Inference Rules in Local Search for Max-SAT

André Abramé and Djamal Habet
Aix Marseille University, LSIS UMR 7296
Av. Escadrille Normandie Niemen
13397 Marseille Cedex 20 (France)
{andre.abrame, djamal.habet}@lsis.org

Abstract—In the last years, many advances were accomplished in the exact solving of the Max-SAT problem, especially by the definition of new inference rules and a better estimation of lower bounds in branch and bound based methods. However, and oppositely to the SAT problem, fewer works exist on approximate methods for Max-SAT, mainly local search ones which have shown their potency for SAT. In this paper, we illustrate that including inference rules in a classical local search solver for SAT improves its performances when solving the Max-SAT problem. The obtained results confirm the efficiency of our approach.

Keywords—Max-SAT; Local Search; Inference Rules; Max-resolution.

I. INTRODUCTION

The Maximum Satisfiability (Max-SAT) problem is to find an assignment of Boolean values to the variables of a propositional formula expressed in a Conjunctive Normal Form (CNF) that maximizes the number of satisfied clauses. This NP-hard problem [1] is the optimization version of the SAT decision problem. During the last years, the interest in studying Max-SAT has grown significantly because of its capacity to express and to solve real-life problems in various domains such as routing, scheduling, bio-informatics and also academic problems (Max-Cut, Max-Clique, etc).

There are two main approaches to solve Max-SAT. The first one is exact and generally based on the Branch & Bound (BnB) algorithm. Many competitive implementations exist [2], [3], [4] which differ mainly on their variable selection heuristic, their lower bound estimation and the inference rules they use. The second approach concerns approximation algorithms which do not guarantee the optimality of the solutions but return a result of good quality in a reasonable time. Such algorithms include semi-definite programming, pseudo Boolean optimization, etc [5], [6], [7]. Local search (LS) methods, efficient when dealing with SAT, belong also to these approximation methods. Any LS algorithm for SAT can be used to solve the unweighted Max-SAT problem with minor modifications. However, specific adaptations have been made on LS to be applied for Max-SAT. The first one is owed to Hansen and Jaumard [8]. Among the most recent works, Zang et al. [9] use backbones to improve a classical LS algorithms, Smyth et al. [10] adapt tabu search to Max-SAT, Lardeux et al. combine complete and incomplete search in [11] by using a three values scheme. More recently, Kroc et al. [12] use a classical DPLL SAT solver to guide a local search solver running in parallel. To the best of our knowledge, there is no more recent work on LS adaptation for Max-SAT. A general presentation of the Max-SAT problem is made in [13] and two states of the art of LS for Max-SAT exist, one complete but outdated [14] and a more up to date one [15].

Inference rules have shown their interest in BnB algorithms, for example to simplify the formula or to improve the estimation of lower bounds. However, such rules have rarely been exploited in a local search algorithm to make a Max-SAT instance easier to solve. To the best of our knowledge, except Heras and Baneres in [16] which use such rules in a preprocessing step before applying classical LS algorithms, there is no work dealing with this point. In this paper, we use an inference rule combining unit propagation and Max-resolution for deducting some of the Minimal Unsatisfiable Subsets (MUSes) (for a formal definition of MUS and its relationship with Max-SAT, see for instance [5]) of the instances. We integrate this rule in an existing LS algorithm for SAT (Novelty++ with adaptive noise setting [17], [18]). The resulting algorithm, IRAnovelty++, improves significantly the performances of the LS search, particularly when solving Max-2-SAT instances. One of the difficulties to overcome in order to make this rule applicable in a LS algorithm is the definition of refutation trees in the LS context. This issue has already been studied for the SAT context, for example in [19], [20], [21], [22], [23]. Moreover, a balance must be struck between the time spent on applying the rule and the time consumed in solving the instance. Otherwise, the resulting algorithm may be inefficient in practice.

This paper is organized as follows. In Section II, we give some basic definitions and notations used in this paper. The inference rules and the local search algorithm used in our implementation are introduced in Sections III and IV. We detail in Section V our implementation. Eventually, the experiments and the results obtained are presented in Section VI and we conclude in Section VII.

