Optimizing One Million Variable NK Landscapes by Hybridizing Deterministic Recombination and Local Search

Francisco Chicano
University of Málaga
Bulevar Louis Pasteur, 35
Malaga, Spain 29071
chicano@lcc.uma.es

Darrell Whitley
Colorado State University
1100 Center Avenue Mall
Fort Collins, Colorado, USA 80523-1873
whitley@cs.colostate.edu

Gabriela Ochoa
University of Stirling
Stirling FK9 4LA
Stirling, Scotland, UK
gabriela.ochoa@cs.stir.ac.uk

Renato Tinós
University of São Paulo
Av. Bandeirantes, 3900
Ribeirão Preto, São Paulo, Brazil 14040901
rtinos@ffclrp.usp.br

ABSTRACT

In gray-box optimization, the search algorithms have access to the variable interaction graph (VIG) of the optimization problem. For Mk Landscapes (and NK Landscapes) we can use the VIG to identify an improving solution in the Hamming neighborhood in constant time. In addition, using the VIG, deterministic Partition Crossover is able to explore an exponential number of solutions in a time that is linear in the size of the problem. Both methods have been used in isolation in previous search algorithms. We present two new gray-box algorithms that combine Partition Crossover with highly efficient local search. The best algorithms are able to locate the global optimum on Adjacent NK Landscape instances with one million variables. The algorithms are compared with a state-of-the-art algorithm for pseudo-Boolean optimization: Gray-Box Parameterless Population Pyramid. The results show that the best algorithm is always one combining Partition Crossover and highly efficient local search. But the results also illustrate that the best optimizer differs on Adjacent and Random NK Landscapes.

CCS CONCEPTS

•**Theory of computation →Random search heuristics;**

KEYWORDS

Gray-box optimization, pseudo-Boolean optimization, Partition Crossover, Hamming Ball Hill Climber

ACM Reference format:
Francisco Chicano, Darrell Whitley, Gabriela Ochoa, and Renato Tinós. 2017. Optimizing One Million Variable NK Landscapes by Hybridizing Deterministic Recombination and Local Search. In *Proceedings of GECCO '17, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3071178.3071285

1 INTRODUCTION

NK Landscapes [5] have often been used as benchmark problems for testing algorithms designed to solve k-bounded pseudo-Boolean optimization problems, as well as evolutionary algorithms designed to work with a binary representation. There are two common forms of NK Landscapes. Adjacent NK Landscapes can be solved in polynomial time using dynamic programming. Random NK Landscapes are NP-Hard [12].

Two innovations in recent years have resulted in improved algorithms for solving k-bounded pseudo-Boolean optimization problems. One innovation is the use of lookahead methods that can identify improving moves in constant time [1, 9]. This makes traditional random mutation operators unnecessary. The second innovation is the development of Partition Crossover [8]. Partition Crossover is a deterministic form of recombination that analytically decomposes parents into *recombining components.* These recombining components, in turn, decompose the evaluation function into linearly separable subfunctions during recombination. If q recombining components are found, Partition Crossover finds the best of 2^q offspring in linear time.

By combining constant time identification of improving moves with Partition Crossover, we are able to find globally optimal solutions on Adjacent NK landscape instances with one million variables. We have developed two algorithms that combine 1) efficient local search using the identification of improving moves and 2) Partition Crossover. One algorithm is hierarchical in construction and the other algorithm is more linear in construction. We compare these algorithms to Goldman's Parameterless Population Pyramid algorithm, which is one of the best state-of-the-art algorithms for pseudo-Boolean optimization in a gray-box setting [2]. We also analyze the Local Optima Networks induced by the runs of the algorithms and capture some internal metrics to understand the working principles of the algorithms.

The rest of the paper is organized as follows. Section 2 presents the background work. The two algorithms proposed in this paper are described in Section 3. Section 4 describes the experimental studies performed and their results, explaining their meaning and providing some insight on the working principles of the algorithms. The paper finishes with some conclusions and future work outlined in Section 5.

2 BACKGROUND

A *pseudo-Boolean function* is a real-valued function of Boolean variables. A k-bounded pseudo-Boolean function f of N variables is written as a sum of M subfunctions, each one depending on at most k variables:

$$f(x) = \sum_{l=1}^{M} f_l(x), \tag{1}$$

where $f_l(x)$ is a subfunction depending on k decision variables. These functions are also called Mk Landscapes by Whitley *et al.* [11]. Well known examples of these kind of functions are NK Landscapes (with $k = K + 1$), MAX-kSAT and Unconstrained Quadratic Optimization (with $k = 2$). In Gray-Box Optimization the search algorithm has access to the structure of the objective function given in Equation (1), but makes no assumption on the subfunctions themselves.

NKQ landscapes are a kind of Mk landscape where there is a subfunction per variable ($M = N$), subfunction f_i depends on variable x_i and other $K = k - 1$ variables, and the codomain of each subfunction is the set $\{0, 1, \ldots, Q - 1\}$. The subfunctions are randomly initialized. If subfunction f_i depends on consecutive variables (x_i, x_{i+1}, . . ., x_{i+k-1}) the NKQ landscapes follow an adjacent model. If f_i depends on x_i and other $K = k - 1$ random variables, the model is random. There are some other models in between [11], but the adjacent and random models are extreme in the sense that one is very easy to solve and the other is very hard to solve. Adjacent NKQ landscapes can be optimized in polynomial time $O(N)$ using dynamic programming [12]. Random NKQ landscapes, however, are NP-hard when $K = k - 1 \geq 2$.

