Tunneling Between Plateaus: Improving on a State-of-the-Art MAXSAT Solver using Partition Crossover

Wenxiang Chen
and Darrell Whitley
Colorado State University
Fort Collins, Colorado, USA
chenwx@cs.colostate.edu
whitley@colostate.edu

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

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

ABSTRACT

There are two important challenges for local search algorithms when applied to Maximal Satisfiability (MAXSAT). 1) Local search spends a great deal of time blindly exploring plateaus in the search space and 2) local search is less effective on application instances. This second problem may be related to local search's inability to exploit problem structure. We propose a genetic recombination operator to address both of these issues. On problems with well defined local optima, *partition crossover* is able to "tunnel" between local optima to discover new local optima in O(n) time. The *PXSAT* algorithm combines partition crossover and local search to produce a new way to escape plateaus. Partition crossover locally decomposes the evaluation function for a given instance into independent components, and is guaranteed to find the best solution among an exponential number of candidate solutions in O(n) time. Empirical results on an extensive set of application instances show that the proposed framework substantially improves two of best local search solvers, AdaptG^{2}WSAT and Sparrow, on many application instances. PXSAT combined with AdaptG^{2}WSAT is also able to outperform CCLS, winner of several recent MAXSAT competitions.

KEYWORDS

MAX-kSAT, partition crossover. local search, hybrid genetic algorithms

ACM Reference Format:
Wenxiang Chen and Darrell Whitley, Renato Tinós, and Francisco Chicano. 2018. Tunneling Between Plateaus: Improving on a State-of-the-Art MAXSAT Solver using Partition Crossover. In *GECCO '18: Genetic and Evolutionary Computation Conference, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3205455.3205482

1 INTRODUCTION

Boolean Satisfiability (SAT) is the first problem proven NP-Complete [9]. Maximum Satisfiability (MAXSAT) is the optimization version of SAT. The goal of MAXSAT is to find an assignment that satisfies the maximal number of clauses. An efficient way of solving many optimization problems, such as automated design debugging [7] and the maximum clique problem [16], is by converting these problems into MAXSAT and applying a MAXSAT solver. MAXSAT-based approaches can even outperform specialized solvers in areas like nonlinear dimensional reduction [6] and Bayesian network learning [4].

The two major search paradigms for solving MAXSAT instances are Systematic Search such as branch-and-bound solvers [1] and local search algorithms [24]. Local search can reliably solve uniform random instances with one million variables to optimality in recent SAT competitions[1]. Despite its demonstrated raw power in solving difficult uniform random instances, local search still suffers from the following two prominent issues.

(1) Local search solvers frequently encounter a sequence of states where it is difficult to reduce the number of unsatisfied clauses. Moving through these regions, called *plateau moves*, usually dominates the running time of local search solvers [11, 22]. Furthermore, the valuable history of information accumulated after high quality solutions are visited is typically abandoned, which seems unwise.
(2) Local search solvers have poor performance on application SAT instances. Application SAT instances that have been converted to MAXSAT problems typically have internal structure. In particular, *decomposability* focuses on how well the variable interactions of an application instance can be decomposed. Decomposability has been extensively studied and exploited by systematic SAT solvers with success [2, 12]. In constrast, local search solvers ignore structure and the potential decomposability of application instances.

We present a new framework called *PXSAT*, based on the recombination operator *Partition Crossover (PX)* [23]. PXSAT is a form of hybrid genetic algorithm that combines recombination with local search. However, unlike most recombination operators, partition crossover is deterministic in its selection of crossover points, and partition crossover offers performance guarantees. Partition crossover is also explicitly designed to be used in combination with local search.

PX takes as input two solutions which are local optima, or otherwise are good solutions found on a plateau of the search space. PX is often able to create a "tunnel" that directly moves from two known locally optimal solutions to arrive at new local optima in

[1] http://satcompetition.org/

GECCO '18, July 15–19, 2018, Kyoto, Japan

ACM ISBN 978-1-4503-5618-3/18/07…$15.00
https://doi.org/10.1145/3205455.3205482

$O(n)$ time. Partition Crossover (including the IPT operator) has been applied successfully on the Traveling Salesman Problem [28–30] as well as on NK-Landscapes [8]. It has not previously been applied to MAXSAT. Previous researchers [21, 26] have reported that high quality local optima typically share partial solutions with optimal solutions. Using PX in combination with local search algorithms also has the potential of finding improving moves that tunnel from one plateau to a better plateau by changing hundreds (or even thousands) of variables at the same time.

PX can also be used to exploit the decomposability of MAXSAT application instances. PX fixes what can be considered pseudo backbone variables (i.e., variable assignments shared among local optima [20]), to locally *decompose* the Variable Interaction Graph (VIG) into q components that are independent from each other. PX then recombines partial solutions from different components such that the best solution among all possible 2^q reachable solutions is found. This occurs in $O(n)$ time. Previous studies report that many application instances do have high decomposability and often can be decomposed into hundreds or thousands of components [5, 31]. These results suggest that PX has the potential to be very useful on MAXSAT application instances.

While combining local search and PX is in principle simple, doing so while also controlling execution costs is not trivial. State-of-the-art local search MAXSAT solvers are highly optimized, so that each improving move takes only $O(1)$ time [25]. Each application of PX takes $O(n)$ time. However it seems likely this cost can be reduced with further research. In the current implementation, PX is only triggered when there is no improvement after αn local search moves.

Another related problem in utilizing PX is deciding what candidate solutions to recombine. When there are well defined local optima, this is less of an issue. Selecting which parents to recombine when the parents reside on a plateau is more difficult. However, new theoretical finding can guide the design of the PXSAT to avoid triggering unproductive applications of Partition Crossover.