II. NOTATIONS

Let $X = \{x_1, \ldots, x_n\}$ be a set of propositional variables. A literal $l$ is a variable $x_i$ or its negation $\neg x_i$. A clause is a...
finite disjunction of literals and a formula $\Sigma$ in conjunctive normal form (CNF) is a conjunction of clauses. Alternatively, clauses can be represented as sets of literals and a formula as a multiset of clauses.

An assignment $I$ is a set of literals which does not contain both a variable and its negation. If $|I| = n$ then $I$ is complete and it is partial otherwise. A variable $x_i$ such that neither $x_i$ nor $\neg x_i$ belongs to $I$ is unassigned. For a given literal $l$, we denote $\Sigma|_l$ the formula obtained by applying $l$ on $\Sigma$. Formally, $\Sigma|_l = \{ c | c \in \Sigma, \{l, \neg l\} \cap c = \emptyset \} \cup \{c/\{l\} | c \in \Sigma, \neg l \in c \}$. This application can be extended to any assignment $I = \{l_1, l_2, \ldots, l_k\}$, such that $\Sigma|_I = (\ldots((\Sigma|_{l_1})|_{l_2})\ldots)|_{l_k}$.

A literal $l$ is satisfied by an assignment $I$ iff $l \in I$ and it is falsified iff $\neg l \in I$. A clause is satisfied iff at least one of its literals is satisfied, and it is falsified (or conflicting) iff all its literals are falsified. By convention the empty clause, noted $\Box$, is conflicting. The Max-SAT problem consists of finding an assignment that maximizes (minimizes) the number of satisfied (falsified) clauses.

III. INFECTION RULES

In the recent years, research on Max-SAT was focused on the exact solving methods, based on BnB, and particularly on one of their main components: the inference rules which transform a formula $\Sigma$ into an equivalent one $\Sigma'$. We denote this transformation by $\Sigma \Rightarrow \Sigma'$.

In the case of the SAT problem, the inference rules must preserve the satisfiability of the instance. However, when applied to Max-SAT, they must also preserve the number of unsatisfied clauses for every assignment. They can have several purposes, such as improving the applicability of the other solver components or memorizing some deductions to avoid redundancy. In this section, we recall the powerful inference rule for Max-SAT defined in [3] which uses the Max-SAT adaptation of two well known inference rules (unit propagation and resolution) for SAT.

A clause containing only one literal is a unit clause. In presence of such a clause, all the clauses containing the literal and all the occurrences of its negation can be removed. The iterative application of this rule until no more unit clause remains is called unit propagation. We denote $\Sigma^*$ the formula obtained by applying unit propagation to a formula $\Sigma$. This inference rule preserves the satisfiability, but can lead to a non-optimal solution in number of satisfied clauses and is therefore inapplicable to Max-SAT. However, recent complete Max-SAT solvers perform a simulated unit propagation (i.e. the consequences are not applied on the formula) to detect conflicts and improve their lower bound estimation.

Another useful inference rule for SAT is resolution [24]. If two clauses $\{x, A\}$ and $\{-x, B\}$ are present in a formula (with $A$ and $B$ disjunction of literals), then the clause $\{A, B\}$ can be added without affecting its satisfiability.

Again, this rule does not preserve equivalency for Max-SAT. A variant for Max-SAT, named Max-resolution, has been proposed in [11] (and later studied in [25, 26]). It can be defined as follows:

$$\Sigma = \{x, A\} \cup \{-x, B\} \cup \Sigma'$$

where $\Sigma'$ is a multiset of clauses. Two of the newly produced clauses are not disjunctions of literals and need to be transformed to preserve the CNF form of the resulting formula. It should be noted that this transformation produces $(|A| + (|B| - 1))$ clauses of sizes varying from $\min(|A|, |B|) + 1$ to $|A| + |B| - 1$.

