. This procedure should try to enter the feasible region and, once a feasible solution is found, the hill climber runs in the way described above.

5. EXPERIMENTAL RESULTS

In order to check the performance of the proposed hill climber, we implemented a simple Multi-Start Hill Climber algorithm that generates a feasible random solution, a weight vector $\mathbf{w}$ and, then, runs Algorithm 1. These three steps are iterated until a time limit of 1 minute is reached. Our implementation tries to avoid the re-evaluation of the feasibility of moves when all of them are in the feasible region. In order to do this, it keeps the minimum value of the constraint Scores and updates it when the Scores are changed (in constant time). It then checks if the minimum plus the current constraint value is nonnegative. If this is the case, it can certify in constant time that all moves stay in the feasible region and no re-classification of moves occurs (which is computationally costly). Thus, for unconstrained or slightly constrained problems, it recovers the constant time per move that previous work achieved.

The machine used in all the experiments has an Intel Core 2 Quad CPU (Q9400) at 2.7 GHz, 3GB of memory and Ubuntu 14.04 LTS. Only one core of the Processor is used. The algorithm was implemented in Java 1.6 and the source code is publicly available in Github[8].

We used a constrained version of MNK Landscapes [1] as benchmark functions. An MNK Landscape is a vector Mk Landscape where all $m_i = N$ for all $1 \leq i \leq d$ and each subfunction $f_i^{(l)}$ depends on x_i and other K more variables (thus, $k = K + 1$). In our experiments $K = 3$. The subfunctions $f_i^{(l)}$ and $g_i^{(l)}$ were randomly generated using integer numbers in the ranges [-49, 50] and [-50, 49]. The sum of subfunctions was not divided by N[9]. The ranges of values used for the subfunctions codomain were chosen in such a way that the fraction of unfeasible solutions is small in one case (for range [-49, 50]) and large in the other. This way we can compare what happens in slightly and highly constrained problems.

Each component f_i and g_i can be considered a shifted NKq Landscape [2]. We also focused on the *adjacent model* of NKq Landscape, where the variables each subfunction depends on are consecutive, that is, $x_i, x_{i+1}, \ldots, x_{i+K}$. This ensures that the number of subfunctions a given variable appears in is bounded by a constant and the theoretical results of Sections 2 and 3 apply.

5.1 Runtime

In this first experiment we measure the time per move of our hill climber. This experiment will provide a concrete value for the wall clock time the hill climber requires, although this value depends on the hardware and the programming skills of the developer. Before running the hill climber, it is necessary to run a *problem-dependent* initialization procedure, where the Scores to be stored in memory are determined. This procedure is run only once per problem instance and, for this reason, we did not take it into

[8]https://github.com/jfrchicanog/EfficientHillClimbers in branch *constrained-multiobjective*

[9]This is different from the original MNK Landscapes, where real values between 0 and 1 are used and the sum of subfunctions is normalized dividing it by N. However, we want to avoid floating-point inaccuracy problems.

account to compute the time per move. In our experiments this initialization procedure required between 138 and 6,308 milliseconds. The rest of the time required by the multi-start hill climber is divided by the number of moves taken and reported as average time per move.

Figure 3 shows the average time per move in microseconds for the multi-start hill climber solving constrained MNK Landscapes, where N varies from $10,000$ to $100,000$, the range of values for the subfunctions codomain is $[-49, 50]$, the number of objectives is $d = 1$ and $d = 2$, the number of constraints is $b = 1$ and $b = 2$, and the exploration radius r varies from 1 to 3. These instances are slightly constrained with around 2% of the search space being unfeasible[10]. We performed 30 independent runs of the algorithm for each configuration, and the results are the average of these 30 runs.

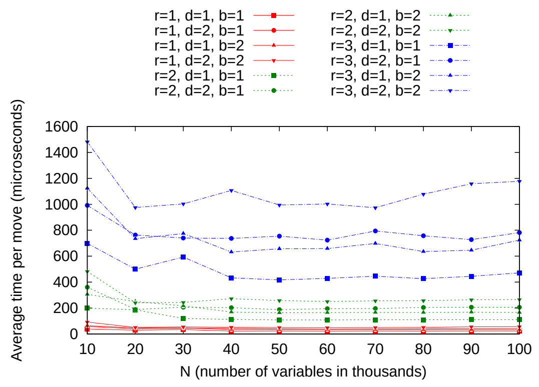


Figure 3: Average time per move in microseconds for the Multi-Start Hill Climber based on Algorithm 1 for constrained MNK Landscapes with $d = 1, 2$, $b = 1, 2$, $N = 10,000$ to $100,000$, subfunctions codomain $[-49, 50]$, and $r = 1$ to 3.