2.1 Variable Interaction Graph

An important tool which can be constructed under Gray Box Optimization is the Variable Interaction Graph (VIG) [11]. The VIG is a graph $G = (V, E)$, where V is the set of Boolean variables and edges E contains all the pairs of variables (x_i, x_j) that have *nonlinear interactions*. These nonlinear interactions can be captured in two ways. 1) We can assume that every pair of variables that appear together in a subfunction has a nonlinear interaction. For NK Landscapes, this assumption is virtually always true. 2) An alternative method for constructing the VIG is to convert the k-bounded pseudo-Boolean function into a Walsh polynomial [3], and then look at every pair of variables to determine if there is Walsh coefficient indexed by that pair of variables. This second method is more precise, and in some cases the difference may be significant. This alternative method is also efficient because the Walsh polynomial can be constructed in $O(N)$ time.

The following illustrates the construction of a Variable Interaction Graph for a Random NK Landscape. The NK Lanscape has 18 variables and subfunctions (numbered from 0 to 17), and $K = 2$ ($k = 3$). We will refer to variables using numbers, e.g., $9 = x_9$. The NK Landscape sums over the following 18 subfunctions:

$f_0(0, 6, 14)$	$f_5(5, 4, 2)$	$f_{10}(10, 2, 17)$	$f_{15}(15, 7, 13)$
$f_1(1, 0, 6)$	$f_6(6, 10, 13)$	$f_{11}(11, 16, 17)$	$f_{16}(16, 9, 11)$
$f_2(2, 1, 6)$	$f_7(7, 12, 15)$	$f_{12}(12, 10, 17)$	$f_{17}(17, 5, 16)$
$f_3(3, 7, 13)$	$f_8(8, 3, 6)$	$f_{13}(13, 12, 15)$	
$f_4(4, 1, 14)$	$f_9(9, 11, 14)$	$f_{14}(14, 4, 16)$	

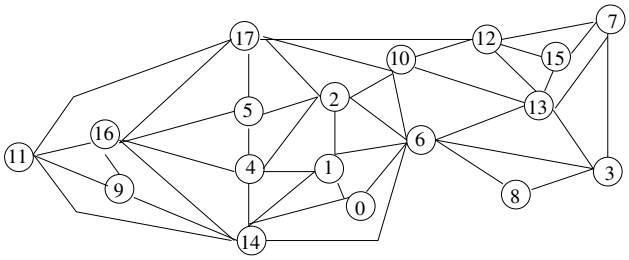


Figure 1: Sample Variable Interaction Graph (VIG).

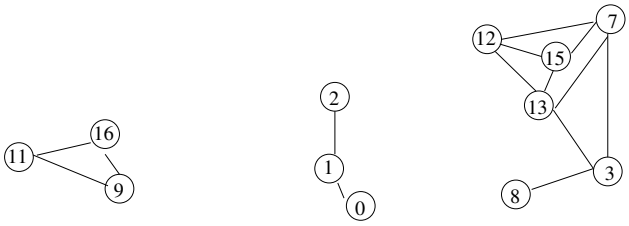


Figure 2: Recombination Graph for the solutions (parents) P_1 = 0000000000000000 and P_2 = 1111000111011101110.

From these subfunctions, assume we extract the nonlinear interactions that are shown in Figure 1. In this example, every pair of variables that appear together in a subfunction has a nonlinear interaction.

2.2 Partition Crossover

We can use the Variable Interaction Graph to construct a deterministic recombination operator: Partition Crossover (PX) [8]. If the parent strings are locally optimal then Partition Crossover acts as a tunneling algorithm that can move directly from local optima to local optima with high probability. In our case a local optimum is defined as a solution with fitness value no lower than its neighbors.

Partition Crossover is a form of *greedy, deterministic recombination*. It takes two solutions (parents), extracts the variable assignments they share, and then uses these shared variable assignments to decompose both the VIG and the evaluation function. Referring to the illustration in Figure 1, let the two parents be

$$P_1 = 0000000000000000 \text{ and } P_2 = 1111000111011101110$$

Therefore, $x_4 = x_5 = x_6 = x_{10} = x_{14} = x_{17} = 0$ in both parents. Otherwise, $x_i = 0$ in P_1 and $x_i = 1$ in P_2 for all of the other bits. Both parents reside in a hyperplane denoted by $h = ****000***0***0**0$ where $*$ denotes the bits that are different in the two solutions, and 0 marks the positions where they have the same bits values (again, without loss of generality).

We use the hyperplane $h = ****000***0***0**0$ to decompose the VIG in order to produce a *Recombination Graph*. We remove all of the variables (vertices) that have the same "shared variable assignments" and also remove all edges that are incident on the vertices corresponding to these bits. This yields the recombination graph shown in Figure 2.

We can search for connected components of the recombination graph to identify the *recombining components*. The decomposition

shown in Figure 2 results in $q = 3$ recombining components. All of the variables that appear together in the same recombining component in the recombination graph must be inherited together from one of the two parents. The recombination graph also defines a reduced evaluation function. This new evaluation function is linearly separable, and decomposes into q subfunctions defined over the recombining components.

$$g(x') = a + g_1(9, 11, 16) + g_2(0, 1, 2) + g_3(3, 7, 8, 12, 13, 15),$$

where $g(x') = f|_h(x')$ and x' is restricted to a subspace of the hyperplane h that contains the parent strings P_1 and P_2 as well as all of their potential offspring under Partition Crossover. The constant $a = f(x') - \sum_{i=1}^{3} g_i(x')$ depends on the common variables.