Empirical results on an extensive set of application instances shows that combining PX with local search algorithms can yield substantially better results than local search alone. We improve two of best local search solvers, AdaptG^{2}WSAT and Sparrow. PXSAT combined with AdaptG^{2}WSAT is also able to outperform CCLS, winner of several recent MAXSAT competitions.

2 VARIABLE INTERACTION AND TUNNELING

Let $f(x)$ be the evaluation function for an assignment $x \in \mathbb{B}^n$, where $f(x)$ counts the number of unsatisfied clauses. Consider the following MAX-3SAT function composed of the following clauses. Denote this function *Example 1:*

a: 1 -3 6	**l: -6 10 13**	**q: -11 16 17**	**v: -15 -7 -13**
b: 2 -1 6	**m: 8 -18 6**	**r: 12 -10 17**	**w: 16 -9 -11**
c: -1 2 4	**n: 7 -12 -15**	**s: -13 -12 15**	**x: 17 -5 -16**
d: -4 1 14	**o: 9 11 14**	**t: 14 -4 16**	**y: -18 -7 13**
e: -5 4 2	**p: -10 -2 17**	**u: -9 14 16**	**z: 3 6 -14**

A clause **a: 1 -3 6** has a label **a** and variables **1 -3 6**. A positive variable (e.g. **1**) is satisfied by an assignment of True. A negated variable (e.g. **-3**) is satisfied by a False assignment. Each clause is in Conjunctive Normal Form; at least one literal must be satisfied for the clause to be satisfied. The goal is to maximize the number of satisfied clauses. Let m denote the number of clauses, and n denote the number of variables.

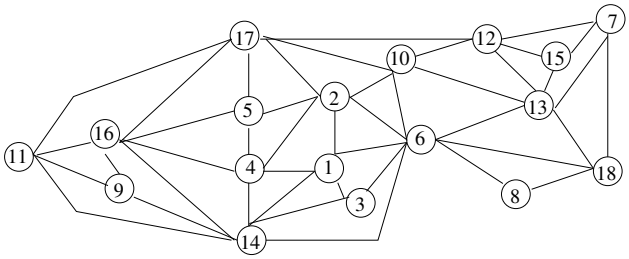

Figure 1: An illustration of the VIG.

From these clauses, we can extract the nonlinear interactions between the variables. An exact way to compute the nonlinear interactions is to use a discrete Fourier transform to generate a discrete Fourier polynomial; this can be done in $O(n)$ time assuming $m = O(n)$. We can be less exact and assume that if two variables appear together in a single clause, there is a nonlinearity between those variables. The true nonlinear interactions must be a subset of this set. This leads to the following definition:

Definition 2.1 (Variable Interaction Graph (VIG)). A variable interaction graph has a set of vertices which are the variables of a MAXSAT instance. If two variables, x_i and x_j appear together in a clause, there is an edge $e_{i,j}$ in the VIG.

The VIG has at most $3m = O(n)$ edges for MAX-3SAT. Figure 1 presents the *VIG* for *Example 1.* Assume we have two candidate solutions $P1$ and $P2$ to *Example 1* that have been found by local search, where neither solution can be improved by a single bit flip.

$$P1 = 00000\ 00000\ 00000\ 000$$
$$P2 = 11100\ 01110\ 11101\ 101$$

$P1$ satisfies all of the clauses except clause **o** but flipping bits 9 or 11 or 14 causes clauses **q** or **u** or **z** respectively to be unsatisfied. $P2$ satisfies all of the clauses except clause **v** but flipping bits 15 or 13 or 7 causes clauses **s** or **y** or **m** to be unsatisfied.

Clearly, $x_4 = x_5 = x_6 = x_{10} = x_{14} = x_{17} = 0$ in $P1$ and $P2$. For all other bits, $x_i = 0$ in $P1$, and $x_i = 1$ in $P2$. The two solutions are contained in the hyperplane $^{***}000^{***}0^{***}0^{**}0^{*}$ where $*$ denotes the bits that are different in the two solutions, and 0 marks the positions where variables share the same assignment.

We use the hyperplane $^{***}000^{***}0^{***}0^{**}0^{*}$ to decompose the VIG to generate a *recombination graph.* We remove all of the variables (vertices) that have the same assignments and remove the edges incident on the removed vertices. The *Recombination Graph* is shown in Figure 2.

The recombination graph breaks the VIG into connected subgraphs, which we will define as *recombining components.* In Figure 2 there are $q = 3$ recombining components. Variables that are connected in the recombination graph represent complementary partial solutions. The recombination graph also decomposes the evaluation function $f(x)$ into linearly separable subfunctions. Thus, in

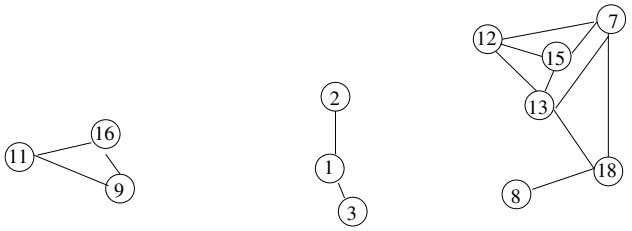

Figure 2: The Recombination Graph with three separable recombining components for the parent $P1$ = 00000 00000 00000 000 **and** $P2$ = 11100 01110 11101 101.

Example 1 we can define a new subfunction $g(x')$.

$$g(x') = a + g_1(x_9, x_{11}, x_{16}) + g_2(x_1, x_2, x_3) + g_3(x_7, x_8, x_{12}, x_{13}, x_{15}, x_{18})$$

where a is a constant and $g(x') = f(x)$ but where the domain of function $g(x')$ is restricted to the largest hyperplane subspace containing strings $P1$ and $P2$. This means that x' samples only a proper subset of the variables represented by x; the other variables that are shared in common by $P1$ and $P2$ have been fixed in value. Since clause **o** is unsatisfied by $P1$ and clause **v** is unsatisfied by $P2$, and because **o** and **v** are in different components of recombination graph in Figure 2, the offspring (e.g., 00000 00010 10000 100) is guaranteed to improve upon both parents. In this case, PX instantly jumps to the global optimum with all clauses satisfied.