We can observe an almost constant time per move as N increases. Since this is a slightly constrained instance, the hill climber explores the feasible region most of the time, where the implementation trick mentioned above prevents the re-classification of all the moves, which takes linear time. We also observe that the radius of exploration has the highest influence on the time. This time is proportional to $(3ck)^r$ (see Section 2.1), but k is fixed in this experiment ($k = K+1 = 4$). Thus, only the number of objectives d, the number of constraints b, and the radius r externally affect the time. The objectives and constraints affect the value of c, in particular, $c = (K+1)(d+b)$. This expression appears in the base of $(3ck)^r$, but r is in the exponent. This explains why r has the highest influence. On the other hand, since the sum $d+b$ influences c, the difference between $d = 1, c = 2$ and $d = 2, c = 1$ is small in Figure 3.

In Figure 4 we show the average time per move (in microseconds) obtained by the multi-start hill climber solving a highly constrained MNK Landscape. The subfunctions codomain here is $[-50, 49]$ and N varies from $1,000$ to $10,000$. There is only one constraint and two objectives ($b = 1$, $d = 2$). The fraction of unfeasible solutions in this case is around 80%.

[10]This is an estimation computed as the fraction of unfeasible solutions obtained after generating random solutions.

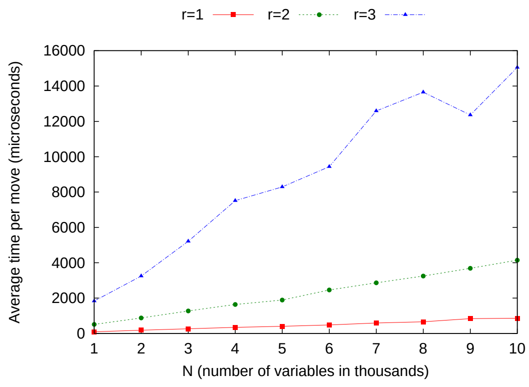


Figure 4: Average time per move in microseconds for the Multi-Start Hill Climber based on Algorithm 1 for constrained MNK Landscapes with $d = 2$, $b = 1$, $N = 1,000$ to $10,000$, subfunctions codomain $[-50, 49]$, and $r = 1$ to 3.

In this case, we can appreciate a clear linear increase in the time per move as N increases. Observe that the time per move in this figure is higher than the time per move in Figure 3, even when the search space is smaller (N is 10 times smaller). The reason is that the hill climber enters the unfeasible region very often and it needs to re-classify all the moves almost after every move, which takes linear time.

5.2 Quality of the Solutions

Exploring larger neighborhoods allows the hill climber to jump to better solutions but it also requires more time per move. Thus, increasing r does not necessarily improve the quality of the solutions when time is the stopping criterion. For each problem instance, there is a value of r for which the quality of the solutions is maximum. We will investigate here this maximum for the problems we are solving.

In Figure 5 we show the average value of the best solution found for the single-objective slightly constrained instances ($d = 1$). We can observe that the quality of the solutions increases as r does. This means that the optimum value for r is 3 or higher in this case. We can also observe how the number of constraints seems not to affect the quality of the solutions in this case and the two lines are overlapped. The number of constraints must certainly have an influence in the quality of the solutions found by the hill climber. But this influence is negligible in slightly constrained problems like these ones.

We now repeat the analysis for the bi-objective instances ($d = 2$) using the 50%-empirical attainment surfaces[11] to compare the fronts obtained when r is changed. In Figures 6(a) and 6(b) we show 50%-EAS for instances with $N = 10,000$ and $N = 50,000$, constrained by $b = 1$ and $b = 2$ functions, respectively. In this case we notice that the 50%-EAS obtained with $r = 2$ always dominates the one obtained with $r = 1$ (this happens also in the instances that are not shown). However, the 50%-EAS obtained for

[11]The 50%-empirical attainment surface (50%-EAS) limits the region in the objective space that is dominated by half the runs of the algorithm. It generalizes the concept of *median* to the multi-objective case (see [9] for more details).

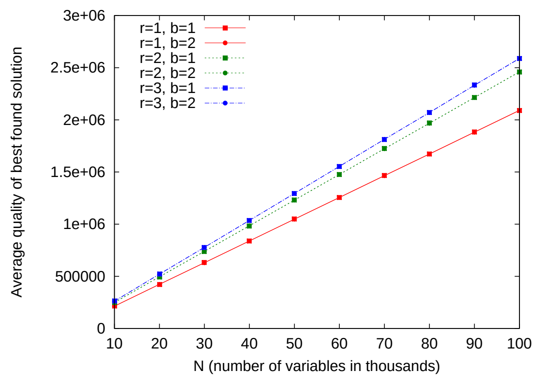
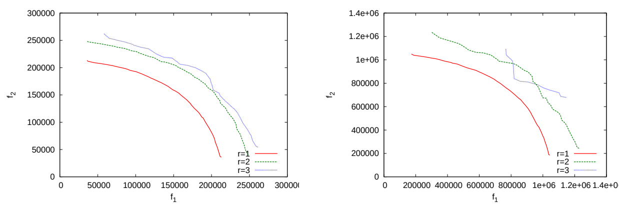


Figure 5: Average (over 30 runs) solution quality of the best solution found by the Multi-Start Hill Climber based on Algorithm 1 for a MNK Landscape with $d = 1$, $b = 1, 2$, subfunctions codomain $[-49, 50]$, $N = 10,000$ to $100,000$, and $r = 1$ to 3.