We can now see how Partition Crossover works. Every recombination over q recombining components induces a new *separable* function $g(x')$ that is defined as:

$$g(x') = a + \sum_{i=1}^{q} g_i(x'). \tag{2}$$

Since $g(x')$ is a separable function, Partition Crossover can be greedy and select which parent yields the best partial solution for each subfunction $g_i(x')$. The following Partition Crossover Theorem was originally proven to hold for the Traveling Salesman Problem [10]. Tinós et al. [8] have proven the following result also holds for all k-bounded pseudo-Boolean functions.

Theorem 2.1 (The Partition Crossover Theorem). *Given q linearly separable recombining components, Partition Crossover returns the best of $2^q - 2$ reachable solutions distinct from parent solution P_1 and P_2 in $O(N)$ time.*

2.3 Hamming Ball Hill Climber

For Mk landscapes, Whitley and Chen [9] proved that the location of improving moves can be determined in constant time for the Hamming distance 1 neighborhood. Two solutions are neighbors if they differ by a single bit flip. This result was later generalized by Chicano et al. [1], who proposed a hill climber that explores the solutions contained in a Hamming ball of radius r around a solution in constant time. The concept of a *Score function* is at the core of both results [4]. For $v, x \in \mathbb{B}^n$, and a pseudo-Boolean function $f : \mathbb{B}^n \to \mathbb{R}$, we denote the *Score* of x with respect to *move* v as $S_v(x)$, defined as follows:

$$S_v(x) = f(x \oplus v) - f(x), \tag{3}$$

where $\oplus$ denotes the exclusive OR bitwise operation. The Score $S_v(x)$ is the change in the objective function when we move from solution x to solution $x \oplus v$, that is obtained by flipping in x all the bits that are 1 in v. If a move can be decomposed in two non-interating moves v_1 and v_2 (they are sets of nonadjacent variables in the VIG) then the score can be written as the sum of two other scores [1]:

$$S_{v_1 \cup v_2}(x) = S_{v_1}(x) + S_{v_2}(x). \tag{4}$$

This result makes it possible to explore all the solutions at Hamming distance r or less, with the help of the scores of those moves of size at most r whose variables are connected in the VIG. If the number of subfunctions a variable appears in is bounded by a constant, then the number of scores to store in memory is $O(N)$ and the identification of an improving move can be done in constant time [1]. When Partition Crossover is combined with Hamming Ball Hill Climbing (HBHC), we can copy some scores from the parents to the child, with the goal of saving some computation[1].

In this paper, we restrict our attention to the Hamming distance 1 neighborhood. This is for two reasons. 1) While the r-ball lookahead is still $O(N)$ in complexity, the cost is exponential in r. There are also more restrictions on the form of the evaluation function for the r-ball lookahead compared to the Hamming distance 1 neighborhood. 2) The use of Partition Crossover in the current paper also appears to accelerate search more efficiently than the r-ball lookahead.

3 ALGORITHMS

We present in this section the two algorithms combining efficient Hamming distance 1 local search and Partition Crossover.

3.1 Hierarchical Recombinative Local Search (HiReLS)

One of the simplest things we can do to combine HBHC and PX is to apply local search to pairs of random solutions (to generate local optima), and then combine these local optima using PX; we can then apply local search to the resulting solution. This way we obtain a local optimal solution with a fitness value that is no worse than that of the parents. Let us name *level-1 local optima* to these solutions and *level-0 local optima* to the solutions generated after applying local search to random solutions. The average fitness value of level-1 local optima is higher than that of level-0 local optima (we are maximizing). We can obtain level-2 local optima recombining two level-1 local optima using PX and then applying HBHC. The average fitness of the solutions implicitly explored by PX is the average fitness of the parent solutions. This is a trivial consequence of the separability of $g(x')$ in (2). Thus, combining level-1 local optima should provide better solutions, in general, than combining level-1 and level-0 local optima. This idea can be iteratively applied to find local optima at different levels with increasing average fitness values. This is what HiReLS does, whose pseudocode is in Algorithm 1. Figure 3 shows a graphical illustration of the search space exploration of HiReLS. Only one solution per level needs to be stored. As the search progresses new solutions are stored in memory. In our experiments no more than 11 levels were required.

3.2 Deterministic Recombination and Iterated Local Search (DRILS)

Partition Crossover provides potentially better solutions when the number of connected components in the recombination graph is large. In order to increase this number, the solutions to recombine should not be too different, meaning that the Hamming distance between them should not be too large. Thus, the recombination with a random local optimum (as HiReLS does) is probably not the best way to exploit Partition Crossover.

DRILS is a kind of Iterated Local Search, where HBHC is used as the local search algorithm and PX is used to recombine consecutive solutions. DRILS first uses local search to find a local optimum.

[1] The details of this saving in computation are in Appendix A (supplementary material).

Algorithm 1 HiReLS

```
1:  stack ← ∅
2:  while not stopping condition do
3:     current ← HBHC(random());
4:     current.level ← 0;
5:     if stack.isEmpty() or stack.peek().level > 0 then
6:        stack.push(current);
7:     else
8:        pxSuccess ← true;
9:        while !stack.isEmpty() and pxSuccess and
          stack.peek().level = current.level do
10:          top ← stack.pop();
11:          child ← PX(top, current);
12:          pxSuccess ← child ≠ top and child ≠ current;
13:          if pxSuccess then
14:             current ← HBHC(child);
15:             current.level++;
16:          end if
17:       end while
18:       if pxSuccess then
19:          stack.push(current);
20:       end if
21:    end if
22: end while
```

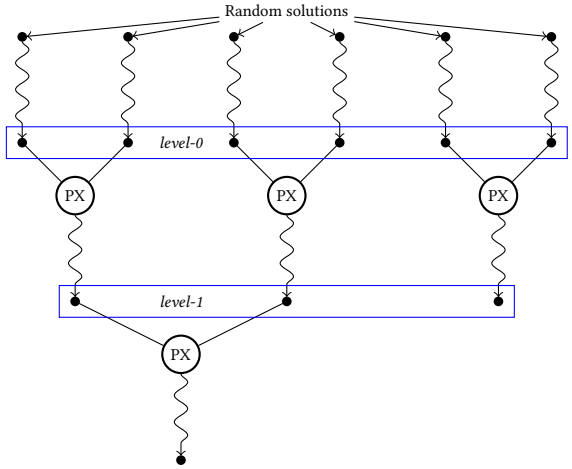


Figure 3: An illustration of HiReLS. Filled circles are local optima and curly arrows represent the HBHC.