In general the recombination graph induces a new evaluation function

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

where each subfunction $g_i(x')$ evaluates one connected recombining component of the recombination graph. Let G_i denote the recombining component corresponding to subfunction $g_i(x')$. Note that if there are 2 parents and q recombining components, these partially solutions can be recombined in 2^q ways; we refer to this as the set of *reachable solutions*. We can now prove the following result:

Theorem 2.2 (PX Theorem). *Given a recombination graph with q recombining components, Partition Crossover (PX) returns the best of 2^q reachable solutions in O(n) time.*

Proof. Because the function $g(x')$ is linearly separable, we can greedily select the best partial solution from $P1$ and $P2$ independently for each subfunction g_i. The q greedy choices yields the best of 2^q reachable solutions. □

Of course, an improvement only occurs if $P1$ and $P2$ have *recombining components* that also have different evaluations, even when $f(P1) = f(P2)$. But we can also say something more about the potential offspring that are generated by Partition Crossover.

First consider the simple case where there are only 2 recombining components, and thus there are only 2 potential offspring that are different from the parents. Assume the parents are P_1 and P_2 and two potential are C_1 and C_2. Under the cost function $f(\mathbf{x})$ [27]:

$$f(P_1) + f(P_2) = f(C_1) + f(C_2)$$

If there are only 2 recombining components, the offspring can inherit from either P_2 or from P_2 in each components. This generates 2^2 combinations, but 2 of the 4 are the original parents. The other two are offspring C_1 and C_2. Thus, by definition: $f(P_1) + f(P_2) = f(C_1) + f(C_2)$.

Next, in this paper we generalized this idea. As already noted, given q recombining components the number of potential offspring produced by recombining parents P_1 and P_2 is 2^q. Using simple counting arguments that average over all possible offspring yields the following result:

$$\frac{f(P_1)}{2} + \frac{f(P_2)}{2} = \frac{1}{2^q} \sum_{i=1}^{2^q} f(C_i)$$

This result has special significant for MAXSAT. Assume that $f(P1) = f(P2)$. However, if *any* of the offspring represents a disimproving move, there must also exist an offspring that represents an improving move. This makes Partition Crossover very different than local search, where the discovery of a disapproving move says nothing about the existence of an improving move.

The Partition Crossover operator is also highly exploitive in nature: crossover retains the best combination of "alleles" already present in the parents, but Partition Crossover can never generate new "alleles" that are not found in the parents.

2.1 Tunneling between Local Optima

Tunneling methods in the form of PX are able to take two solutions as input that are locally optimal, and return a new solution that is also a local optimum in $g(x')$. Thus, tunneling is able to do something that local search cannot: move directly from known local optima to new local optima in one step.

A Partition Crossover operator (e.g., IPT crossover) has already been developed for the Traveling Salesman Problem (TSP) and is a critical part of the Lin Kernigham Helsgaun (LKH) algorithm [30], the best iterated local search algorithm for the TSP. Tunneling methods have also been developed for general k-bounded pseudo-Boolean optimization problems [23].

Because MAX-kSAT problems are characterized by numerous large plateaus, it is more difficult to characterize the "tunneling" behavior of PX on MAX-kSAT. But for Weighted MAX-kSAT (and k-bounded pseudo-Boolean optimization problems in general) we can prove the following result.

Theorem 2.3. *Assume that solutions $P1$ and $P2$ are well defined local optima on a Weighted MAX-kSAT instance. All of the reachable solutions are also local optima under function $g(x')$.*

Proof. Assume bit x_b is referenced by both $f(x)$ and $g(x')$. Because $g(x')$ is linearly separable x_b appears in only one subfunction $g_i(x')$. Because $g_i(x')$ is a linearly separable subfunction relative to $P1$ and $P2$, if $f(x)$ is locally optimal for $P1$ and $P2$ the subfunction $g_i(x')$ must also be locally optimal for $P1$ and $P2$. It follows that all offspring that inherit an assignment from $P1$ or $P2$ for $g_i(x')$ are also locally optimal under $g(x')$. □

To be clear, a solution that is locally optimal in $g(x')$ might not be locally optimal in $f(x)$, but if a solution is not locally optimal in $f(x)$ the improving move can only result from flipping a bit

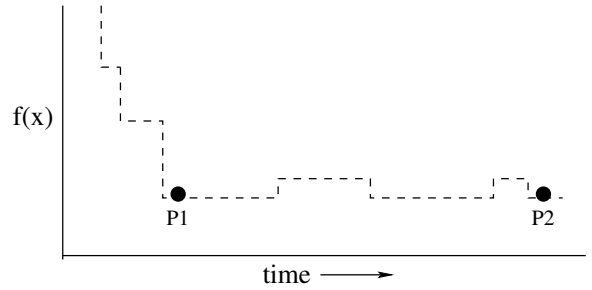

Figure 3: This figure illustrates how PX is combined with local search. The dashed line tracks changes in $f(x)$. When a plateau is reached, a solution $P1$ is captured. After αn moves with no improving moves, another solution $P2$ is selected, and PX is used to recombine $P1$ and $P2$.

assignment shared in common by $P1$ and $P2$. We have examined hundreds of k-bounded pseudo-Boolean functions, and empirically we find that the best of the 2^q reachable solutions is also a local optimum in $f(x)$ more than 80 percent of the time.

Since the proof of Theorem 2.3 is expressed in terms of improving moves, we also automatically get the following result for free.

