Acquiring Local Preferences of Weighted Partial MaxSAT

Hong Huang, Laura Climent and Barry O'Sullivan
Insight Centre for Data Analytics
Department of Computer Science, University College Cork, Ireland
{hong.huang|laura.climent|barry.osullivan}@insight-centre.org

***Abstract*—Many real-life problems can be formulated as boolean satisfiability (SAT). In addition, in many of these problems, there are some hard clauses that must be satisfied but also some other soft clauses that can remain unsatisfied at some cost. These problems are referred to as Weighted Partial Maximum Satisfiability (WPMS). For solving them, the challenge is to find a solution that minimizes the total sum of costs of the unsatisfied clauses. Configuration problems are real-life examples of these, which involve customizing products according to a user's specific requirements. In the literature there exist many efficient techniques for finding solutions having minimum total cost. However, less attention has been paid to the fact that in many real-life problems the associated weights for soft clauses can be unknown. An example of such situations is when users cannot provide local preferences but instead express global preferences over complete assignments. In these cases, the acquisition of preferences can be the key for finding the best solution. In this paper, we propose a method to formalize the acquisition of local preferences. The process involves solving the associated system of linear equations for a set of complete assignments and their costs. Furthermore, we formalize the characteristics and size of the complete assignments required to acquire all local weights. We present an heuristic algorithm that searches for such assignments which performs promisingly on many benchmarks from the literature.**

I. Introduction

The challenge of solving many real-life problems is often beyond simply finding a satisfiable assignment due to the preferences specified within the problem. Frequently, users specify preferences for certain combinations and the objective becomes finding the optimal solution. This solution maximizes the satisfiability of the specified preferences. Some typical examples of real-life applications that present such characteristics are configuration problems, scheduling problems, etc. In these problems, the industries intend to configure their products based on user's preferences. The term of configuration was used to introduce a specific form of a design task in [1]. In configuration problems a configurator can be regarded as a interactive system by which users configure a product according to their preferences [2]. By configuring a product, user preferences and compatibility constraints are satisfied [3].

Some examples of very well known configurable products are vehicles, computers, etc. (see [4] for a survey of product configuration frameworks). Configuration problems can involve thousands of variables that determine many different configurations [5]. In [6], the authors mentioned that while the task of traditional configuration problem is finding solutions that meet user requirements strictly, it is more natural to express the problem also in terms of preferences. In this case, requirements that need to be satisfied strictly can be considered as hard constraints, and the preferences are called soft constraints.

Soft constraints allow users to express their local preferences over constraints, variables and/or tuples. However, in more recent works (e.g. [6]) novel ways of eliciting preferences is considered whereby users express preferences over complete assignments. The challenge then becomes how to model such general preferences locally. Recall that the objective of solving problems with preferences is to find an optimal solution. For this reason, it is very important to acquire local preferences with the motivation to help find an optimal solution of the problem. In this context, this paper focuses in the local preferences acquisition task.

The Weighted Partial MaxSAT problem (WPMS) allows us to express local preferences since it incorporates weights associated with soft clauses. Then, by means of the weights associated to the soft constraints, the users can fix the preferences locally, over each soft constraint. Each weight is the penalty associated with a solution that does not satisfy the clause. The higher the weight of a clause, the higher is the preference of the satisfaction of that clause. The objective function is to find a solution to the hard clauses that minimizes the sum of the weights of the unsatisfied soft clauses.

In this paper we consider the situation in which a set of complete assignments/solutions to a WPMS and their corresponding objective function values are known; as previously mentioned, this information could be provided by the users. However, in general we assume that the specific weights associated with the soft clauses are unknown. Our approach focuses on the scenario in which some weights are known. The main motivation for acquiring such weights is to be able to obtain the optimal solution.

In this paper, we have formalized the acquisition of local preferences through the formalism of Weighted Partial MaxSAT (Section II). We have approached this task by modeling the problem as a system of linear equations and subsequently using linear algebra techniques to solve the resultant system, which we describe in Section III. Further-

more, we are interested in the characteristics and size of the complete assignments, and their total costs, required to acquire all local weights, which we consider in Section IV. Due to the high complexity of solving such a problem, we propose an heuristic algorithm in Section V. In order to show its effectiveness, we evaluate it with a wide range of WPMS benchmarks in Section VI.

II. Preliminaries

The SAT and MaxSAT problems are well known to be NP-Hard [7], [8]. In SAT, for a given boolean formula, the objective is to find an assignment that satisfies all clauses in the formula. In MaxSAT the objective is to find an assignment that maximizes the number of satisfied clauses.

A CNF is a set of clauses $(C_1, C_2, ..., C_m)$ in conjunctive normal form where C_i is a disjunction of literals. A literal is a boolean variable x or its negation $\overline{x}$ [9]. The SAT problem is to find an assignment $var(f) \rightarrow \forall C_i \in f, C_i \rightarrow True$. Let ϕ be a function such that $\phi(C) \rightarrow 1$ iff $C \rightarrow True$, otherwise $\phi(C) \rightarrow 0$. The MaxSAT problem is to find an assignment that satisfies the maximum number of clauses: $var(f) \rightarrow max(\sum_1^m \phi(C_i))$.