Then DRILS perturbs the local optimum by randomly flipping αN bits, where α is a small fraction (below 0.15 in the experiments). We call the parameter α the *perturbation factor*. This process results in a soft restart and, after applying HBHC, it generates a new local optimum that should be relative close in Hamming distance to the previous local optimum. These two consecutively generated local optima can now be recombined using Partition Crossover. The offspring solution can also be improved by HBHC if necessary. The process is then iterated: the most recently discovered local optimum is perturbed and a new local optimum is generated. A graphical illustration of the algorithm is presented in Figure 4 and the pseudocode is shown in Algorithm 2.

Algorithm 2 DRILS

```
1:  current ← HBHC(random());
2:  while not stopping condition do
3:     next ← HBHC (perturb(current));
4:     child ← PX(current, next);
5:     if child = current or child = next then
6:        current ← next;
7:     else
8:        current ← HBHC(child);
9:     end if
10: end while
```

Figure 4: Graphical illustration of DRILS. Curly arrows represent HBHC while normal arrows represent a perturbation flipping αN random bits.

4 EXPERIMENTAL STUDY

In this section we analyze the performance of our two proposals on adjacent and random NKQ Landscapes. We will also compare the performance with one of the best state-of-the-art algorithms for pseudo-Boolean optimization in a gray-box setting: the Gray-Box Parameterless Population Pyramid algorithm (GB-P3) [2].

In all the experiments the radius of the neighborhood in the Hamming Ball Hill Climber was set to 1. The machine used in the experiments is a multicore machine with four Intel Xeon CPU (E5-2670 v3) at 2.3 GHz, a total of 48 cores, 64 GB of memory and Ubuntu 14.04 LTS. HiReLS and DRILS were implemented in Java 1.6 and the memory usage was limited to 3GB during all the executions. The source code is freely available in GitHub[2].

4.1 Solving Adjacent NKQ Landscapes

In a first experiment we run HiReLS, DRILS and GB-P3 using 50 different instances of the adjacent NKQ Landscapes and 10 independent runs per instance. The stopping condition for all algorithms is to reach five minutes of computation[3]. The number of variables is $N = 100,000$, the value for Q is 64 and the value for $K = k - 1$ was changed from 1 to 5 (10 instances were generated for each value of K). In the case of DRILS we used different values for the perturbation factor α: 0.005, 0.01, 0.05, 0.10 and 0.15. In Figures 5 and 6 we plot the average fitness (over 100 samples, 10 instances and 10 runs) found by the algorithms at each time. For the sake of clarity we omitted the results of DRILS of perturbation factors 0.05 and 0.10 and we only show the plots for $K = 1$ and $K = 5$.

[2] https://github.com/jfrchicanog/EfficientHillClimbers

[3] The stopping condition is arbitrary, but most of the algorithms seem to converge after five minutes. A stopping condition based on the algorithm progress should be used in future work.

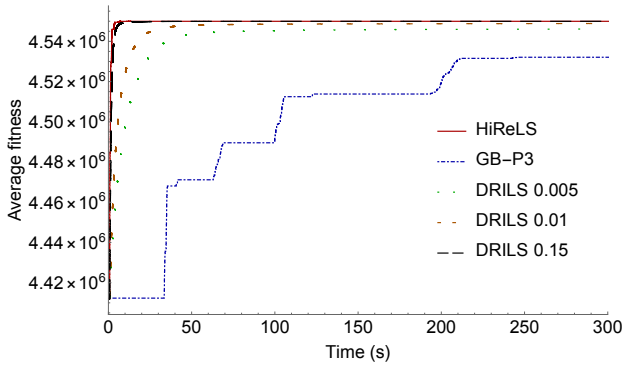


Figure 5: Average fitness over time for HiReLS, DRILS and GB-P3 in the adjacent NKQ landscapes for $K = 1$ ($k = 2$).

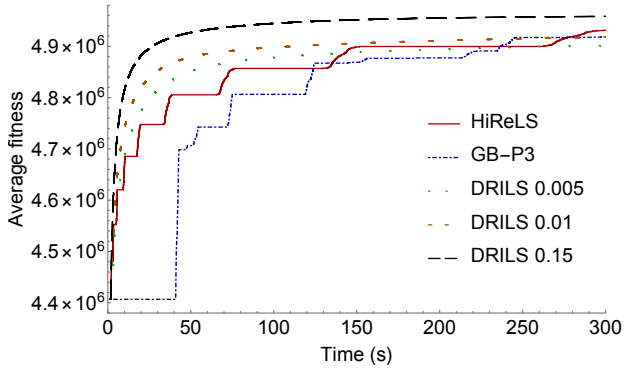


Figure 6: Average fitness value over time for HiReLS, DRILS and GB-P3 in the adjacent NKQ landscapes for $K = 5$.

