Master Thesis

Modeling selected computational problems as SAT-CNF and analyzing structural properties of obtained formulas

Michal Mrowczyk

31^{st} of March 2016

AGH University of Science and Technology
Faculty for Computer Science, Electronics and Telecommunication

Supervisor
Prof. Dr. Piotr Faliszewski

Referees
Prof. Dr. aaa bbb
Prof. Dr. xxx yyy

Date of the graduation (optional)
xx.yy.zzzz

"Let your plans be dark and impenetrable as night, and when you move, fall like a thunderbolt." — Sun Tzu, The Art of War

Contents

Abstract

The boolean satisfiability was the first computational problem to be proven NP complete.

The proof of this fact was established independently by Stephen Cook and Leonid Levin over 40 years ago.

Since then numerous problems were shown to be NP complete.

Nevertheless, boolean satisfiability (SAT) arguably still has remained the most fundamental NP complete problem out there.

It is possible to convert all problems in NP to SAT by using polynomial time reductions.

In this thesis I provide step by step description of reduction from OWA-Winner problem (to be precise it's decision version) to SAT-CNF.

In order to do this I investigate known techniques of reducing Integer Factorization to SAT-CNF and encoding boolean cardinality constraints.

Having reduced both Integer Factorization and OWA-Winner problems to SAT-CNF I consider experimental ways of exploring the structure of obtained boolean formulae instances.

1. Introduction

1.1. The Boolean Satisfiability and CNF

The Boolean Satisfiability (SAT) is a decision problem [1]

It is a problem stemming from mathematical logic and asking whether given boolean formula F has a satisfying assignment.

Satisfying assignment simply means an assignment that evaluates to $TRUE$

SAT was the very first problem to be proven NP-complete [Coo71] and remains one of the most frequently studied problems in computational complexity theory.

Although finding satisfying truth assignments or proving unsatisfability seems to be hard in general there are tools - solvers (PicoSAT, MiniSat, Glucose, Lingeling, etc.) that can deal with really large instances in practice.

Solving SAT is not only a theoretical challenge. There a lot of practical applications that can be modeled using boolean functions.

Examples of such problems in electronic design automation (EDA) include formal equivalence checking, model checking, formal verification of pipelined microprocessors [REBV99], automatic test pattern generation [Lar], routing of FPGAs [GJNS02], planning [Kau], and scheduling [HZS] problems.

In this section we will define notions needed to understand SAT problem.

Definition 1. Boolean formula F is one of following:

- b - boolean variable
- (F_1) - formula F_1 in parentheses
- $\overline{F_1}$- negation of formula F_1
- $F_1 \wedge F_2$- conjunction of formulas F_1 and F_2
- $F_1 \vee F_2$- disjunction of formulas F_1 and F_2
- $F_1 \Rightarrow F_2$- implication (F_1 implies F_2)
- $F_1 \Leftrightarrow F_2$- equivalence of F_1 and F_2

[1] Decision problem is a problem with YES/NO answer. In formal languages theory such a problem can be viewed as a formal language containing strings (problem instances) for which the answer is YES.

The definition above is stating a formal grammar used to generate the language of valid boolean formulas.

It is important to mention the precedence of operators (from highest to lowest priority):

- () - parentheses have the highest priority
- $\overline{x}$ - negation (of x)
- $\wedge$ - conjunction
- $\vee$ - disjunction
- $\Rightarrow$ - implication
- $\Leftrightarrow$ - equivalence

Definition 2. Truth assignment is a function ψ assigning a truth value to every variable in a formula F (set of variables is denoted as $vars(F)$): $\psi : vars(F) \rightarrow \{TRUE, FALSE\}$

Definition 3. Valuation Let ψ be a truth assignment to variables of F. We can define $\Psi : \{F|F\,is\,a\,boolean\,formula\} \times \{\psi|\psi\,is\,a\,truth\,assignment\,to\,F\} \rightarrow \{TRUE, FALSE\}$ (valuation of F under assignment ψ) in a following recursive way:

- $\Psi(b, \psi) = \psi(b)$ (a valuation of formula consisting of a single boolean variable is simply a truth assignment to this variable)
- $\Psi((F_1), \psi) = \Psi(F_1, \psi)$ (parentheses does not affect valuation)
- $\Psi(\overline{F_1}, \psi) = \begin{cases} TRUE & \Psi(F_1, \psi) = FALSE \\ FALSE & o/w \end{cases}$
- $\Psi(F_1 \wedge F_2, \psi) = \begin{cases} TRUE & \Psi(F_1, \psi) = TRUE\,and\,\Psi(F_2, \psi) = TRUE \\ FALSE & o/w \end{cases}$
- $\Psi(F_1 \vee F_2, \psi) = \begin{cases} TRUE & \Psi(F_1, \psi) = TRUE\,or\,\Psi(F_2, \psi) = TRUE \\ FALSE & o/w \end{cases}$
- $\Psi(F_1 \Rightarrow F_2, \psi) = \Psi(\overline{F_1} \vee F_2)$
- $\Psi(F_1 \Leftrightarrow F_2, \psi) = \Psi((F_1 \Rightarrow F_2) \wedge (F_2 \Rightarrow F_1), \psi)$

Definition 4. Satisfiability Let F be a boolean formula and Ψ be a valuation function. We will call F to be satisfiable (SAT) iff there exists a satisfying assignment ψ such that: $\Psi(F, \psi) = TRUE$. If a formula is not satisfiable then we call it unsatisfiable ($UNSAT$)

Example 5. Consider a following boolean formula: $F \equiv x_1 \wedge (\overline{x_1} \vee x_2)$. Formula F is clearly satisfiable because $\Psi(x_1 \wedge (\overline{x_1} \vee x_2), \{\psi(x_1) = TRUE, \psi(x_2) = TRUE\}) = TRUE$. In other words assignment $x_1 = TRUE$ and $x_2 = TRUE$ is a satisfying assignment.

Example 6. Consider a following boolean formula: $F \equiv (x_1 \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2})$. It is easy to check that this formua is unsatisfiable because under all possible truth assignments it valuates to $FALSE$

Definition 7. Conjunctive Normal Form (CNF) is a special form in which we can write boolean formulas. We say that formula F is written in CNF if:

1. F is a conjuncion of clauses i.e. $F \equiv \bigwedge_{i=1}^{m} c_i$
2. Every clause c in F is a disjunction of literals: $c \equiv \bigvee_{i=1}^{|c|} l_i$
3. Literal l is either a boolean variable or it's negation: $l = b$ or $l = \overline{b}$

Example 8. $F \equiv (x_1 \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2})$ is in a CNF. The set of clauses is: $\{(x_1 \vee x_2), (x_1 \vee \overline{x_2}), (\overline{x_1} \vee x_2), (\overline{x_1} \vee \overline{x_2})\}$. The set of literals: $\{x_1, \overline{x_1}, x_2, \overline{x_2}\}$

Remark 9. Every boolean formula can be transformed into CNF efficiently. One way of doing it is to employ so-called Tseytin transformation [Tse68], which can be summarized in following steps:

- Generate parsing (derivation) tree for a boolean formula F based on boolean formulas grammar definition 1
- For every internal node in a generated tree introduce a boolean variable b and add clause(s) assuring that it is logically equivalent to subformula derived from it's children. For instance consider a $F_1 \rightarrow F_2 \vee F_3$ (meaning: F_1 is a parent, $F_2, \vee, F_3$ are children in a derivation tree). Recursively applying Tseytin transformation on F_1 introduces variables: f_2 for F_2 and f_3 for F_3. When introducing variable f_3 to represent F_3 we have to add following logical equivalence constraint: $f_1 \Leftrightarrow (f_2 \vee f_3)$ which can be written in CNF as: $(\overline{f_1} \vee f_2 \vee f_3) \wedge (\overline{f_2} \vee f_1) \wedge (\overline{f_3} \vee f_1)$
- For the root node we need to assure that variable representing it r is set to $TRUE$. It is enough to add (r) clause to express this constraint.

Formula generated in above steps is **equisatisfiable** (satisfiable iff original formula is satisfiable) to original formula.

1.2. The OWA-Winner problem

In this section we provide a brief introduction and statement of OWA-Winner problem.[2]

We consider Integer Linear Programming (ILP) formulation of this problem.

The OWA-Winner problem was originally introduced in [PF] and is related to voting and elections. The formal definition is presented below.

[2]OWA stands for Ordered Weighted Average

Definition 10. In the OWA-Winner problem we are given a set $N = [n]$ of agents, a set $A = \{a_1, ..., a_m\}$ of items, a collection of agent's utilities $(u_i, a_j)_{i\in[n],a_j\in A}$, a positive integer $K(K \leq m)$, and a $K - number\, OWA\, \alpha^{(K)}$. The task is to compute a subset $W = \{w_1, ..., w_K\}$ of A such that $u_{ut}^{\alpha^K}(W) = \sum_{i=1}^{n} \alpha^{(K)}(u_{i,w_1}, ..., u_{i,w_K})$ is maximal.