(a) $N = 10,000$, $b = 1$ (b) $N = 50,000$, $b = 2$

Figure 6: 50%-empirical attainment surfaces of the 30 runs of the Multi-Start Hill Climber based on Algorithm 1 for a MNK Landscape with $d = 2$, subfunctions codomain $[-49, 50]$, and $r = 1$ to 3.

$r = 3$ does not always dominate the one with $r = 2$. As N increases, the 50%-EAS obtained with $r = 3$ shrinks and is usually dominated by the 50%-EAS obtained with $r = 2$. Thus, we can say that $r = 2$ is the optimal value for these instances.

6. CONCLUSIONS AND FUTURE WORK

The use of problem knowledge is always beneficial to design more effective search procedures and operators. In the case of pseudo-Boolean optimization the formal structure of Mk Landscapes has been previously used to develop efficient hill climbers that explore a Hamming ball of radius r in constant time if the number of subfunctions each variable appears in is bounded by a constant. We conclude in this work that this $O(1)$ time cannot be guaranteed if we want to solve constrained problems and we proposed a new hill climber for multi-objective constrained problems whose worst case runtime is $O(n)$, which is still low compared to the complete exploration of the Hamming ball when $r > 1$.

Combining the proposed hill climber with high-level search algorithms seems a promising line of work that could produce new very efficient optimization algorithms. Another future line of research is the adaptaion of the key ingredients of this hill climber to other search spaces, like permutations or integer vectors.

Acknowledgements

This research was partially funded by the Fulbright program, the Spanish Ministry of Education, Culture and Sport (CAS12/00274), the Spanish Ministry of Economy and Competitiveness and FEDER (TIN2014-57341-R), the University of Málaga, Andalucía Tech, the Air Force Office of Scientific Research, Air Force Materiel Command, USAF (FA9550-11-1-0088), the FAPESP (2015/06462-1) and CNPq.

7. REFERENCES

[1] Hernan E. Aguirre and Kiyoshi Tanaka. Insights on properties of multiobjective MNK-landscapes. In *Proceedings of CEC*, volume 1, pages 196–203, 2004.

[2] Wenxiang Chen, Darrell Whitley, Doug Hains, and Adele Howe. Second order partial derivatives for NK-landscapes. In *Proceeding of GECCO*, pages 503–510, New York, NY, USA, 2013. ACM.

[3] Francisco Chicano, Darrell Whitley, and Andrew M. Sutton. Efficient identification of improving moves in a ball for pseudo-boolean problems. In *Proceedings of GECCO*, pages 437–444. ACM, 2014.

[4] Francisco Chicano, Darrell Whitley, and Renato Tinós. Efficient hill climber for multi-objective pseudo-boolean optimization. In *Proceedings of EvoCOP*, pages 88–103, 2016.

[5] Yves Crama, Pierre Hansen, and Brigitte Jaumard. The basic algorithm for pseudo-boolean programming revisited. *Discrete Applied Mathematics*, 29(2-3):171–185, 1990.

[6] Brian W. Goldman and William F. Punch. Gray-box optimization using the parameter-less population pyramid. In *Proceedings of GECCO*, pages 855–862, New York, NY, USA, 2015. ACM.

[7] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufman, 2004.

[8] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Springer-Verlag, 2004.

[9] J. Knowles. A summary-attainment-surface plotting method for visualizing the performance of stochastic multiobjective optimizers. In *Proceedings of ISDA*, pages 552–557, 2005.

[10] Luis Paquete, Tommaso Schiavinotto, and Thomas Stützle. On local optima in multiobjective combinatorial optimization problems. *Annals of Operations Research*, 156(1):83–97, 2007.

[11] Ivo G. Rosenberg. Reduction of bivalent maximization to the quadratic case. *Cahiers Centre Etudes Rech. Oper.*, 17:71–74, 1975.

[12] Darrell Whitley. Mk landscapes, NK landscapes, MAX-kSAT: A proof that the only challenging problems are deceptive. In *Proceedings of GECCO*, pages 927–934, New York, NY, USA, 2015. ACM.

[13] Darrell Whitley and Wenxiang Chen. Constant time steepest descent local search with lookahead for NK-landscapes and MAX-kSAT. In *Proceedings of GECCO*, pages 1357–1364, 2012.

[14] Darrell Whitley, Wenxiang Chen, and Adele E. Howe. An empirical evaluation of O(1) steepest descent for NK-landscapes. In *Proceedings of PPSN*, pages 92–101, 2012.