Experiments indicate that HiReLS and DRILS with $\alpha = 0.15$ are the best algorithms for low values of K. When the value of K increases, making the problem harder, HiReLS is clearly outperformed by DRILS. In particular, the version with the highest perturbation factor ($\alpha = 0.15$) is always the best in this set of experiments. As the perturbation factor decreases, the performance of DRILS is worse. We can also observe that the curves of HiReLS and GB-P3 have a stair-like shape. This is a consequence of their leveled structure. Both algorithms proceed by promoting (or generating) solutions from one level to another. The solutions at the highest levels are of better quality. But reaching the highest level requires a good amount of time, which increases with K.

HiReLS and DRILS can scale to 1 million variables NKQ Landscapes. Figure 7 shows the average fitness over time of these algorithms when $K = 3$. The behaviour of the algorithms is similar to the case of $100,000$ variables with $K = 1$. We also observe the same relative performance for the other values of K we tried (from 1 to 5). We also run GB-P3 using 1 million variables, but it does not find any solution in five minutes. In fact, GB-P3 requires almost 10 hours for the initialization phase. This is the reason why its results do not appear in Figure 7.

4.1.1 Scalability to find the global optimum. We noticed that HiReLS is able to find the global optimum in a short time for low

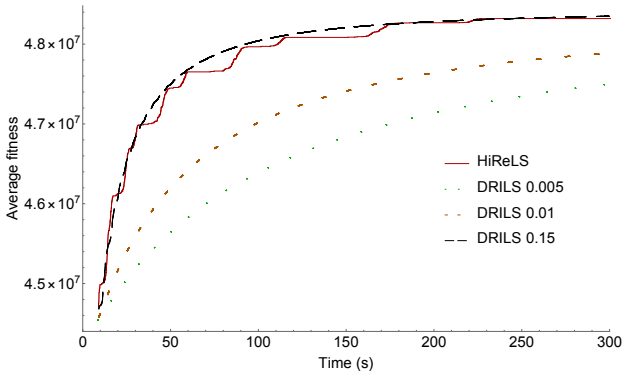


Figure 7: Average fitness value over time for HiReLS and DRILS in the adjacent NKQ landscapes for $K = 3$ and $N = 10^6$.

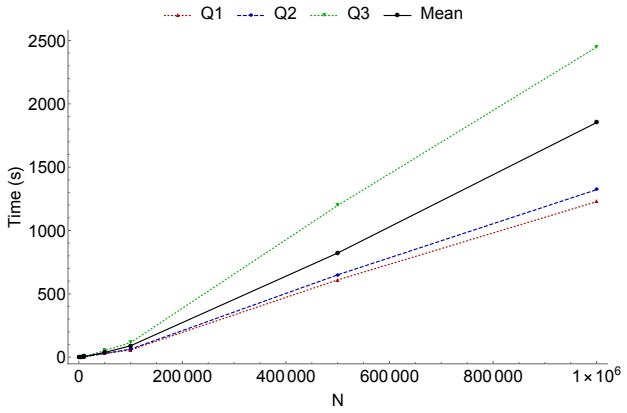


Figure 8: Quartiles and mean of the time (in seconds) required by HiReLS to find the global optimum for increasing values of N in an adjacent NKQ Landscapes with $K = 2$.

values of K. We wonder how fast can HiReLS find a global optimum as N increases. To answer this question we run HiReLS to solve Adjacent NKQ Landscapes with values of N ranging from 1,000 to 1,000,000. The stopping condition was set to reach the global optimum, which we previously computed using Wright *et al.*'s dynamic programming algorithm [12]. We generated 30 instances with $K = 2$ and run the algorithms 30 times per instance (sample size of 900 values per N). In Figure 8 we show the first and third quartiles (dotted lines), median (dashed line) and average time (solid line) in seconds required by HiReLS to solve the instances to optimality.

The required time increases linearly with the size of the instance. This is a nice finding, because HiReLS is not designed to solve the adjacent NKQ Landscapes (as is dynamic programming); it uses only the information in the VIG, and this is enough to solve the adjacent model to optimality in polynomial time.

4.1.2 The influence of K in the runtime. The large increase in the time required to find the global optimum when K is increased suggests that K has a big influence in the runtime. This is well-known in the case of the dynamic programming algorithm, where

the runtime increases linearly with N but exponentially with K. In this section we analyze the time required by HiReLS to find the global optimum when K is increased. For this experiment we used $N = 1,000$ and K varies from 1 to 5. For each value of K we generated 30 random instances and we run HiReLS 30 times per instance (a sample of 900 values per K). In Figure 9 we show the first and third quartiles (dotted lines), the median (dashed line) and the average (solid line) of each sample for each value of K. The time (in seconds) is shown in logarithmic scale. We can observe a growth in time that is slightly higher than exponential, confirming our hypothesis about the influence of K in the runtime.

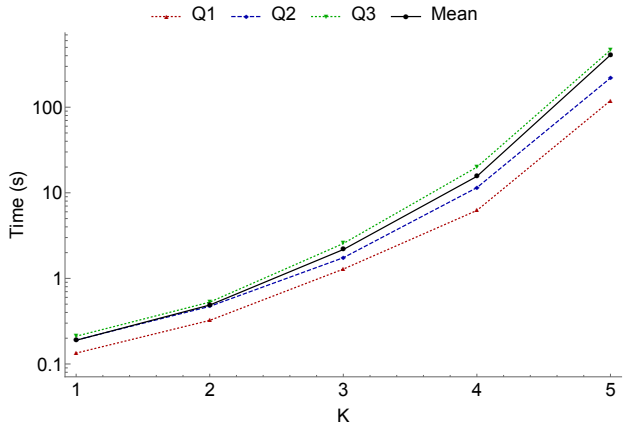


Figure 9: Quartiles and mean of the time (in logarithmic scale) required by HiReLS to find the global optimum for increasing values of K in an adjacent NKQ Landscapes with $N = 1,000$.