The following notation was described in [10]. The weighted partial MaxSAT (WPMS) problem is composed of a set of hard clauses and a set of soft clauses, each with a weight. The weight associated with each soft clause C_i is denoted as w_i. A weighted clause is a pair (C_i, w_i), where w_i is natural number. The formula defining a WPMS problem is as follows:

$$\varphi = \{(C_1, w_1), .., (C_m, w_m), .., (C_{m+1}, \infty), .., (C_{m+m'}, \infty)\}$$

where the first m clauses are soft and the last m' are hard. The weight of hard clauses should be ∞ compared to soft ones. $var(\varphi)$ and $var(C)$ denote the set of variables in φ and C, respectively. I is a function that gives the truth assignment for literals. Then, $I : var(\varphi) \rightarrow \{0, 1\}$ is noted as the assignment for φ. $cost(\varphi) = \sum_{i=1}^m w_i(1 - I(C_i))$. Each w_i is added to the total cost iff clause C_i is unsatisfied. The objective function of the WPMS is to minimize $cost(\varphi)$, i.e., to minimize the total cost associated with the set of unsatisfied clauses.

III. The Approach

Our objective is to acquire the unknown weights of the soft clauses of a WPMS problem. Without loss of generality and with the purpose of simplifying notation, we consider that all the weights of the m soft clauses of the WPMS are unknown. We use the fact that, by definition, the relation between w and $cost(\varphi)$ is linear. We define a set $B = \{b_1, \ldots, b_m\}$ of m 0/1 variables. We define each 0/1 variable as $b_i \leftrightarrow (1 - I(var(C_i))$. Therefore, b_i is equal to 1 if the clause C_i is unsatisfied. Otherwise b_i is equal to 0. Then, for each complete assignment provided by the user, we obtain its associated B. By also using the global cost provided by such an assignment, we obtain a linear equation of the form $cost(\varphi) = \sum_{i=1}^m w_i b_i$, where the variables representing the weights (w) are unknown. Note that this is equivalent to the definition of $cost(\varphi)$ provided earlier. Therefore, by using several complete assignments, we obtain different linear equations.

In linear algebra it is well known that a system of linear equations is determined when the number of linear equations is equal to the number of unknown variables, and all the linear equations are independent of each other. The determined case means that there is only a single unique solution, and therefore all the unknown variables can be determined. On the other hand, the system of linear equations is undetermined if the number of linear independent equations is fewer than the number of variables. For this reason, we aim to achieve a system of m linearly independent (*LI*) equations. We would like to mention that the linear equations of some complete assignments provided by the user could be dependent and, therefore, this is an aspect that must be taken into consideration. We define the set of complete assignments provided by user as $S = \{s_1, \ldots, s_m\}$. Then, the coefficient matrix of our system of linear equations can be defined as in Figure 1. Where, b_{ij} is a 0/1 variable expressing if the soft clause C_j is satisfied by the complete assignment s_i. If b_{ij} is equal to 1, it means that it is not satisfied. Otherwise, it is satisfied. For example, if $b_{52} = 1$ then the complete assignment s_5 does not satisfy the soft constraint C_2.

$$\begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1m} \\ b_{21} & b_{22} & \cdots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mm} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} = \begin{bmatrix} cost(\varphi)_1 \\ cost(\varphi)_2 \\ \vdots \\ cost(\varphi)_m \end{bmatrix}$$

$$\Updownarrow$$

$$\left[\begin{array}{cccc|c} b_{11} & b_{12} & \cdots & b_{1m} & cost(\varphi)_1 \\ b_{21} & b_{22} & \cdots & b_{2m} & cost(\varphi)_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mm} & cost(\varphi)_m \end{array}\right] \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}$$

$$\Downarrow$$

$$\left[\begin{array}{cccc|c} b'_{11} & b'_{12} & \cdots & b'_{1m} & cost(\varphi)'_1 \\ 0 & b'_{22} & \cdots & b_{2m} & cost(\varphi)'_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & b'_{mm} & cost(\varphi)'_m \end{array}\right]$$

Figure 1. The coefficient matrix of our system of linear equations.

For solving this system of linear equations we use Gaussian elimination [11], [12], which performs elementary row reductions by performing operations over the coefficient matrix. With the first part of the process of Gaussian elimination the original matrix of coefficients is converted

into its row echelon form. The second part of the algorithm continues the row reduction until its convergence to a row reduced echelon form, in order to find the values of the undetermined variables, i.e. to find the solution of the system of linear equations. The formula to solve the unknown variable $w_i, i \in \{1, ..., m\}$ is $w_i = \frac{1}{b'_{ii}} \left(cost(\varphi)'_i - \sum_{j=i+1}^{m} b'_{ij} w_j \right)$. After solving the mth row of the matrix we are able to acquire w_m. Then we substitute its value into the equation of $(m-1)$th row to acquire w_{m-1}, etc.