Theorem 2.4. *Assume that solutions $P1$ and $P2$ are on plateaus such that there is no improving move from solution $P1$ or from $P2$. There are no improving moves from any of the reachable offspring solutions under the function $g(x')$.*

2.2 The Cost of PX compared to Local Search

Modern MAXSAT local search algorithms are able to find Hamming distance 1 improving moves in $O(1)$ time using techniques like gradient-based promising variable selection [15, 25]. This means that PX should only be applied when local search has difficulty finding an improving move, which typically means that search has become stuck on a plateau. PX has the potential to find tunneling moves to an improved solution on a better plateau.

A illustration of how PX is combined with local search appears in Figure 3. When a new plateau is reach, a solution $P1$ is recorded. If an improving move is not found after αn moves, another solution on the plateau, $P2$ is selected. Note there are αn moves separating $P1$ and $P2$, which also implies that local search cannot easily escape from the plateau. $P1$ is then recombined with $P2$.

We can efficiently evaluate recombining components of the recombination graph using the *Score* vector used by all modern local search algorithms to track improving moves (i.e., the *Score* vector tracks moves that "makes" clauses true and "breaks" clauses so that they are not longer true). Assume there are candidate solutions $P1$ and $P2$ which decomposes the VIG into q partitions. Assume the current *Score* vector is defined related to the current solution $P2$. Note that the solution $P1$ and $P2$ have complementary assignments for each subfunction $g_i(x')$. To evaluate the recombining component G_i ($i \in [1, q]$), flip all of the bits in G_i, updating the *Score* vector after each bit flip. The sum of the changes provided by the Score vector is equal to the change in evaluation between $g_i(P2)$ and $g_i(P1)$. While this reduces the runtime, this cost of PX is still $O(n)$.

Algorithm 1 PXSAT: A Generic Framework based on PX

```
1:  x ← rand();                                          ▷ random initialization
2:  x_best ← x; e_best ← f(x); i ← 0; i_best ← 0;
3:  while termination condition not met do
4:      x ← LS(x);                                       ▷ one bit flip by local search
5:      if f(x) < e_best then                            ▷ improvement
6:          x_best ← x; e_best ← f(x); i_best ← i;
7:      else if i > i_best + αn then                     ▷ stagnation
8:          x ← px(x, x_best);
9:          ▷ reset regardless of outcome of PX
10:         x_best ← x; e_best ← f(x); i_best ← i;
11:     i ← i + 1
```

The following theorem addresses the trade-off in applying PX or doing $\theta(n)$ steps of local search.

Theorem 2.5. *Given $\theta(n)$ time, where n is the number of variables, local search with gradient-based variable selection implicitly checks $\theta(n^2)$ candidate solution, while PX implicitly checks $\theta(2^q)$ candidate solutions, where q is the number of components in the recombination graphs.*

Proof. In local search with gradient-based promising variable selection, an improving move among n neighbor solutions can be discovered in $\theta(1)$ time. Given $\theta(n)$ time, $\theta(n)$ such moves can be performed. A total of $\theta(n^2)$ candidate solutions can thus be implicitly checked given $\theta(n)$ explicit evaluations. For PX, only $\theta(1)$ crossovers can be performed in $\theta(n)$ time, so that a total of $\theta(2^q)$ candidate solutions are implicitly checked given $\theta(n)$ explicit evaluations. □

3 PXSAT

We will use our theoretical results to motivate the use of PX for MAXSAT. We are not replacing local search with PX. Instead, we present a general framework that combines PX with local search. We design the framework with simplicity in mind, so that the framework can be incorporated into any existing local search solver. With the introduction of PX, some of computational resources previously spent on local search inevitably needs to be allocated for PX. Using PX offers new opportunities of exploring a very different set of candidate solutions on plateaus that are difficult to escape by local search.

We now present PXSAT framework in Algorithm 1. x_{best} keeps tracks of the best candidate solution within a variable-length interval. At initialization, randomly generate the current solution x. x_{best} is set to x. When local improves and updates x_{best} in less than αn steps, the interval keeps expanding. Otherwise, when there is no improvement over x_{best} for αn steps, a stagnation is detected. In case of stagnation, apply PX to x_{best} and x, reset x_{best} to x, which starts a new interval.

4 EMPIRICAL RESULTS

PXSAT is first used to improve two of best performing local search SAT solvers on application instances, AdaptG^{2}WSAT [17] and Sparrow [3]. This shows *relative performance improvement* achieved by incorporating PXSAT into existing solvers. We then demonstrate

Base Solver	AdaptG^{2}WSAT	Sparrow
#No Diff (No PX)	13	11
#No Diff (PX Ties)	4	2
# Better Δsol (Total)	63	73
# Better Δsol (Sig)	38	40
# Worse Δsol (Total)	8	6
# Worse Δsol (Sig)	2	0
# Faster $\Delta time$ (Total)	11	8
# Slower $\Delta time$ (Total)	3	2

Table 1: Comparing PXSAT versions with the original local search solvers. "#No Diff (No PX)": number of instances where PX is never triggered. "#No Diff (PX Ties)": number of instances where PX is triggered and average solving time and average solution quality are tied between the original solver and its PXSAT version. "# Better (Worse) Δsol": number of instances where the PXSAT version has better (worse) average solution quality. "# Faster (Slower) $\Delta time$": number of instances where the PXSAT version has faster (slower) average solving time.

that the improvement achieved by PXSAT is significant by comparing AdaptG^{2}WSAT-PX and Sparrow-PX with CCLS [18], a state-of-the-art local search solver designed specifically for MAXSAT. This exhibits the *absolute performance* of PXSAT. Empirical results show that the performance of a 10-year-old SAT solver AdaptG^{2}WSAT can be lifted by PXSAT so that it even outperforms the state-of-the-art MAXSAT solver on every instance tested. We also establish a theoretical model for predicting and the success of PXSAT.