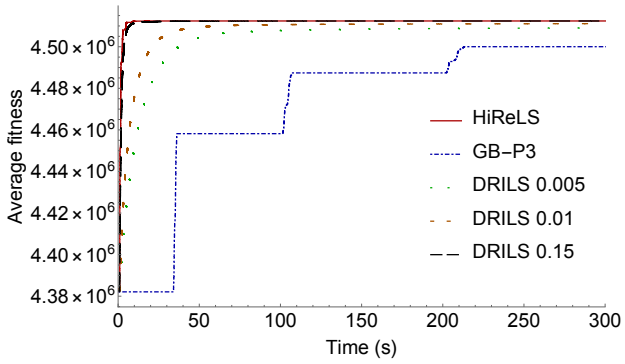


Figure 10: Average fitness value over time for HiReLS, DRILS and GB-P3 in the random NKQ landscapes for $K = 1$.

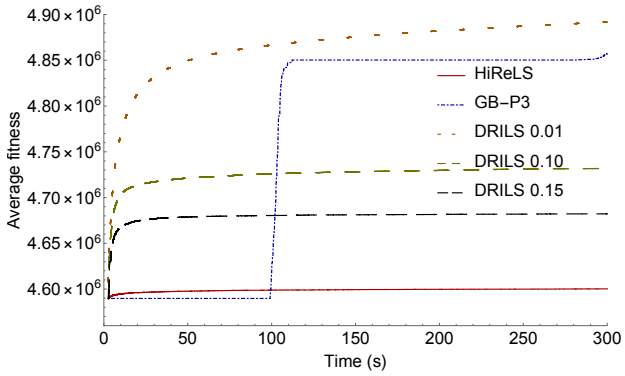


Figure 11: Average fitness value over time for HiReLS, DRILS and GB-P3 in the random NKQ landscapes for $K = 5$.

4.2 Solving Random NKQ Landscapes

This section focuses on solving the random model of NKQ Landscapes. For $K \geq 2$ ($k \geq 3$), this model is NP-hard and we want to evaluate how our algorithms perform on NP-hard problems. We run HiReLS, DRILS and GB-P3 to solve 50 different instances of the random NKQ Landscapes (10 instances for each value of K). In all the cases 10 independent runs were executed per algorithm and the average was computed with a sample of 100 values. All the algorithms are configured to stop after five minutes of computation. In Figures 10 and 11 we show the plots with the average fitness for $K = 1$ and $K = 5$, respectively. Three variants of DRILS with different perturbation factors are shown (the most representative ones), while the other two are omitted.

HiReLS has a completely different behavior for $K = 1$ and $K > 1$. When $K = 1$ HiReLS is one of the best algorithms, together with DRILS for higher values of the perturbation factor. However, when $K > 1$, HiReLS is the worst algorithm. The value of K has a dramatic impact in the performance of HiReLS in the random model. This is not observed in the adjacent model. Interestingly, the random model with $K = 1$ is solvable in polynomial time. Thus, we conclude that HiReLS is among the best algorithms solving "easy" problems. For $K > 1$ the best algorithm is always DRILS with an appropriate perturbation factor. As K increases, the fraction of flipped variables during the perturbation should be decreased. This was also observed in the adjacent model. We will analyze the reason for this behavior in Section 4.4. Finally, GB-P3 is always outperformed by a DRILS with an appropriate factor.

The results of HiReLS and DRILS for 1 million variables random NKQ Landscapes, shown in Figure 12, follow the same trend as in the case of $N = 100,000$ variables. For all the values of K we observe the same relative performance for $N = 100,000$ and $N = 1,000,000$. We couldn't run GB-P3 due to its long initialization time.

4.3 Local Optima Network Visualization

In order to better understand the search dynamics of HiReLS and DRILS, we visualized the Local Optima Networks (LON) [7] induced by example runs of the algorithms. A Local Optima Network (LON) is a graph where nodes are local optima and edges represent transitions among them with a given search operator. In our case the local optima are traced during one run of each algorithm solving 1 million variable NKQ Landscapes. In HiReLS, transitions between local optima can only occur via recombination, which is represented by red arcs in the LONs (an arc from each parent to the child). In DRILS there are two possible transitions among local optima: using Partition Crossover and using a perturbation followed by local search. Crossover transitions are visualized with

red arcs and perturbation followed by local search are represented with blue arcs. Figures 13 and 14, show representative LONs for the two algorithms and landscape models.

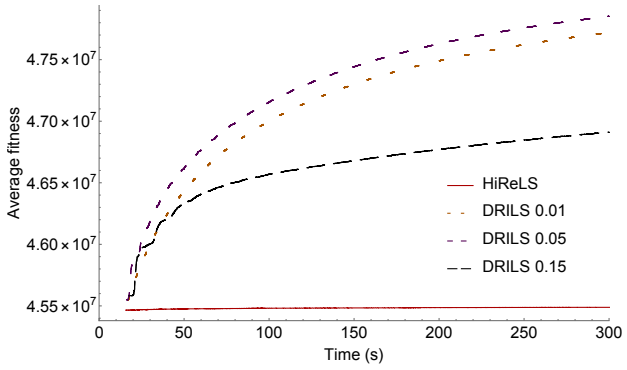
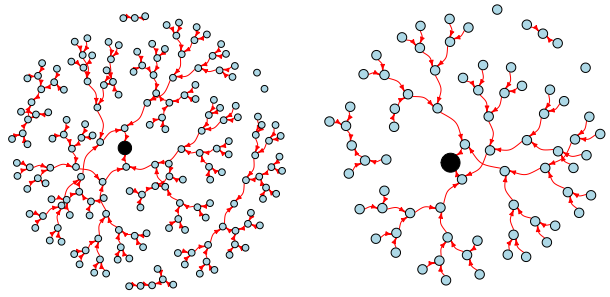
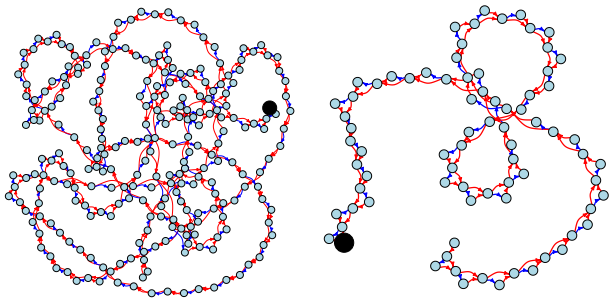