ILP formulation of OWA-Winner.

Theorem 11. *OWA-Winner problem can be stated as a following integer linear program:*

$$
\begin{aligned}
&maximize \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{K} \alpha_k u_{i,a_j} x_{i,j,k} \\
&subject\ to: \\
&(a): \sum_{i=1}^{m} y_i = K \\
&(b): x_{i,j,k} \leq y_j && ,i \in [n]; j \in [m]; k \in [K] \\
&(c): \sum_{j=1}^{m} x_{i,j,k} = 1 && ,i \in [n]; k \in [K] \\
&(d): \sum_{k=1}^{K} x_{i,j,k} \leq 1 && ,i \in [n]; j \in [m] \\
&(e): \sum_{j=1}^{m} u_{i,a_j} x_{i,j,k} \geq \sum_{j=1}^{m} u_{i,a_j} x_{i,j,(k+1)} && ,i \in [n]; k \in [K-1] \\
&(f): x_{i,j,k} \in \{0,1\} && ,i \in [n]; j \in [m]; k \in [K] \\
&(g): y_j \in \{0,1\} && ,j \in [m]
\end{aligned}
$$

Proof. Let's define the meaning of variables used in ILP formulation above:

$[n]$ - set of agents, $A = \{a_1, ..., a_m\}$ - set of items, $\alpha = \{\alpha_1, ..., \alpha_k\}$ - OWA vector

u_{i,a_j}- utility that agent i derives from item a_j

$$x_{i,j,k} = \begin{cases} 1 & for\, agent\, i\, item\, a_j\, is\, k-th\, most\, preferred\, from\, items\ \textbf{in\ a\ solution} \\ 0 & o/w \end{cases}$$

$$y_j = \begin{cases} 1 & item\, j\, is\, taken\, in\, a\, solution \\ 0 & o/w \end{cases}$$

By maximizing: $\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{K} \alpha_k u_{i,a_j} x_{i,j,k}$ we maximize a total sum of weighted utilities that agents derives from items . This is consistent with OWA-Winner

problem statement. Below we clarify why conditions (a)-(g) are necessary in ILP formulation:

(a) - This condition states that exactly K items are chosen in a solution.

(b) - If item a_j is not chosen in a solution then there should be no agent i for whom this item appears on $k-th$ position from items appearing in a solution. This constraint enforces that x and y are mutually consistent with each other.

(c) - For agent i there is exactly one item on $k-th$ most preferred place from items appearing in a solution.

(d) - For agent i and item a_j we demand that agent i perceives item a_j on at most one place from K taken into solution. Note that agent i may not perceive item a_j among his/her list of K most preferred items (but still item a_j might have been taken into solution).

(e) - For agent i utility derived from item appearing on $k-th$ position in a solution is not smaller than utility derived from item appearing on $k+1-th$ position in a solution.

(f) - $x_{i,j,k}$ is a binary variable for $i \in [n]; j \in [m]; k \in [K]$

(g) - y_j is a binary variable for $j \in [m]$ □

2. Reducing selected computational problems to SAT-CNF

2.1. Basic notions and definitions

Below we introduce vocabulary used in the following sections.

Most of the terms should be familiar and self-explanatory.

Definition 12. Boolean variable x - variable taking values from $\{0, 1\}$ (being either FALSE or TRUE)

Definition 13. Sequence (of boolean variables) $< x_1, x_2, x_3, .., x_n >$- ordered collection of boolean variables of fixed size

Definition 14. Length of a sequence - number of boolean variables associated with a sequence. $length(< x_1, x_2, .., x_n >) = n$

Sequences of length n can be used to represent $n - bit$ integers. Each variable in a sequence is representing exactly one bit of an integer.

Remark 15. When using sequence $X =< x_1, x_2, .., x_n >$ to represent integers we stick to the convention that x_1 corresponds to the least significant bit and x_n corresponds to the most significant bit.

2.2. Reducing Integer Factorization to SAT-CNF

Since Integer Factorization problem belongs to class NP[1] there is a way to reduce it to SAT-CNF in polynomial time.

Arguably, the most direct way of doing it is to encode multiplication circuit as a SAT-CNF formula.

One of such encodings is available in: [Sre04]

Following subsections contain descriptions of various constraints. The main goal of each subsection is to establish either a CNF encoding for a given constraint or an algorithm producing such an encoding.