In [3], these inference rules are applied to improve the lower bound estimation made by their BnB solver MiniMaxSat. The main idea is to use the Max-resolution to perform deductions on the formulas as it is done by the clause learning mechanism of SAT solvers: when a conflict is detected by unit propagation, a succession of resolution steps are applied on the clauses which have led to the conflict, producing a learnt clause. MiniMaxSat performs, at each decision, a simulation of unit propagation. When a conflict is derived, the set of clauses which has led to
it forms an unsatisfiable subset of the formula. From this subset, the empty clause \( \square \) can be derived by several Max-resolution steps made in a given order. These resolution steps make up the refutation tree and can be easily deduced from the implication graph (or from a simple propagation queue). Contrary to the SAT learning mechanism, the set of clauses which have led to the conflict is removed from the formula and compensation clauses are added to preserve the number of satisfied and falsified clauses for each possible assignment. To limit the number of clauses added to the formula, MiniMaxSat applies this method only when all the clauses of the unsatisfiable subset contain less than five literals. Fig. 1 gives an example of the application of the presented inference rules. It should be noted that MiniMaxSat applies this inference rule at each node of the search tree, but the resulting transformation is only kept in the sub-part of the search tree. The transformations are undone when the algorithm backtracks.

IV. NOVELTY++ ALGORITHM

This section provides a brief overview of local search algorithms for SAT, with a focus on Novelty++ [17] which is used in our approach.

Algorithm 1 presents a general scheme of a local search (LS) algorithm based on WalkSat (originally introduced in [27]). It starts by randomly choosing a complete assignment \( I \) (function random_complete_assignment, line 4) and iteratively selects a variable to flip (lines 7-9) in order to improve the number of satisfied clauses, until either a satisfying assignment is found (lines 5-6) or a given maximum number of flips (FreqRestarts) is reached. This process is repeated up to a maximum number of MaxRestarts restarts. This restart mechanism can be viewed as a diversification mechanism.

WalkSat variants differ on their heuristic to select the variable to flip [28]. Such heuristic is based on two counters: break and make which calculate, for a given variable \( x_i \), the number of clauses in \( \Sigma \) which will be falsified and satisfied respectively if \( x_i \) is flipped. Let \( \text{score}(x_i) \) be the difference between \( \text{make}(x_i) \) and \( \text{break}(x_i) \) \( \text{score}(x_i) = \text{make}(x_i) - \text{break}(x_i) \). WalkSat family provides several heuristics to pick a variable to flip (function select_var_to_flip) from a randomly selected unsatisfied clause \( c \), returned by the function select_falsified_clause. In the original WalkSat implementation, if there are variables in \( c \) with a null break value, randomly pick one of them, otherwise with a probability \( p \), randomly pick a variable from \( c \) (random walk) and with probability \( 1 - p \), randomly pick one of the variables with the smallest break value.

In Novelty [28], the variables appearing in \( c \) are sorted by their score, breaking ties in favor of least recently flipped variables. Let consider the best and the second best variables. If the best variable is not the most recently flipped one in \( c \) then it is selected. Else, with a probability \( p \), the algorithm selects the second best one and with the probability \( 1 - p \) it picks the best variable.

In [17], Novelty is extended to Novelty++ as follows: with probability \( dp \) (diversification probability), the algorithm picks the least recently flipped variable in \( c \) (corresponding to a diversification) and with probability \( 1 - dp \), it does as Novelty.

The optimal values for \( p \) and \( dp \) depend on the solved instance and are hard to determine. In [18], it is proposed a dynamic scheme to adjust the values of such probabilities depending on the evolution of the search: if the number of falsified clauses has not been reduced in a given number of flips then \( p \) and \( dp \) are increased to favor the diversification. Oppositely, when the number of falsified clauses is reduced \( p \) and \( dp \) are decreased. Novelty++ with adaptive probability setting will be referred as AdaptNovelty++.

V. IRANovelty++: LS AND INFERENCE RULES FOR MAX-SAT

Our main purpose is to integrate inference rules in a local search algorithm in order to improve the solving of the Max-SAT problem. Hence, we add to AdaptNovelty++ the necessary elements to apply the inference rule described in Section III to form an original Max-SAT solver, IRANovelty++.