Setup: The benchmark set is constructed as follows. From 150 satisfiable instances from the *crafted track* and 150 satisfiable instances from the *industrial track* in the SAT competition 2014, sample three instances (smallest/median/largest in size in terms of number of variables) from each class if there are more than three, otherwise select all instances. We only selected the satisfiable instances so that the optimum is known to have a zero evaluation. We excluded nine extremely large instances due the memory limits of the machines available for these experiments. In all, we used 102 instances; these instances were preprocessed offline by SatELite [10]. We also evaluated PXSAT on crafted instances and industrial instances from MAXSAT Evaluation 2016[2]. However, these instances were not challenging enough; local search is often able to find an improving solution within αn iterations. As long as local search continues to find improving moves, Partition Crossover is not triggered.

The parameter α in PXSAT was fixed to be 1, 2, 4, 8 and 16 on each instance, and the best performance among the five settings are used for comparison with the original SAT solvers. The parameter α was tuned by running some preliminary examples of recombination for that instance. In approximately half of the cases (48), $\alpha = 1$ was best.

4.1 Improving State-of-the-Art Local Search SAT Solvers

We combine PXSAT with AdaptG^{2}WSAT [17] and Sparrow [3]. AdaptG^{2}WSAT has been found to be one of the best performing local search solvers on application instances [14]. Sparrow, on the other hand, performed the best among all local search solvers in both crafted SAT track and application SAT track in SAT Competition [3] 2014. Both solvers are available from the UBCSAT [4] collection [24]. The average of the best solutions found over 10 trials, each of 5000 seconds, was compared.

Table 1 summarizes the impact of PXSAT when incorporated into AdaptG^{2}WSAT and Sparrow. There are 11 instances for Sparrow and 13 instances for AdaptG^{2}WSAT that are easy enough to solve without any stagnation of over αn iterations. Thus in these case, PX is not triggered. On the remaining instances, PXSAT has a *strong positive influence* on both average solution quality and average solving time. Mann-Whitney test [19] is employed to test whether the differences in averages are statistically significant, assuming a significance level of 0.05. We use *percentage improvement* to measure the improvements. It is defined as

$$\Delta = (orig - pxsat)/orig \times 100\%, \tag{1}$$

where $orig$ is the metric (Δsol denotes solution quality and $\Delta time$ denotes solving time) for the base solver and $pxsat$ is for its PXSAT version.

PXSAT improves AdaptG^{2}WSAT on 63 instances in terms of average solution quality; it is worse on only 8 instances. The percentage improvement in average solution quality is 22.4%. The improvements in solution quality achieved by PXSAT on 38 instances are statistically significant. AdaptG^{2}WSAT-PX is only significantly worse on two instances. Considering only the instances where the differences are statistically significant, the percentage improvement in solution quality boosts to 27.2%. This is a substantial improvement considering how difficult it is for AdaptG^{2}WSAT to find a better solution. On average, AdaptG^{2}WSAT stagnates through the last 54.9% of total solving time and local search is unproductive. On the instance `5-SATISFIABLE`, AdaptG^{2}WSAT fails to find any improving solution in the last 4965 (= 99.3% × 5000) seconds. PXSAT also accelerates AdaptG^{2}WSAT on 11 instances, and slows down AdaptG^{2}WSAT on 3 instances, with an average time reduction of 25.6%. However, due to the stochastic nature of local search where fluctuation in solving time is high, there are only a couple of instances whose solving time differences are statistically significant.

For Sparrow, PXSAT improves on 73 instances in terms of average solution quality, out of which 40 are statistically significant. There is not a single instance where adding PXSAT results in poorer average solution quality. The average percentage improvement in solution quality over all instances with statistically significant differences is 26.4%. Despite an average improvement of 25.1% on 10 instances where the solving times are different, none of the differences is statistically significant.

Interestingly, for both AdaptG^{2}WSAT and Sparrow, there are 40 instances where the differences in solution quality are statistically

[2] http://maxsat.ia.udl.cat
[3] http://www.satcompetition.org/2014/
[4] https://github.com/dtompkins/ubcsat/releases/tag/v1.2beta18