Note that it is guaranteed that there is a unique solution for a system of linear equations if the column vectors of the coefficient matrix are all independent. That is to say if several variables b_i take the same values for all the complete assignments they are linearly dependent (*LD*) and therefore we cannot acquire the weights associated with them. The same scenario occurs when a b_i has an associated column vector with all its elements set to 0. In this case, this column vector is *LD* with respect to the other column vectors and we cannot learn its weight. Even so, in real applications this scenario does not make sense because it would mean that the soft clause C_i is always satisfied by all the complete assignments provided by the user. Therefore, there is a contradiction since this clause is not soft because it always must be satisfied. And consequently, it does not have an associated weight. A similar case occurs when a variable b_i has an associated column vector with all its elements equal to 1. In practice this means that the soft clause C_i is not satisfied by any complete assignments provided by the user.

An Example. Given a WPMS that has three soft unary clauses with unknown weights $\varphi = \{(C_1, w_1), (C_2, w_2), (C_3, w_3), (C_4, w_3)\}$, where the clauses are: $C_1 : x_1$, $C_2 : (x_1 \vee x_2)$, $C_3 : (x_1 \vee x_3 \vee x_4)$ and $C_4 : (\overline{x}_2 \vee x_3)$. Let's consider that the set of solutions in Table I and their global costs are known, then we can obtain the associated $\{b_1, b_2, b_3, b_4\}$ (Table I).

Table I
SOLUTIONS AND THEIR COSTS.

Solution				Is C_i UNSAT?				cost(φ)
x_1	x_2	x_3	x_4	b_1	b_2	b_3	b_4	
1	1	0	1	0	0	0	1	1
0	1	1	1	1	0	0	0	2
0	1	0	0	1	0	1	1	6
0	0	0	0	1	1	1	0	7

Hence, the coefficient matrix of the associated linear equations can be defined as per Figure 2. By Gaussian elimination we can acquire the values of the unknown variables which, in our case, represent the unknown weights associated with the soft clauses. For this toy example we obtain $w_1 = 2$, $w_2 = 2$, $w_3 = 3$ and $w_4 = 1$ (see the last matrix). Then the objective function of the WPMS is to minimize $cost(\varphi) = 2x_1 + 2x_2 + 3x_3 + x_4$. After acquiring the clause weights we can use a WPMS solver and find a solution that has a better cost than the given solutions (e.g. $(x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1)$ has a cost of 0).

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 6 \\ 7 \end{bmatrix}$$

$$\Updownarrow$$

$$\left[\begin{array}{cccc|c} 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 2 \\ 1 & 0 & 1 & 1 & 6 \\ 1 & 1 & 1 & 0 & 7 \end{array}\right] \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 3 \\ 1 \end{bmatrix}$$

Figure 2. Linear equations for Example 1.

IV. ACQUIRING THE UNKNOWN WEIGHTS

In Section III we mentioned that in order to acquire all the m unknown weights that compose the associated system of linear equations, the number of linear equations should be equal to the number of unknown variables and that all linear equations should be independent of each other. In toy problems, such as the one introduced earlier, this does not represent a challenge because of the low number of unknown weights. However, for large instances, such as the ones derived from real applications, there could be thousands of unknown weights. In such cases, finding thousands of complete assignments whose equations are linearly independent (LI) among them becomes a challenging task. Let's consider a collection of *LI* vectors S involving the same m variables. As previously mentioned, each vector is denoted as b_k and b_{kj} refers to the coefficient of the variable in position j. The same notation is applied to b'_j, where b' is a new vector that we are considering. Then, by definition b' is *linearly dependent* (*LD*) with respect to the set of vectors S if:

$$\exists \vec{c} \in \mathbb{R}^{|S|} \forall j \in \{1, \ldots, m\} : b'_j = \sum_{k}^{|S|} c_k b_{kj}. \qquad (1)$$

From the negation of Equation 1, it can be deduced that b' is linearly *independent* with respect to the set of vectors S if:

$$\forall \vec{c} \in \mathbb{R}^{|S|} \exists j \in \{1, \ldots, m\} : b'_j \neq \sum_{k}^{|S|} c_k b_{kj}. \qquad (2)$$

A system of linear equations has only one solution, and therefore all its variables can be determined, if it is

composed of m *LI* vectors, where m is the number of variables involved in the system. Then, following the above reasoning, each vector of the system has to satisfy the linearly independence condition of Equation 2 with respect to the other $m-1$ vectors. From this reasoning we can state that the set of m *LI* vectors involving m variables is computed as:

$$\forall i \in \{1,\ldots,m\} \forall \vec{c} \in \mathbb{R}^{m-1}, \\ \exists j \in \{1,\ldots,m\} : b_{ij} \neq \sum_{k \neq i}^{m-1} c_k b_{kj}. \tag{3}$$

V. An Heuristic Algorithm

The objective function of the WPMS with n-ary clauses is to minimize $f(x) = \sum w_i b_i$, where each b_i is a 0/1 variable that is equal to 1 if the associated soft clause is unsatisfied and 0 otherwise. In order to acquire m unknown weights we present an heuristic algorithm that uses a state-of-the-art SAT solver for finding m solutions whose associated equations are linearly independent. Algorithm 1 aims to find up to m *LI* solutions, which are stored in $\mathcal{X}$. Then, if the size of $\mathcal{X}$ is equal to m, it stops since it means that by using Gaussian elimination we are able to find all the m unknown weights.