Figure 12: Average fitness value over time for HiReLS and DRILS in the random NKQ landscapes for $K = 3$ and $N = 10^6$.



(a) Adjacent (b) Random

Figure 13: Local Optima Network for one run of HiReLS with $N = 10^6$, $K = 3$. The best local optimum is highlighted in black.



(a) Adjacent (b) Random

Figure 14: Local Optima Network for one run of DRILS with $N = 1,000,000$, $K = 3$.

The LONs illustrate quite well how the algorithms work (compare these LONs with Figures 3 and 4). In both algorithms the number of local optima visited during search is lower for the random model. Ochoa *et al.* [7] showed that the random model of NK Landscapes has fewer local optima than the adjacent model. The basins of attraction must be larger in the random model, thus local search requires more steps to reach local optima. Since we stop the algorithms after five minutes, fewer local optima can be visited in the random model.

For HiReLS (Figure 3) we observe some isolated connected components. These local optima correspond to solutions that were used to build a local optimum for which Partition Crossover failed to generate an offspring (it cannot improve the parents). The presence of many isolated components reduces the efficacy of the algorithm. In DRILS all the recombinations were successful (a crossover failure should appear in the LON as a node with one incoming and one outgoing blue arc). In fact, the goal of adding a perturbation to DRILS is to increase the probability of successful recombination (because the solutions are near enough in the search space). We can see in the LON that this perturbation is working properly.

4.4 On the Perturbation Factor of DRILS

We can easily fix the perturbation factor of DRILS using an automatic parameter tuning tool, like iRace [6]. However, we noticed a trend in the optimal value of the perturbation factor. As K increases the optimal value is lower. We think the performance of the algorithm is related to the number of components identified by Partition Crossover. In order to find evidences of this hypothesis we summed the number of connected components found by each application of PX in every single run, we averaged these numbers over all the independent runs on the same instance and the same perturbation factor, and we computed a rank of perturbation factors in terms of connected components (the lower the rank the higher the number of connected components). We did the same for the final fitness value (after five minutes). Then, we averaged these ranks over all the instances with the same value of K. The resulting averaged ranks are in Table 1 for the random NKQ Landscapes with 1 million variables.

We observe a direct correlation between the rank by the fitness value and the rank by the number of connected components. In the cases where they differ (highlighted in boldface), the difference is at most one unit. Thus, we conclude that increasing the number of connected components found by PX should improve performance. The average number of connected components $\overline{q}$ of the recombination graph is shown in Table 2. There are high values of $\overline{q}$ for some combinations of K and α. For example, $\overline{q} = 16,259$ for $K = 1$ and $\alpha = 0.15$, which means that the best of more than $2^{16,259} = 10^{4,894}$ solutions are obtained in some applications of Partition Crossover. We also notice that the best rank in Table 1 does not always correspond with the highest value of $\overline{q}$ in Table 2. The ranking of Table 1 is computed using the sum of connected components found by all the applications of PX in one run, which takes into account also the number of times that PX is applied. Table 2 shows averages and does not consider how many times PX is applied. The number of times that PX is successfully applied is also important to performance.

Table 1: Average ranking of the perturbation factor values (across rows) in terms of sum of number of components found by PX and average final fitness value in the runs of DRILS for the random NKQ Landscapes with $N = 10^6$.

	Perturbation Factor (α)									
	0.005		0.01		0.05		0.10		0.15	
K	PX	Fit	PX	Fit	PX	Fit	PX	Fit	PX	Fit
1	5.0	5.0	4.0	4.0	3.0	3.0	2.0	2.0	1.0	1.0
2	5.0	5.0	4.0	4.0	**1.1**	**1.3**	**1.9**	**1.7**	3.0	3.0
3	**3.0**	**3.3**	2.0	2.0	1.0	1.0	**4.0**	**3.7**	5.0	5.0
4	**2.0**	**3.0**	1.0	1.0	**3.0**	**2.0**	4.0	4.0	5.0	5.0
5	**1.9**	**2.7**	**1.1**	**1.0**	**3.0**	**2.3**	4.0	4.0	5.0	5.0

Table 2: Average number of components $\overline{q}$ found by Partition Crossover in the runs of DRILS for the random NKQ Landscapes with $N = 10^6$ and different values for α and K.

	Perturbation Factor (α)				
K	0.005	0.01	0.05	0.10	0.15
1	683	1,314	6,059	11,442	16,259
2	967	1,772	6,938	11,426	13,428
3	1,041	1,810	4,970	3,639	2,367
4	993	1,657	1,780	661	301
5	903	1,344	517	100	38

It is not hard to theoretically compute the optimal value for the perturbation factor for the Adjacent NK Landscape if no hill climbing is applied to the perturbed solution (see Appendix B in supplementary material). The optimal perturbation factor is $1/(K+1)$ and the expected number of components is approximately $Ne^{-1}/(K+1)$. DRILS applies HBHC after the perturbation and, for this reason, the previous expressions are not strictly correct. However, as we increase K the optimal perturbation factor decreases and the same happens with the number of components (and performance). In the case of the Random NK Landscape the theoretical prediction is not so easy to do, but we also observe in Table 1 and Figures 10 and 11 an empirical inverse relationship between the optimal perturbation factor and performance.