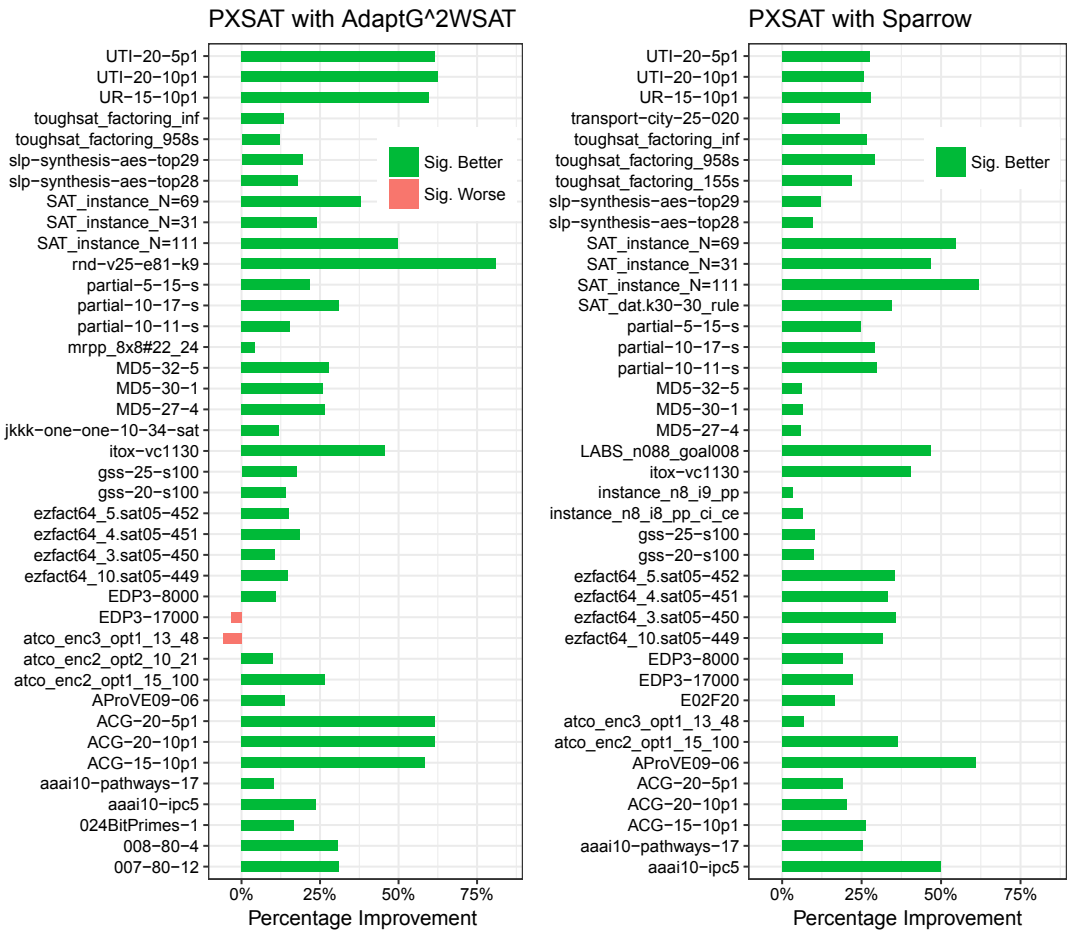

Figure 4: Comparing PXSAT versions with the original local search solvers on instances where the average solution quality differences are statistically significant.

significant. Figure 4 details the differences on the these instances. When PXSAT makes a significant impact on the solution quality, it is almost always a positive impact across the board, except on only two instances. Moreover, the decrease in solution is rather small (3% and 5% respectively), compared with the gain in all the other instances. There are 33 overlapping instances between the two sets. This suggests that PXSAT is particularly useful for a specific set of instances. We next study the common properties they share and how the properties benefit PXSAT.

4.2 Why and When PXSAT works?

The benefit of applying PX τ times can be quantified in terms of number of inspected candidate solutions as

$$r = \frac{\sum_{i=1}^{\tau} 2^{q_i}}{\tau \times n^2}, \tag{2}$$

Because $\sum_{i=1}^{\tau} 2^{q_i}$ can be prohibitively large, we instead record the average number of components $\overline{p} = \sum_{i=1}^{\tau} q_i/\tau$, and use $\overline{p}$ to derive a lower bound on r.

Theorem 4.1. *Let* $\check{r} = 2^{\overline{p}}/n^2$, *then* $\check{r} \leq r$.

Proof. $\check{r} \leq r$ is equivalent to

$$2^{\frac{1}{\tau}\sum_{i=1}^{\tau} q_i} \leq \frac{1}{\tau}\sum_{i=1}^{\tau} 2^{q_i}. \tag{3}$$

Jensen's inequality [13] states, if X is a random variable and φ is a convex function,

$$\varphi(\mathbb{E}[X]) \leq \mathbb{E}[\varphi(X)] \tag{4}$$

where the equality holds if and only if $x_1 = x_2 = ... = x_n$ or φ is linear. Let $\varphi(q) = 2^q$. Since an exponential function is a convex function, we have

$$2^{\frac{1}{\tau}\sum_{i=1}^{\tau} q_i} = \varphi(\mathbb{E}[q]) \leq \mathbb{E}[\varphi(q)] = \frac{1}{\tau}\sum_{i=1}^{\tau} 2^{q_i}.$$

□

In practice, we also apply a log10 transformation to $\check{r}$, and define the result as $\check{lr}$.

$$\check{lr} = log10(\check{r}) = \overline{p} \times log10(2) - log10(n^2). \tag{5}$$

Based on the empirical results and the number of components collected for Sparrow, we perform a correlation analysis between

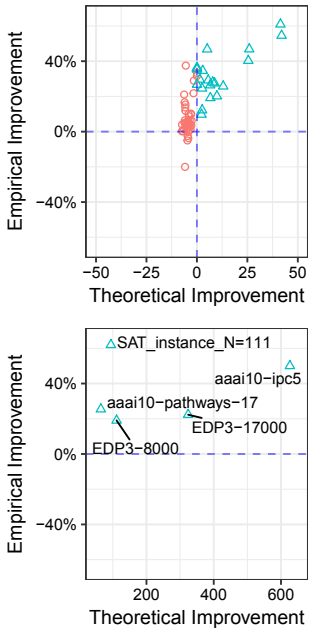

Figure 5: Empirical $\overline{\Delta sol}$ versus theoretically approximated $\check{lr}$ for Sparrow. Left subfigure has a scale of [-50,50] on X-axis, while right subfigure has a scale of [50,650] on X-axis. Points with $\overline{\Delta sol} > 0$ ($\overline{\Delta sol} < 0$) are colored blue (red).

$\overline{\Delta sol}$ and $\check{lr}$. Figure 5 presents the outcome. The figure is split into two subfigures with different scales on X-axis, because $\check{lr}$ is still very large on some instances. For example, on aaai10-ipc5 $\check{lr} = 626$, the number of candidate solutions filtered by PX is at least $\check{r} = 10^{626}$ times larger than the number are filtered by Sparrow. This is a result of the instance being decomposed into *over 2117 components* in one application of PX.

There is a clear positive correlation between $\overline{\Delta sol}$ and $\check{lr}$, which is confirmed by Spearman correlation of 0.656[5]. Given an instance, increasing the number of component is indeed critical for the performance of PXSAT. On the 27 instances where $\check{lr} > 0$, PXSAT always improves $\overline{\Delta sol}$.