Algorithm 1: m-LinearlyIndependentSols

```
   Data: A (the propositional formula), B ← {b_1, ..., b_m}
   Result: X (a set of up to m LI solutions).
1  X ← ∅ // stores the LI solutions found;
2  F ← ∅ // stores the new clauses to be added;
3  i ← 1;
4  while s ← (getNextSolution(A ∧ b_i ∧ F)) do
5      if checkLI(X ∪ {s}) then
6          X ← X ∪ {s};
7          if |X| = m then
8              return X;
9          R ← reduced row echelon matrix of X;
10         for b_i ∈ B do
11             if b_i is determined in R then
12                 B ← B \ {b_i} // update undetermined
                     variables;
13         F ← (¬(x_1 ↓_B) ∧ ··· ∧ ¬(x_|X| ↓_B)), where
             x_i ∈ X ;
14     do
15         i ← (i%|B| + 1) // i iterates over |B| ;
16     while b_i ∉ B;
17 return X;
```

For this purpose, we use `getNextSolution` which returns the next solution of a propositional formula: here, any complete SAT solver can be used. Then, when there are no more solutions returned by such function, Algorithm 1 stops. The input of `getNextSolution` is the propositional formula of the hard clauses of the WPMS instance analyzed (denoted as A), the next targeted undetermined variable ($b_i \in B$), and F. $B \leftarrow \{b_1,\ldots,b_m\}$ are the variables associated with the soft clauses with unknown weights. F is a set of new clauses that are automatically updated in order that `getNextSolution` avoids finding new solutions that are a combination of the previous solutions found.

However, even if F is able to prevent some *LD* solutions, unfortunately, they are not all the possible *LD* solutions. For this reason, after obtaining a new solution, Algorithm 1 checks whether the solution is linearly independent with respect to the rest of solutions in $\mathcal{X}$ (function `checkLI` in line 5). If the checking is positive, then the new solution is added to $\mathcal{X}$. This check for linearly independence can be done by comparing the singular value (sv) of $s\downarrow_B, \forall s \in \mathcal{X}$ with tolerance (T) where $s\downarrow_B$ is the corresponding assignments for $b's$, or checking whether there exists a row with all entries equal to 0 in the result matrix after computing the reduced row echelon form using Gaussian elimination. In order to get sv, singular value decomposition must be done for input $\mathcal{X}$. If $|\{sv \in SVD | sv > T\}| = |\mathcal{X}|$ then $\mathcal{X}$ is *LI*.

We would like to mention that in order to make it efficient, the search process should prevent previously found solutions from being found again. In our algorithm if a new solution is *LI* then that solution is blocked. Since we updated F by forbidding the assignment of B which is the projection of a found *LI* solution to B, since we were interested in whether the soft clauses are satisfied or not. By doing that, we are not just preventing the rediscovery of that *LI* solution, but also preventing the discovery of other solutions that will have the same projection onto B. For example, if we have $s1 : (1,0,0,1) \rightarrow B : (1,0,0)$ and $s2 : (1,1,1,0) \rightarrow B : (1,0,0)$. After we find $s1$ and it is *LI* with respect to the set of solutions ($\mathcal{X}$), then we update F by forbidding the projection of $s1$ to B which is $B : (1,0,0)$. Then the search process will not return $s2$ since the projection of $s2$ to B is already forbidden.

If a solution returned from the search process is *LD*, we handle this situation using the function `getNextSolution`. In our algorithm, the function `getNextSolution` always returns a new solution that it has not been discovered yet. Therefore, it automatically avoids solutions that have been previously returned. In order to find the set of *LI* solutions, the search process is mostly equal to searching for the solutions of the original problem. In each iteration the process searches for a solution to the new problem which is adapted from the original SAT problem by adding extra constraints ($b_i \wedge F$). In the worst case, the search process for the full set of *LI* solutions (m *LI* solutions) requires us to find all the solutions for the original SAT problem.

In addition, B is updated if any $b_i \in B$ is determined after performing Gaussian elimination for obtaining the reduced row echelon operation matrix of $\mathcal{X}$. In case that there is a row where $b_i = 1$ and the other entries are zeros, b_i is

determined. Then, the determined $b's$ are deleted from the B set. F is also updated so that F contains the negations of the solutions in $\mathcal{X}$ but only for the variables in B. This is due to the fact that the same combinations of the variables in B of a solution in $\mathcal{X}$ would be *LD*.

In lines 14-16, at the end of an iteration, we re-compute the value of index i. The index i is the index for b in the original problem. Again, we mentioned above that in each iteration of the search process we focused on searching for a solution that includes a specific soft clause in its projection to B. We use the soft clause index i to specify such a soft clause, e.g., $A \wedge b_i \wedge F$ as the input of `getNextSolution` in line 4. The idea is no matter the search process which is targeting a specific soft clause (b for specific i that is b_i), we return a *LI* solution or a *LD* solution, we will attempt to search for a solution for next soft clause in the index (b_{i+1}). If there is a row where $b_i = 1$ and the other entries are zeros – again that means b_i is determined – we jump over that index value and keep plus 1 until b_i is not determined. Then it will jump back to line 4 and search for a *LI* solution which including the specific b_i in its projection to B.