[1] It is easy to verify given n and numbers p and q if $n = pq$ It is enough to compute product pq which can be easily done in polynomial time and compare it to n

2.2.1. Encoding equality of sequences $X = Y$

To represent equality between sequences X and Y it suffices to encode 'variable-wise' equality

$X =< x_1, x_2, .., x_n >$

$Y =< y_1, y_2, .., y_n >$

$X = Y$:

$$\bigwedge_{i=1}^{n} (x_i \Leftrightarrow y_i)$$

$$\bigwedge_{i=1}^{n} ((\overline{x_i} \vee y_i) \wedge (x_i \vee \overline{y_i}))$$

(in conjunctive normal form)

2.2.2. Encoding not equality between sequence and integer $X \neq I$

This type of constraint is especially useful when we want to enforce that some sequence X is **not** equal given integer I. For example we may wish that our factor X (represented by sequence) is not equal 1

For this to hold we need to encode $X \neq 1$ constraint as a SAT-CNF formula (set of clauses)

$X =< x_1, x_2, .., x_n >$

I- integer

$X \neq I$:

$$\bigvee_{i=1}^{n} y_i$$

where: $y_i = x_i$ if $i - th$ bit of I is 0 (If $i - th$ bit of I is 1 then $y_i = \overline{x_i}$)

Example 16. Let $I = 13$ and $X =< x_1, x_2, x_3, x_4 >$ Constraint $X \neq I$ can be encoded as $(\overline{x_1} \vee x_2 \vee \overline{x_3} \vee \overline{x_4})$

2.2.3. Encoding shift equality constraint $Y = 2^i X$

This constraint is basically stating that after shifting X by i positions to the left we obtain Y

$X =< x_1, x_2, .., x_n >$

$Y =< y_1, y_2, .., y_n >$

$Y = 2^i X$:

$$(\bigwedge_{j=1}^{i} \overline{y_j}) \wedge \bigwedge_{j=i+1}^{n} ((y_j \vee \overline{x_{j-i}}) \wedge (\overline{y_j} \vee x_{j-i}))$$

2.2.4. Encoding left variable-wise multiplication $bX = Y$

b - boolean variable

$X =< x_1, x_2, .., x_n >$

$Y =< y_1, y_2, .., y_n >$

$bX = Y \iff (b \wedge x_1, b \wedge x_2, .., b \wedge x_n) = (y_1, y_2, .., y_n)$ (meaning of left variable-wise multiplication)

$bX = Y$:

$$\bigwedge_{i=1}^{n} ((b \vee \overline{y_i}) \wedge (x_i \vee \overline{y_i}) \wedge (y_i \vee \overline{b} \vee \overline{x_i}))$$

2.2.5. Encoding addition $X + Y = Z$

In order to encode addition constraint between sequences we need to introduce additional sequence C representing carry bits.

$X =< x_0, x_1, .., x_{n-1} >$

$Y =< y_0, y_1, ..y_{n-1} >$

$Z =< z_0, z_1, .., z_{n-1} >$

$C =< c_0, c_1, .., c_n >$(Please note that carry sequence has length of $n + 1$)

$X + Y = Z$ (with carry C):

$$
\begin{aligned}
&(\overline{c_0}) \wedge (\overline{c_n}) \\
&\wedge \bigwedge_{i=1}^{n} ((\overline{c_i} \vee x_{i-1} \vee c_{i-1}) \wedge (\overline{c_i} \vee x_{i-1} \vee y_{i-1}) \wedge (\overline{c_i} \vee y_{i-1} \vee c_{i-1}) \\
&\wedge (c_i \vee \overline{x_{i-1}} \vee \overline{c_{i-1}}) \wedge (c_i \vee \overline{x_{i-1}} \vee \overline{y_{i-1}}) \wedge (c_i \vee \overline{y_{i-1}} \vee \overline{c_{i-1}})) \\
&\wedge \bigwedge_{i=0}^{n-1} ((z_i \vee y_i \vee x_i \vee \overline{c_i}) \wedge (z_i \vee y_i \vee \overline{x_i} \vee c_i) \wedge (z_i \vee \overline{y_i} \vee x_i \vee c_i) \wedge (z_i \vee \overline{y_i} \vee \overline{x_i} \vee \overline{c_i}) \\
&\wedge (\overline{z_i} \vee y_i \vee x_i \vee c_i) \wedge (\overline{z_i} \vee y_i \vee \overline{x_i} \vee \overline{c_i}) \wedge (\overline{z_i} \vee \overline{y_i} \vee x_i \vee \overline{c_i}) \wedge (\overline{z_i} \vee \overline{y_i} \vee \overline{x_i} \vee c_i))
\end{aligned}
$$

2.2.6. Encoding multiplication $PQ = N$

Formula for computing product of two numbers $n = pq$ can be expressed as:

$$pq = q_0 p + q_1 2p + q_2 2^2 p + ... + q_{k-1} 2^{k-1} p$$

Careful reader can notice that formula above is basically a **sum of shift multiplications** for which we have already shown appropriate encodings. We need a lot of additional variables (and sequences) to construct CNF encoding of $PQ = N$.