The first issue to overcome is the integration of simulated unit propagation (necessary to build refutation trees, and thus to apply the Max-resolution based inference rule) in AdaptNovelty++. As local search algorithms work on complete assignments, while simulated unit propagation acts on the basis of partial ones, our solver keeps simultaneously two assignments: a complete one for the local search (named \( I_\alpha \)) and a partial one for the detection of conflicts by unit propagation (\( I_{up} \)). The filling of \( I_{up} \) is done with the achieved flips according to the AdaptNovelty++ heuristic. It should be noted that keeping the coherency of the implication graph is much more difficult in a LS process than in a complete

---

### Algorithm 1: WalkSat

**Data:** \( \Sigma, \text{MaxRestarts}, \text{FreqRestart} \)

**Result:** If a satisfiable assignment \( I \) is found then \( I \), else \( \emptyset \).

1. **begin**
2. \[ \text{for } i = 0 \text{ to MaxRestarts do} \]
3. \[ \quad \text{for } j = 0 \text{ to FreqRestart do} \]
4. \[ \quad I = \text{random_complete_assignment}(); \]
5. \[ \quad \text{if } \text{is_satisfied}(\Sigma|I) \text{ then} \]
6. \[ \quad \quad \text{return } I; \]
7. \[ \quad c = \text{select_falsified_clause}(\Sigma|I); \]
8. \[ \quad x_i = \text{select_var_to_flip}(c); \]
9. \[ \quad I = \text{flip} (I, x_i); \]
10. **return** \( \emptyset \)
11. **end**
solver. Indeed, an opposite value can be given to a variable already assigned by propagation and thus all its assignment sources via propagation must be undone.

To apply the inference rule in IRAnovelty++, the second step is the building of refutation trees when conflicts are discovered by unit propagation (i.e. when there exists a clause \( c \) such as for all literal \( l \in c, \neg l \in I_{up} \)). This point does not require much explanation, as it is similarly done and largely documented in the complete SAT solvers context.

Eventually, the Max-resolution based inference rule is applied according to the refutation tree, as described in Section III. In MiniMaxSat, the formula transformations made by the Max-resolution rule are kept only in the sub-part of the search tree where they have been made. Thus, the formula transformations allow a better estimation of the lower bound and act as a (partial) memorization scheme. It should be noted that it is not the case in our approach. IRAnovelty++ keeps the transformations even after the assignment which has led to them has been undone. The main interest of this use of the Max-resolution rule is that it dynamically builds Minimal Unsatisfiable Subsets (MUS) and thus it simplifies the formula by making some conflicts obvious.

We define two parameters to control the amount of transformations made and to limit the size of the resulting formula: \( \text{LimUP} \) and \( \text{LimRES} \).

- \( \text{LimUP} \) defines the minimum percentage of the original clauses which must remain in the transformed formula. Obviously, with \( \text{LimUP} = 1 \) no transformations are allowed while with \( \text{LimUP} = 0 \) the formula is completely transformed.

- \( \text{LimRES} \) defines the maximum size of the clauses allowed in the refutation tree. It aims to spur the deduction of small resolvents, and thus of MUS, and it limits the size and the number of the added clauses.

Again, with values less than the size of the clauses of the original formula no transformations are allowed and the algorithm acts as a simple local search solver.

Moreover, IRAnovelty++ includes a new diversification mechanism which erases the current rewriting and restores the original formula after \( \text{FreqRewriting}[^\Sigma] \) flips. This mechanism may reduce the impact of the quality of the transformation on the efficiency of IRAnovelty++. It also includes the classical diversification mechanisms presented in Section IV the local search is restarted by a new random assignment after \( \text{FreqRestart}[^\Sigma] \) flips and the adaptive mechanisms control the greediness of the algorithm.