Interestingly, there are 35 instances where $\check{lr} < 0$ (which suggests PXSAT inspects less candidate solutions) and yet $\overline{\Delta sol} > 0$ (empirical results show PXSAT improves the performance). Two (non-exclusive) reasons exist. First, q_i across multiple applications of PX is probably very different, thus $\check{r}$ underestimates r, which results into a smaller $\check{lr}$. Second, PXSAT is exploring a neighborhood that is drastically different from the one-bit neighborhood in local search solvers. Using PXSAT is beneficial because PX can find improving moves when local search cannot, even if PXSAT checks fewer candidate solutions.

Table 2 presents summary statistics on the number of components and success rate of PX applications. Despite a small median number of components ($\approx$ 16), PXSAT manages to achieve a median success rate of 50.2% on PX applications. This is impressive because

[5]A Spearman correlation of 1 results when the two variables are monotonically related.

	Min	Median	Mean	Max
q	2.000	16.707	129.482	2897.361
Success Rate	0.01%	50.2%	43.2%	96.6%

Table 2: Summary statistics on number of components (q) and PX Success Rate at finding improving moves for Sparrow-PX.

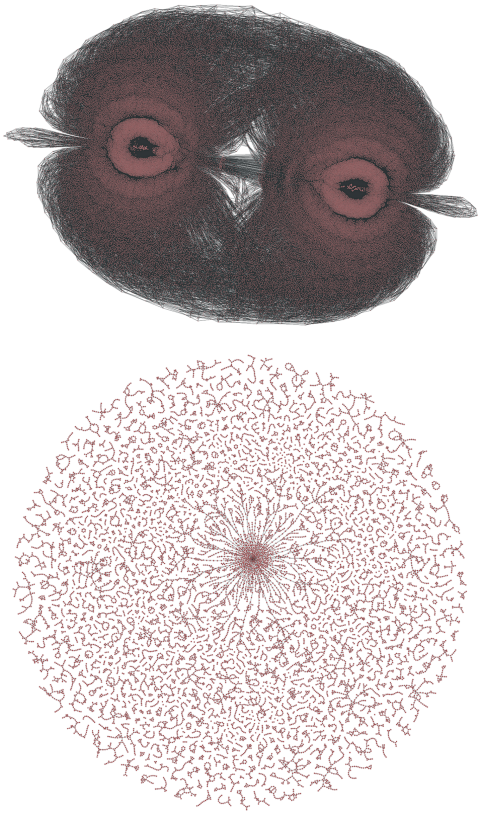

Figure 6: The VIG (top) and the decomposed recombination graph (bottom) for SAT_instance_N=111. In this instance, tunneling will return the best of 2^{842} solutions.

PX is only triggered on plateaus that are difficult to escape for local search.

Figure 6 visualizes the VIG and the decomposed recombination graph of the instance where Sparrow-PX has the largest performance gain. The original graph appears axisymmetric with two densely connected cores on each side and the connections between the two cores are sparser. PX successfully breaks one of the cores, leading to 842 components. One application of PX yields an instant improvement of 316 additional satisfied clauses, while Sparrow fails to discover any improving move in the last 72001 bit flips.

Overall, $\check{lr}$ is a very conservative predictor for the success of PXSAT. When $\check{lr} > 0$, PXSAT always improves the performance in the empirical study. When $\check{lr} < 0$, there is still a good chance of making a positive impact. PXSAT improves 35 out of 41 instances with $\check{lr} < 0$.

4.3 Improving on a State-of-the-Art MAXSAT Solver

CCLS [18] is one of the best custom designed solvers for MAXSAT, and has won several categories of the incomplete algorithms track of MaxSAT Evaluation 2013 to 2016. Because the CCLS source code is not publicly available, we used the binary [6] files.

CCLS was evaluated on the same benchmark for comparison, in order to access the *absolute performance* of PXSAT-equipped local search solvers. CCLS has better average solution quality than the original AdaptG^{2}WSAT and Sparrow on 7 instances and 27 instances, respectively. Thanks to PXSAT, AdaptG^{2}WSAT-PX consistently outperforms CCLS on every single instance tested, while Sparrow-PX takes the lead on five additional instances.

5 CONCLUSIONS

PXSAT employs a powerful recombination operators, Partition Crossover (PX), to exploit decomposability on application instances and to escape plateaus. PX uses common assignments among local optima to decompose Variable Interaction Graphs (VIGs) into q components. The best solution of 2^q candidate solutions can be greedily constructed in linear time. Empirical studies on an extensive set of application instances show PXSAT statistically significantly improves the performance of two best local search solvers, AdaptG^{2}WSAT and Sparrow, on application instances. The improvement in solution quality is as much as 80%.

We present theoretical analysis for highlighting the search efficiency of PXSAT as well as guiding the design of PXSAT. A theoretical performance model is developed to understand why and when PXSAT is useful. The model successfully predicts when PXSAT is likely to improve the performance. The model shows, given the same amount of time, the number of candidate solutions inspected by PXSAT is up to 10^{626} times more than the greedy operators in modern local search solvers.

This work opens up many interesting future directions that requires further investigation. PX takes linear time to decompose the VIGs, as it is currently employing a top-down approach for decomposition. As an alternative, we can construct the decomposition in a bottom-up manner by tracking how the second parent deviates from the first parent, and incrementally add vertices to the recombination graph. This could potentially result in a reduction of the time complexity of Partition Crossover applied to MAXSAT.