VI. Evaluation

The objective of our experiments was to evaluate the effectiveness of Algorithm 1 in finding the full set of m *LI* solutions and subsequently using them for acquiring the m unknown weights in a WPMS instance. In our experiments, we used SCIP[1] as a SAT solver for finding the solutions that satisfy the hard clauses. As previously mentioned, after each iteration, we added or updated the clauses in the SAT instance in order to find a new *LI* solution. We set up a stop-condition to terminate the search: an iteration-limit $\#NoNewSolIters > \#SoftClaus - \#LISols$.

We performed our experiments on a set of benchmarks taken from the MaxSAT evaluation competitions.[2] These datasets are either from `random`, `crafted` or `industrial` subsets of the `Weighted Partial MaxSAT` catalog of MaxSAT Evaluations competitions 2006. Table II shows the benchmarks analyzed in ascending order by the number of total clauses in the dataset. Column `#Total Claus` represents the number of total clauses in the dataset. The hard clauses and soft clauses in the column `#Hard Clause` and `#Soft Clause` respectively. We assume that all the soft clauses have unknown weights. The number of variables is represented as `#Vars`. In the `Result` part of the Table II, we list the number of *LI* solutions found, the number of acquired unknown weights, runtime and the number of iterations in experiment.

Regarding the variety of the instances evaluated, there are some that are small, with no more than 100 soft clauses, e.g. `8.wcsp.dir` has only 8 soft clauses. In addition, there are some big instances with several thousand clauses, e.g. `cap72.wcsp` has 3978 total clauses, 1664 hard variables and 814 soft clauses. We have evaluated instances with unary soft clauses, e.g. `29.wcsp.dir`, and n-ary soft clauses, e.g. `cat_paths_60_70_0001.txt.wcnf`.

The results of the experiments performed show that for many benchmarks we were able to find the same number of *LI* solutions as the number of soft clauses. Therefore, in such cases, we were able to acquire all the unknown weights of all the soft clauses by using Gaussian elimination. These cases are highlighted in Table II in the columns `#LI Sols` and `#ACQ Claus`. The total number of iterations of Algorithm 1 that are necessary to find all the unknown weights is reasonable. There are "perfect" scenarios in which the number of iterations are the minimum possible, i.e. they are equal to the number of unknown weights to be acquired. These cases are highlighted in Table II in the column `#Iters`. In addition, in many other cases, the number of iterations is not much higher than this "perfect" case.

Table II shows that for small benchmarks, it is more likely to find the full set of *LI* solutions. However, there are other instances for which many *LI* solutions share the same values for certain b variables. For example, variables always assigned to 0 or 1, or those variables that are *LD* to each other. In both cases, they have the same value in all the solutions. Therefore, the unknown weights associated with those b variables cannot be acquired. An example of a benchmark that has this situation is `cat_paths_60_70_0002.txt`. For this reason, there does not exist a full set of *LI* solutions for such instance.

In Table II, the benchmarks whose `#LI Sols` column value is underlined means that it is not possible to find a full set of *LI* solutions because there does not exist such a set. The other benchmarks whose results show that we cannot find such a set of solutions in the experiment are due to the limit on the number of iterations. In the cases for which we cannot find a full set of *LI* solutions, for the small benchmarks it is more likely to find a close number of *LI* solutions to the unknown weights. For example, we found 65 *LI* solutions of 70 soft variables in the `cat_paths_60_70_0002.txt` benchmark. For the very small benchmarks the search is fast ($<$1 second for the benchmark `8.wcsp.log.wcnf` and `8.wcsp.dir.wcnf`). For small benchmarks, the runtime is reasonable. However, the runtime is high for big benchmark (e.g `29540` seconds for `505.wcsp.dir`). This is due to the algorithm iterating at least as many times as the number of unknown variables in order to find the full set of *LI* solutions and for each iteration, the SAT solver runs until a solution is found. In addition, the runtime of the SAT solving process might increase after iterations since we add new clauses to the original instance. If a solution given back by the solver is *LI* with respect to the set found so far, we

[1] http://scip.zib.de/

[2] http://www.maxsat.udl.cat/

Table II
BENCHMARKS AND RESULTS.