IRAnovelty++ is detailed in Algorithm 2 where the following notations are used: \( \text{best} \) and \( I_{best} \) are respectively the minimum number of falsified clauses found so far and the corresponding assignment. \( \text{MaxRewritings} \) is the number of the formula transformations which must be made by the algorithm. Also, the following functions are involved: \( \text{assign}(I, l) \) adds the literal \( l \) to the assignment \( I \) and solves the possible incoherence in the implication graph. \( \text{original_clause_rate}(\Sigma, \Sigma') \) calculates the percentage of clauses of \( \Sigma \) which appear in \( \Sigma' \) and \( \text{refutation_tree}(I, c) \) returns the refutation tree of the empty clauses \( c \) built from the assignment \( I \). Eventually, \( \text{inference_rule}(\Sigma, R) \) returns the formula obtained by applying the inference rule on \( \Sigma \) with respect to the refutation tree \( R \).

### Algorithm 2: IRAnovelty++

**Data:** \( \Sigma, \text{MaxRewritings}, \text{FreqRewriting}, \text{FreqRestart}, \text{LimRES} \) and \( \text{LimUP} \).

**Result:** \((b, I)\); \( b \) is the minimum number of falsified clauses found during the search and \( I \) the corresponding assignment.

```plaintext
1 begin
2 \( I_{best} = 0; \text{best} = |\Sigma|; \)
3 for \( i = 0 \) to \( \text{MaxRewritings} \) do
4 \( \Sigma' = \Sigma; I_{up} = 0; \)
5 for \( j = 0 \) to \( \text{FreqRewriting} \) do
6 \( I_{ls} = \text{random_complet_assignment}(); \)
7 for \( k = 0 \) to \( \text{FreqRestart} \) do
8 if \( \text{is_satisfied}(\Sigma[I_{ls}]) \) then
9 \( \text{return} (0, I_{ls}); \)
10 \( c_{j} = \text{select_falsified_clause}(\Sigma[I_{ls}]); \)
11 \( x_{i} = \text{select_var_to_flip}(c_{j}); \)
12 \( I_{ls} = \text{flip}(I_{ls}, x_{i}); \)
13 \( I_{up} = \text{assign}(I_{up}, l); \)
14 // \( l \) is the literal corresponding to \( x_{i} \) modulo \( \text{limup} \)
15 // its value in \( I_{ls} \)
16 if \( \text{original_clause_rate}(\Sigma, \Sigma') > \text{LimUP} \) then
17 for all empty clauses \( c \in \Sigma'|I_{up} \) do
18 \( R = \text{refutation_tree}(I_{up}, c); \)
19 if \( |c| \leq \text{L_RES} \) then
20 \( \Sigma' = \text{inference_rule}(\Sigma', R); \)
21 if \( |\{c \in \Sigma[I_{ls}]\}| < \text{best} \) then
22 \( \text{best} = |\{c \in \Sigma[I_{ls}]\}|; \)
23 \( I_{best} = I_{ls}; \)
24 \text{return} (\text{best}, I_{best})
25 end
```

IRAnovelty++ starts by filling \( I_{ls} \) with a random complete assignment (line 6). For \( \text{FreqRestart} \) steps, if \( I_{ls} \) does not satisfy the input formula (lines 8-9) then the algorithm selects and flips a variable (lines 10-12) and adds the corresponding assignment to \( I_{up} \) (line 13). If \( \text{LimUP} \) has not been reached, the algorithm applies unit propagation on \( \Sigma'|I_{up} \) (line 17) and for each empty clause found it builds the refutation tree (line 18) and applies the inference rule (lines 19-20). \( \text{best} \) and \( I_{best} \) are updated (lines 21-23) if the current number of falsified clauses is smaller than the

\(^1\)As stated in [5], a solution of a Max-SAT instance is a covering set of the set of the MUSes of the instance.
best known found so far. Eventually, in the loop of line 5, IRAnovelty++ performs a restart, while in the loop of the line 3 it achieves a rewriting by emptying $I_{up}$ and restoring the original formula.