5 CONCLUSIONS

We have presented two algorithms, HiReLS and DRILS, combining two gray-box operators: Hamming Ball Hill Climbing and Partition Crossover. These two operators, especially Partition Crossover, are able to avoid exploring many low quality solutions thanks to the use of the VIG. In a typical 5-minutes run of DRILS solving random NKQ Landscapes with $N = 1,000,000$, $K = 3$ and $\alpha = 0.05$, it applied 48 successful recombinations of local optima, with an average of 4,970 components found in each of them (see Table 2), discarding $2^{4,970}$ solutions in each recombination. HBHC found 98 local optima, discarding 1 million solutions in each of them. In total, the number of implicitly considered solutions in 300 seconds is around $10^{1,497}$. This is equivalent to evaluating $10^{1,485}$ solutions per nanosecond using a black-box algorithm, which is impossible using current technology. We have also shown that HiReLS and DRILS beat Goldman's Gray-Box Parameterless Population Pyramid (a state-of-the-art algorithm for pseudo-Boolean optimization) in random and adjacent NKQ Landscapes.

Overall, we conclude that DRILS is the best algorithm in practice from the ones compared here. In particular, it has been always the best in the random model, which is NP-hard. One of the disadvantages of DRILS is that it contains a parameter that has to be tuned: the perturbation factor. We observed in the experiments that this parameter can have a high impact in the performance. The optimal value for the perturbation factor of DRILS depends on the variable interaction graph of the instance. Future work should address how to set a near optimal value for this parameter, or even how to optimally perform the perturbation in DRILS using the information contained in the variable interaction graph.

Industrial problems are not as structured as the adjacent NKQ Landscapes or as random as the random model. Future work should study how the proposed algorithms perform in semi-structured instances that reflect industrial and real-world problems.

ACKNOWLEDGEMENTS

Funding was provided 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, (FA9550-11-1-0088), the Leverhulme Trust (RPG-2015-395), the FAPESP (2015/06462-1) and CNPq (304400/2014-9).

REFERENCES

[1] Francisco Chicano, Darrell Whitley, and Andrew M. Sutton. 2014. Efficient identification of improving moves in a ball for pseudo-boolean problems. In *Proceedings of GECCO, Vancouver, BC, Canada, July 12-16, 2014*, Dirk V. Arnold (Ed.). ACM, NY, USA, 437–444. DOI: http://dx.doi.org/10.1145/2576768.2598304

[2] Brian W. Goldman and William F. Punch. 2015. Gray-Box Optimization Using the Parameter-less Population Pyramid. In *Proceedings of GECCO*. ACM, New York, NY, USA, 855–862. DOI: http://dx.doi.org/10.1145/2739480.2754775

[3] Robert B. Heckendorn, Soraya Rana, and Darrell Whitley. 1999. Polynomial Time Summary Statistics for a Generalization of MAXSAT. In *Proceedings of GECCO - Volume 1*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 281–288. http://dl.acm.org/citation.cfm?id=2933923.2933952

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

[5] Stuart A. Kauffman and Edward D. Weinberger. 1989. The NK model of rugged fitness landscapes and its application to maturation of the immune response. *Journal of Theoretical Biology* 141, 2 (1989), 211 – 245. DOI: http://dx.doi.org/10.1016/S0022-5193(89)80019-0

[6] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez-Cáceres, Mauro Birattari, and Thomas Stützle. 2016. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives* 3 (2016), 43 – 58. DOI: http://dx.doi.org/10.1016/j.orp.2016.09.002

[7] Gabriela Ochoa, Francisco Chicano, Renato Tinós, and Darrell Whitley. 2015. Tunnelling Crossover Networks. In *Proceedings of GECCO*. ACM, New York, NY, USA, 449–456. DOI: http://dx.doi.org/10.1145/2739480.2754657

[8] Renato Tinós, Darrell Whitley, and Francisco Chicano. 2015. Partition Crossover for Pseudo-Boolean Optimization. In *Proceedings of FOGA*. ACM, New York, NY, USA, 137–149. DOI: http://dx.doi.org/10.1145/2725494.2725497

[9] Darrell Whitley and Wenxiang Chen. 2012. Constant time steepest descent local search with lookahead for NK-landscapes and MAX-kSAT. In *Proceedings of GECCO*. ACM, NY, USA, 1357–1364. DOI: http://dx.doi.org/10.1145/2330163.2330351

[10] Darrell Whitley, Doug Hains, and Adele Howe. 2009. Tunneling Between Optima: Partition Crossover for the Traveling Salesman Problem. In *Proceedings of GECCO*. ACM, NY, USA, 915–922. DOI: http://dx.doi.org/10.1145/1569901.1570026

[11] L. Darrell Whitley, Francisco Chicano, and Brian W. Goldman. 2016. Gray Box Optimization for Mk Landscapes (NK Landscapes and MAX-kSAT). *Evolutionary Computation* 24 (Jan-09-2016 2016), 491 – 519. DOI: http://dx.doi.org/10.1162/EVCO_a_00184

[12] Alden Wright, Richard Thompson, and Jian Zhang. 2000. The computational complexity of NK fitness functions. *IEEE Transactions on Evolutionary Computation* 4, 4 (2000), 373–379.