Benchmark Details					Result			
Benchmark	#Total Clause	#Hard Clause	#Vars	#Soft Clause	#LI Sols	#ACQ Clause	Time(s)	#Iters
8.wcsp.log	25	17	12	8	**7**	**7**	<1	15
8.wcsp.dir	29	21	20	8	**8**	**8**	<1	**8**
cat_paths_60_70_0001.txt	397	324	73	73	**73**	**73**	271	74
cat_paths_60_70_0004.txt	442	372	70	70	**70**	**70**	150	73
54.wcsp.log	479	412	96	67	**67**	**67**	171	**67**
54.wcsp.dir	508	441	154	67	41	13	193	109
1502.wcsp.log	534	325	311	209	205	201	6541	415
cat_paths_60_70_0003.txt	540	470	70	70	**70**	**70**	144	**70**
cat_paths_60_70_0007.txt	551	477	74	74	**74**	**74**	249	**74**
cat_paths_60_80_0002.txt	553	472	81	81	**81**	**81**	390	**81**
cat_paths_60_70_0006.txt	566	495	71	71	**71**	**71**	142	**71**
cat_paths_60_70_0000.txt	598	528	70	70	**70**	**70**	137	74
cat_paths_60_70_0005.txt	605	533	72	72	**72**	**72**	111	**72**
cat_paths_60_70_0002.txt	610	540	70	70	65	19	111	136
cat_paths_60_80_0001.txt	612	531	81	81	**81**	**81**	292	**81**
cat_paths_60_80_0003.txt	615	535	80	80	**80**	**80**	272	**80**
cat_paths_60_80_0006.txt	633	551	82	82	**82**	**82**	283	**82**
1502.wcsp.dir	636	427	515	209	81	15	1655	291
29.wcsp.log	692	610	101	82	**82**	**82**	300	**82**
29.wcsp.dir	711	629	139	82	60	14	476	143
503.wcsp.log	934	791	201	143	**143**	**143**	5496	171
404.wcsp.log	1037	937	129	100	**100**	**100**	619	**100**
404.wcsp.dir	1066	966	187	100	99	97	737	199
normalized-mps-v2-20-10-p0033.opb.msat.wcnf	1494	1461	548	33	18	6	27	52
cat_paths_60_140_0002.txt	1833	1692	141	141	**141**	**141**	3436	143
cat_paths_60_130_0000.txt	1936	1806	130	130	129	66	2056	391
cat_paths_60_140_0003.txt	1954	1810	144	144	**144**	**144**	5395	179
42.wcsp.log	2016	1826	247	190	**190**	**190**	10132	328
cat_paths_60_140_0001.txt	2024	1881	143	143	**143**	**143**	4697	162
cat_paths_60_150_0003.txt	2029	1878	151	151	**151**	**151**	5254	**151**
42.wcsp.dir	2073	1883	361	190	177	152	7836	368
cat_paths_60_170_0005.txt	2089	1917	172	172	**172**	**172**	9807	286
cat_paths_60_150_0002.txt	2092	1942	150	150	**150**	**150**	6072	202
cat_paths_60_150_0000.txt	2097	1947	150	150	**150**	**150**	4618	193
cat_paths_60_160_0002.txt	2276	2114	162	162	**162**	**162**	8720	165
cat_paths_60_160_0000.txt	2299	2139	160	160	**160**	**160**	7146	220
cat_paths_60_160_0001.txt	2377	2216	161	161	**161**	**161**	8247	242
cat_paths_60_160_0003.txt	2417	2256	161	161	**161**	**161**	6296	178
cat_paths_60_150_0001.txt	2460	2310	150	150	**150**	**150**	5580	168
cat_paths_60_170_0004.txt	2543	2372	171	171	**171**	**171**	6792	291
cat_paths_60_170_0000.txt	2803	2633	170	170	**170**	**170**	6328	177
cat_paths_60_170_0003.txt	3016	2845	171	171	**171**	**171**	7950	**171**
408.wcsp.log	3149	2949	264	200	**200**	**200**	18228	220
505.wcsp.dir	3536	3296	552	240	**240**	**240**	29540	341
cap61.wcsp	3978	3164	1664	814	47	10	17985	861
cap62.wcsp	3978	3164	1664	814	48	11	17883	862
cap71.wcsp	3978	3164	1664	814	52	10	18232	866
cat_sched_60_160_0000.txt	5314	5154	160	160	143	52	1303	304
cap81.wcsp	6273	5000	2600	1273	31	11	26866	1304
cap82.wcsp	6273	5000	2600	1273	30	13	27113	1304
cap91.wcsp	6273	5000	2600	1273	37	15	27955	1310
cap101.wcsp	6273	5000	2600	1273	37	17	27154	1310
cap102.wcsp	6273	5000	2600	1273	40	22	27277	1313
1504.wcsp.log	6593	5988	929	605	232	2	176013	488

say it is a good solution. The accuracy for the whole search is the percentage of the solutions given back by the solver that are the good solutions. It can be observed in Table II that regardless of the size of benchmark, high accuracy implies less time to finish the search. By comparing the result of benchmark `503.wcsp.log` and `404.wcsp.dir`, there is a huge difference in runtime between them. Table II shows that for small benchmarks the accuracy of search is quite high which means that it has not wasted much time finding the *LD* solutions.

After the search process generated the full set of *LI* solutions, we can use those solutions to compute the associated weights for soft clauses. Again even with some of the *LI* solutions it is still possible for us to acquire some associated weights for soft clauses, but in that case we are not able to acquire all the associated weights for B. This case was also shown in Table II.