VI. EXPERIMENTAL RESULTS

As it was described above, IRAnovelty++ contains several parameters which may strongly guide its performances. In this section, the first aim is therefore to study its behavior according to the value of these parameters. Once achieved, we give an overview of the obtained results under the optimized parameters.

The tests are achieved on blade servers with GNU/Linux operating system. Each server is equipped with 2 Intel Xeon 2.4 Ghz processors and 24 GB of RAM and each processor includes 4 physical cores. Each core is devoted to treat one instance. The tested instances are issued from the MaxSAT 2011 competition. We select 150 random Max-2-SAT instances, 150 random Max-3-SAT instances and 167 crafted Max-2-SAT ones (half from bipartite instances and half from maxcut ones). The running time of each solver is fixed to 900 seconds. Optimum values have been obtained on 450 instances (300 Max-2-Sat and 150 Max-3-Sat) with the complete Max-SAT solver maxsatz [4]. We only consider in our tests the instances for which the optimum is known.

The most natural comparison criterion of solver’s efficiency is the percentage of instances for which it found the optimum. When two solvers have very close results on this first criterion, we use two additional criteria: the time to find the optimum and the number of times this optimum is reached during the resolution.

A. AdaptNovelty++ Tuning

We first study the impact of the restart frequency (expressed as a factor of the size of the input formula) variations on AdaptNovelty++. The percentage of optimum found (Fig. 2 graphic (a)) is almost constant whatever the restart frequency value is. However, the median time to find the optimum and the number of times where the optimum is found (Fig. 2 graphic (b)) are respectively lower and bigger for small restart values (around one time the number of clauses of the input formulas).

One can note that AdaptNovelty++ found the optimum on all the random Max-3-SAT instances while it is significantly less efficient on Max-2-SAT ones. We see two possible explanations. The Max-SAT competition engages complete solvers and most of them includes unit propagation based inference rules. These solvers are more efficient on Max-2-SAT instances than on Max-3-SAT ones. Consequently, the Max-2-SAT instances are more difficult to solve than the Max-3-SAT ones, especially for solvers which do not use unit propagation as AdaptNovelty++. Another possible explanation may be related to the fact that WalkSat like algorithms are less efficient on over-constrained instance [15]. The Max-2-SAT instances contain more MUSes (and thus are more constrained) than the Max-3-SAT ones, and that could explain the observed results.

B. IRAnovelty++ Tuning

We have tuned individually and manually each parameter of IRAnovelty++: the ones relative to the diversification mechanisms, $FreqRestarts$ and $FreqRewriting$, and the ones which control the amount of inference, $LimUP$ and $LimRES$.

$FreqRestarts$: The results are shown in Fig. 3. The best percentage of optimum found (graphic (a)) are obtained with a restart frequency between 3 and 7 times the number of clauses of the formula. The median time to find the optimum and the median number of times the optimum is found (graphic (b)) suggest that 3 is the best performing value for the restart frequency. It should be noted that $FreqRestart$ has no impact on the size of the formula after transformation nor on the number of MUSes found during the transformation.

$FreqRewriting$: Fig. 4 presents the impact of the rewriting frequency on IRAnovelty++ performances. The percentage of optimum found (graphic (a)), the median time to find optimum and the median number of times the optimum was found (graphic (b)) seem relatively constant whatever the value of $FreqRewriting$ is. This indicates that the quality of the rewritings is relatively constant, and thus that this diversification mechanism may not be necessary. As for the restart frequency, $FreqRewriting$ has no impact on the size of the formula after transformation nor on the number of MUSes found during the transformation.

$LimUP$: This parameter defines the minimum ratio of original clauses which must remain in the transformed formula. When this ratio is reached, unit propagation is no longer applied and thus the formula is no longer transformed. Fig. 5 shows its impact on IRAnovelty++ performances. With value 100%, the inference rule is never applied and IRAnovelty++ acts as AdaptNovelty++. The more its value decreases, the more the percentage of instances for which IRAnovelty++ found the optimum increases. Best performances are obtained with value 15%. Smaller ones give slightly lower performances. We can see two possible explanations for this last observation: (1) too much time is spent in transforming the formula or/and (2) the size of the transformed formula slows down the local search. Graphics (c) and (d) of Fig. 5 show respectively the average growth of the size of the formula (in percentage of the original one) and the average number of MUSes found. It should be noted that no MUS are found on random Max-3-SAT instances.