REFERENCES

[1] Abramé, A., and Habet, D. 2015. Ahmaxsat: Description and evaluation of a branch and bound max-sat solver. *Journal on Satisfiability, Boolean Modeling and Computation* 9:89–128.
[2] Ansótegui, C.; Giráldez-Cru, J.; Levy, J.; and Simon, L. 2015. Using community structure to detect relevant learnt clauses. In *International Conference on Theory and Applications of Satisfiability Testing*, 238–254. Springer.
[3] Balint, A., and Fröhlich, A. 2010. Improving stochastic local search for SAT with a new probability distribution. In *Proc. of SAT*, 10–15. Springer.
[4] Berg, J.; JÃđrvisalo, M.; and Malone, B. 2014. Learning Optimal Bounded Treewidth Bayesian Networks via Maximum Satisfiability. In Kaski, S., and Corander, J., eds., *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33 of *Proceedings of Machine Learning Research*, 86–95. Reykjavik, Iceland: PMLR.
[5] Biere, A., and Sinz, C. 2006. Decomposing SAT problems into connected components. *JSAT* 2(1-4):201–208.
[6] Bunte, K.; JÃđrvisalo, M.; Berg, J.; MyllymÃđki, P.; Peltonen, J.; and Kaski, S. 2014. Optimal neighborhood preserving visualization by maximum satisfiability. In *AAAI Conference on Artificial Intelligence*.
[7] Chen, Y.; Safarpour, S.; Marques-Silva, J. a.; and Veneris, A. G. 2010. Automated design debugging with maximum satisfiability. *IEEE Trans. on CAD of Integrated Circuits and Systems* 29(11):1804–1817.
[8] Chicano, F.; Whitley, D.; Ochoa, G.; and Tinós, R. 2017. Optimizing one million variable nk landscapes by hybridizing deterministic recombination and local search. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, 753–760. New York, NY, USA: ACM.
[9] Cook, S. A. 1971. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, 151–158. ACM.
[10] Eén, N., and Biere, A. 2005. Effective Preprocessing in SAT Through Variable and Clause Elimination. In *Proc. of SAT*, 61–75. Springer.
[11] Frank, J. D.; Cheeseman, P.; and Stutz, J. 1997. When gravity fails: Local search topology. *Journal of Artificial Intelligence Research* 7:249–281.
[12] Huang, J., and Darwiche, A. 2003. A structure-based variable ordering heuristic for SAT. In *IJCAI*, volume 3, 1167–1172.
[13] Jensen, J. 1906. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta Mathematica* 30(1):175–193.
[14] Kroc, L.; Sabharwal, A.; Gomes, C. P.; and Selman, B. 2009. Integrating systematic and local search paradigms: a new strategy for MaxSAT. In *Proc. of IJCAI*, 544–551. Morgan Kaufmann Publishers Inc.
[15] Li, C. M., and Huang, W. Q. 2005. Diversification and determinism in local search for satisfiability. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing*, SAT'05, 158–172. Berlin, Heidelberg: Springer-Verlag.
[16] Li, C.-M., and Quan, Z. 2010. An efficient branch-and-bound algorithm based on MAXSAT for the maximum clique problem. In *AAAI Conference on Artificial Intelligence*.
[17] Li, C. M.; Wei, W.; and Zhang, H. 2007. Combining adaptive noise and look-ahead in local search for SAT. In *Proc. of SAT*, 121–133. Springer.
[18] Luo, C.; Cai, S.; Wu, W.; Jie, Z.; and Su, K. 2015. CCLS: an efficient local search algorithm for weighted maximum satisfiability. *IEEE Trans. Computers* 64(7):1830–1843.
[19] Mann, H. B., and Whitney, D. R. 1947. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics* 18(1):50–60.
[20] Monasson, R.; Zecchina, R.; Kirkpatrick, S.; Selman, B.; and Troyansky, L. 1999. Determining computational complexity from characteristic 'phase transitions'. *Nature* 400(6740):133–137.
[21] Qasem, M., and Prugel-Bennett, A. 2010. Learning the large-scale structure of the max-sat landscape using populations. *IEEE Transactions on Evolutionary Computation* 14(4):518–529.
[22] Selman, B., and Kautz, H. A. 1993. An empirical study of greedy local search for satisfiability testing. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, AAAI'93, 46–51. AAAI Press.
[23] Tinós, R.; Whitley, D.; and Chicano, F. 2015. Partition crossover for pseudo-boolean optimization. In *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII*, 137–149. ACM.
[24] Tompkins, D. A. D., and Hoos, H. H. 2005. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In *Revised Selected Papers from the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT 2004)*, 306–320.
[25] Whitley, D.; Howe, A. E.; and Hains, D. 2013. Greedy or not? best improving versus first improving stochastic local search for maxsat. In desJardins, M., and Littman, M. L., eds., *AAAI*. AAAI Press.
[26] Zhang, W. 2004. Configuration landscape analysis and backbone guided local search: part i: Satisfiability and maximum satisfiability. *Artificial Intelligence* 158:1–26.
[27] Whitley, D., Hains, D., and Howe, A. (2009). Tunneling between optima: partition crossover for the TSP. In *Proc. of GECCO'2009*, pages 915–922.
[28] Whitley, D., Hains, D., and Howe, A. (2010). A hybrid genetic algorithm for the traveling salesman problem using generalized partition crossover. In *Proc. of PPSN XI*, pages 566–575. Springer.
[29] Sanchez, D., Whitley, D., and Tinós, R. (2017). Building a better heuristic for the traveling salesman problem: Combining edge assembly crossover and partition crossover. In *Genetic and Evolutionary Computation Conference (GECCO)*, pages 329–336, New York, NY, USA. ACM.
[30] Helsgaun, K. (2012b). LKH version 2.0.7. http://www.akira.ruc.dk/keld/research/LKH/. Updated in November 12, 2012.
[31] W. Chen and D. Whitley 2017 Decomposing SAT Instances with Pseudo Backbones. In *European Conference on Evolutionary Computation in Combinatorial Optimization (EVOCOP)*, Springer. pages 75-90.

[6] http://lcs.ios.ac.cn/~caisw/Code/CCLS2015