In some experiments the number of *LI* solutions returned by the search process and the number of acquired of associated weights for B are less than the number of soft clauses in the problem. For example, in the experiment for `1502.wcsp.log`, the number of soft clauses is 209. But in the search process we only can find 205 *LI* solutions which is not a full set of *LI* solutions ($205 < 209$). Without the full set of *LI* solutions, we still can partially solve the problem by acquiring associated weights for 201 soft clauses. In this case, after solving the system of linear equations that is composed of 205 projection vectors from solutions to B, we determined 201 associated weights for soft clauses. The associated weights for the remaining eight soft clauses are either dependent on each other or unknown after solving the system of linear equations. It means that, if we cannot solve the problem perfectly (finding m *LI* solutions in the search process), we still can acquire some associated weights for soft clauses.

As we mentioned before, we computed the associated weights for soft clauses by solving the system of linear equations. Again, the system of linear equations is composed of the projection of full set of *LI* solutions to B. Many studies have shown that solving a system of linear equations is efficient [13]–[16]. In our case, the runtime for solving the system of linear equations in the experiments was within 2 seconds.

While the above experiments shows that we can acquire the m unknown associated weights for soft clauses if we have the full set of m *LI* solutions and the associated weights for solutions given by user, we are also interested in cases where a user can only roughly estimate the weights of solutions or give the order of solutions. We performed another set of experiments in which we applied some errors to the associated weights for solutions to see if the computed optimal solutions change drastically. In the case that the user can only give the order of solutions, we could consider the given order as some rough weights, e.g. obtaining $S_1 > S_2 > S_3 > ...$ from $S_1 = 100, S_2 = 200, S_3 = 300, ...$ In this set of experiments, we used the acquired weights for soft clauses to compute the optimal solution and its weight (denoted as O). After errors were applied into the weights of solutions, we acquired the new associated weights for soft clauses and computed the new optimal solution and its weight (denoted as O'). Then we computed the Δ between O' and O.

In this experiment, there are three error intervals of 10%, 20% and 30% for each benchmark. Also, for each benchmark we considered two different options with respect to the order of solutions after applying errors. First relates to the order of solutions after applying error as before, the other does not intentionally maintain the order.

In Table III for each interval we present Δ O'-O and the percentage of Δ to the weight of optimal solution before applying errors (denoted as Δ/O). The results shown in Table III are the average result of 10 experiments. 0 in the cell means in that case the optimal solutions before and after applying errors are the same. Otherwise, the Δ/O percentage shows how far the new optimal solution deviated from the optimal solution before applying errors. The table also shows that for some benchmarks, for certain error intervals, the optimal solutions before and after are the same. In other cases, the difference between O and O' is small and it is increased if the error interval increases. Comparing the results in the same error interval and benchmark but different order options, it shows that if the order of solutions after applying errors is maintained as before, the results are better.

The results show that if the user can only roughly estimate the weights of solutions, or just specify the order of solutions, which are similar to applying errors into the weights of solutions, we still can find a close solution to the optimal.

VII. Conclusions and Future Work

In this paper we proposed a method for acquiring the weights of soft clauses in Weighted Partial MaxSAT problems. Configuration problems, for which the users specify global preferences instead of local preferences, represent a motivation for this work. In addition, we exemplified the principles behind our method and formalized the problem of acquiring weights by modeling and solving a system of linear equations. Furthermore, we formalized the problem of finding m linearly independent solutions, which is necessary for acquiring all the m unknown weights. Due to the high complexity of solving such a problem, we also presented a heuristic approach to this. We evaluated our heuristic approach with many well-known benchmarks and we obtained promising results especially when the number of unknown weights was not excessive. We showed that our method acquired the unknown weights in a reasonable number of iterations. However in some cases the runtime was high.

In the future we are planning to work on a more restrictive variant of the problem described in this paper in which we

Table III
APPLIED ERRORS AND THE RESULTING DELTA OF OPTIMAL SOLUTIONS.

	not intentionally in maintaining the order of solutions						remain the order of solutions after applying errors as before					
	10%		20%		30%		10%		20%		30%	
Benchmark	ΔO'-O	Δ/O	ΔO'-O	Δ/O	ΔO'-O	Δ/O	ΔO'-O	Δ/O	ΔO'-O	Δ/O	ΔO'-O	Δ/O
29.wcsp.log	**0**	**0%**	**0**	**0%**	**0**	**0%**	**0**	**0%**	**0**	**0%**	**0**	**0%**
404.wcsp.log	**0**	**0%**	**0**	**0%**	**0**	**0%**	**0**	**0%**	**0**	**0%**	**0**	**0%**
54.wcsp.log	**0**	**0%**	**0**	**0%**	**0**	**0%**	**0**	**0%**	**0**	**0%**	**0**	**0%**
8.wcsp.log	**0**	**0%**	**0**	**0%**	**0**	**0%**	**0**	**0%**	**0**	**0%**	**0**	**0%**
..60_140_0002..	139	0.150%	293	0.316%	472	0.509%	**0**	**0%**	**0**	**0%**	**0**	**0%**
..60_140_0003..	68	0.073%	192	0.207%	282	0.304%	**0**	**0%**	**0**	**0%**	**0**	**0%**
..60_150_0001..	73	0.079%	224	0.243%	400	0.433%	23	0.025%	21	0.023%	21	0.023%
..60_160_0001..	128	0.129%	86	0.087%	388	0.391%	5	0.005%	17	0.017%	17	0.017%
..60_170_0000..	58	0.047%	317	0.259%	434	0.355%	13	0.011%	13	0.011%	26	0.021%
..60_170_0003..	65	0.053%	183	0.149%	484	0.394%	**0**	**0%**	**0**	**0%**	8	0.007%
..60_70_0006..	63	0.143%	88	0.200%	175	0.397%	3	0.007%	4	0.009%	14	0.032%
..60_80_0002..	15	0.033%	23	0.051%	127	0.281%	**0**	**0%**	15	0.033%	16	0.035%