$LimRES$: As the previous one, this parameter controls the amount of inference applied on the input formulas by limiting the application of the inference rule on clauses of high size. For a given conflict, if a clause of the refutation tree is higher than $LimRES$ then the inference rule is not applied and the conflict is simply ignored. As explained before, this limit have two purposes: limiting the size of the
transformed formula and spurring the deduction of MUSes. Fig. 5 shows the impact of this parameter on IRAnovelty++ performances. With values less than 2, the inference rule is never applied and the results are similar to the AdaptNovelty++ ones. The more LimRES value increases, the more the size of the transformed formula and the number of deducted MUSes increase. The best results are obtained with a value of LimRES between 5 and 8. Again, no MUS are found on random Max-3-SAT instances.

C. Results and Discussion

Table 1 compares the performances of IRAnovelty++ and AdaptNovelty++ with their optimized parameters (for AdaptNovelty++, FreqRestart = 1 and for IRAnovelty++, FreqRestart = 3, FreqRewriting = 50000, LimUP = 15% and LimRES = 7). For each solver, the column %O gives the percentage of instances for which the optimum has been found, fIO gives the median time spent in reaching the optimum for the first time and nTO gives the median number of times the optimum has been found during the resolution.

We can observe that IRAnovelty++ with optimized parameters greatly outperforms AdaptNovelty++ on Max-2-SAT instances (random and crafted). However, the results are more contrasting on the Max-3-SAT ones. Even if IRAnovelty++ manages to find the optimum for all the instances and more frequently than AdaptNovelty++, it spends more time in finding the optimum for the first time than AdaptNovelty++ does. In our opinion, this can be explained by the way the inference rule is applied. Except in particular cases (when few clauses of the refutation tree share same
The inference rule produces new clauses of size equal (with input clause of size two) or higher (with input clauses of size greater than 2) than the original ones. This is not a problem on Max-2-SAT instances, where the particular case described above happened sufficiently often to allow the production of unit and empty clauses. But this is not the case on Max-3-SAT instances. Consequently, on Max-3-SAT instances, IRAnovelty++ spent time transforming the formulas, thus increasing their sizes (and making them harder for the local search) with a smaller gain than on Max-2-SAT instances. Several solutions exist to improve this situation, like leading the application of the inference rule by using dedicated heuristics, or more simply by applying it only when the resulting clause sizes are smaller or equal to the original ones.

Fig. 7 compares the best solution found by AdaptNovelty++ and IRAnovelty++ on 263 instances of the benchmark. Each point represents an instance, with in the horizontal axis the best solution found by AdaptNovelty++ and in the vertical axis the one found by IRAnovelty++. On all the represented instances, our solver found generally better solutions than AdaptNovelty++.

Eventually, it should be noted that on few instances IRAnovelty++ manages to find all MUSes. In this situation,
it can prove the optimality of the solution as a complete solver does. However, this is relatively rare (around 1% of the instances) and works only on instances with small optimum (thus small number of MUSes).

VII. CONCLUSION

We have presented in this paper a new solver, IRAnovelty++, which includes inference rules in a classic local search algorithm to solve the Max-SAT problem. The extended tests that we have performed provide useful information on the behavior of our solver and show its robustness on many instances. The results show that the performances of the initial local search are significantly improved. That proves the interest of our approach. Moreover, the way we use the Max-resolution (by using it as a full memorization mechanism for iteratively building MUSes) has never been made before and seem promising.

As future direction, we will work on extending our solver to Weighted and Partial Max-SAT and improving the application of the inference rule on Max-3-SAT instances.

The recent efforts on solving Max-SAT are focused on the exact methods and less interest is given to incomplete methods. Our work aims also in contributing in the development of such approaches.

REFERENCES