can only compute k solutions whose associated equations are linearly independent, where $k < m$. Therefore, we want to find the k solutions that maximize the number of weights acquired.

ACKNOWLEDGEMENTS

This work was supported by Science Foundation Ireland under grant SFI/12/RC/2289.

REFERENCES

[1] J. P. McDermott, "R1: A rule-based configurer of computer systems," *Artif. Intell.*, vol. 19, no. 1, pp. 39–88, 1982. [Online]. Available: http://dx.doi.org/10.1016/0004-3702(82)90021-2

[2] B. O'Sullivan, Ed., *Recent Advances in Constraints, Joint ERCIM/CologNet International Workshop on Constraint Solving and Constraint Logic Programming, Cork, Ireland, June 19-21, 2002. Selected Papers*, ser. Lecture Notes in Computer Science, vol. 2627. Springer, 2003.

[3] U. Junker, "Configuration," in *Handbook of Constraint Programming*, ser. Foundations of Artificial Intelligence, F. Rossi, P. van Beek, and T. Walsh, Eds. Elsevier, 2006, vol. 2, pp. 837–873. [Online]. Available: http://dx.doi.org/10.1016/S1574-6526(06)80028-3

[4] D. Sabin and R. Weigel, "Product configuration frameworks-a survey," *IEEE intelligent systems*, vol. 13, no. 4, pp. 42–49, 1998.

[5] C. Sinz, A. Kaiser, and W. Küchlin, "Formal methods for the validation of automotive product configuration data," *AI EDAM: Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 17, no. 01, pp. 75–97, 2003.

[6] F. Rossi and A. Sperduti, "Acquiring both constraint and solution preferences in interactive constraint systems," *Constraints*, vol. 9, no. 4, pp. 311–332, 2004.

[7] I. Bliznets and A. Golovnev, "A new algorithm for parameterized MAX-SAT," in *Parameterized and Exact Computation - 7th International Symposium, IPEC 2012, Ljubljana, Slovenia, September 12-14, 2012. Proceedings*, 2012, pp. 37–48. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33293-7_6

[8] M. Patrascu and R. Williams, "On the possibility of faster SAT algorithms," in *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, 2010, pp. 1065–1075. [Online]. Available: http://dx.doi.org/10.1137/1.9781611973075.86

[9] N. Eén and A. Biere, "Effective preprocessing in SAT through variable and clause elimination," in *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings*, 2005, pp. 61–75. [Online]. Available: http://dx.doi.org/10.1007/11499107_5

[10] C. Ansótegui and J. Gabàs, "Solving (weighted) partial maxsat with ILP," in *Proceedings of CPAIOR*, 2013, pp. 403–409. [Online]. Available: http://www.cis.cornell.edu/ics/cpaior2013/pdfs/ansotegui.pdf

[11] E. H. Bareiss, "Sylvester's identity and multistep integer-preserving Gaussian elimination," *Mathematics of Computation*, vol. 22, no. 103, pp. 565–578, Jul. 1968.

[12] J. E. Gentle, "Gaussian elimination." in *Numerical Linear Algebra for Applications in Statistics*. pub-SV:adr: Springer-Verlag, 1998, ch. 3.1, pp. 87–91.

[13] A. Bojańczyk, "Complexity of solving linear systems in different models of computation," *SIAM Journal on Numerical Analysis*, vol. 21, no. 3, pp. 591–603, 1984. [Online]. Available: http://www.jstor.org/stable/2157070

[14] V. I. Solodovnikov, "Upper bounds on the complexity of solving systems of linear equations," *Journal of Soviet Mathematics*, vol. 29, no. 4, pp. 1482–1501, 1985. [Online]. Available: http://dx.doi.org/10.1007/BF02104747

[15] O. H. Ibarra, S. Moran, and R. Hui, "A generalization of the fast lup matrix decomposition algorithm and applications," *Journal of Algorithms*, vol. 3, no. 1, pp. 45 – 56, 1982. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0196677482900074

[16] C. Jeannerod, C. Pernet, and A. Storjohann, "Rank-profile revealing gaussian elimination and the CUP matrix decomposition," *CoRR*, vol. abs/1112.5717, 2011. [Online]. Available: http://arxiv.org/abs/1112.5717