Let $ln = length(N)$ and $lq = length(Q)$

Below is a summary of additional sequences used to construct CNF encoding of $PQ = N$:

- S- array of lq sequences of length ln (i.e. $S = [S_0, S_1, .., S_{lq-1}]$ and $length(S_i) = ln$)
- C- array of $lq - 1$ sequences of length $ln + 1$
- M- array of lq sequences of length ln
- R- array of lq sequences of length ln

Instead of writing the encoding down using explicit CNF formula we take approach of providing an algorithm (in form of pseudocode) representing the steps necessary to generate such an encoding: Algorithm 2.1

Each step represent constraint(s) that has to be added.

Last two for loops are there just to fix some variables in P and Q in order to explicitely decrease search space.

Algorithm 2.1 Generating CNF for $PQ = N$

$S_0 = P$
for $i = 1$ **to** $lq - 1$ **do**
  $S_i = 2S_{i-1}$
end for
for $i = 0$ **to** $lq - 1$ **do**
  $M_i = Q_i S_i$
end for
$R_0 = M_0$
for $i = 1$ **to** $lq - 1$ **do**
  $R_{i-1} + M_i = R_i$ // carry=C_{i-1}
end for
$R_{lq-1} = N$
for each pair $(i, j) \in [0, 1, .., ln - 1] \times [0, 1, .., lq - 1]$ **do**
  if $i + j \geq ln$ **then**
    $(\bar{P}_i \vee \bar{Q}_j)$ // to ensure that multiplication result does not have more bits than N
  end if
end for
for $i = 0$ **to** $lq - 1$ **do**
  if $i > \frac{lq-1}{2}$ **then**
    $(\bar{Q}_i)$ // Limiting number of significant bits in Q
  end if
end for

2.2.7. Encoding multiplication $PQ = N, 1 < P < N, 1 < Q < N$

The final step to reduce Integer Factorization to SAT-CNF is to enforce that both P and Q represent nontrivial factors i.e. $1 < P < N, 1 < Q < N$

There are multiple ways to do it, but the most straightforward is to demand:

$$Q \neq 1$$

Up to this point we have shown all steps necessary to convert arbitrary Integer Factorization problem instance to boolean formula in CNF form.

If a formula created in such fashion turns out to be $UNSAT$ then we can be sure that there are no nontrivial factors to original Integer Factorization problem instance (number is prime).

If there is a satisfying assignment then we can recover factors by looking at part of the satisfying assignment that corresponds to P and Q

2.3. Reducing OWA-Winner to SAT-CNF

In this section we develop a machinery needed to reduce OWA-Winner[2] problem to SAT-CNF. To do this we will consider ILP formulation of OWA-Winner problem presented in Chapter 1

2.3.1. Encoding inequality between sequences $X \leq Y$

$X =< x_1, x_2, .., x_n >$ - x_1 is the least significant digit, x_n is the most significant digit

$Y =< y_1, y_2, .., y_n >$ - y_1 is the least significant digit, y_n is the most significant digit

$X \leq Y \iff (x_n < y_n) \vee (x_n = y_n \wedge (x_{n-1} < y_{n-1} \vee ...(x_1 = y_1 \vee (x_1 < y_1))))$

Below (Algorithm 2.2) we provide an algorithm which constructs a boolean formula for $X \leq Y$:

Algorithm 2.2 Encoding $X \leq Y$

1: $f \leftarrow (\bar{x_1} \wedge y_1) \vee ((\bar{x_1} \vee y_1) \wedge (x_1 \vee \bar{y_1}))$
2: **for** $i = 2$ **to** n **do**
3: $\quad f \leftarrow (\bar{x_i} \wedge y_i) \vee (((\bar{x_i} \vee y_i) \wedge (x_i \vee \bar{y_i})) \wedge f)$
4: **end for**
5: **return** f

Formula generated using algorithm Algorithm 2.2 is not in CNF. To convert it to CNF in efficient manners we take advantage of Tseytin transformation [Tse68]

2.3.2. Encoding inequality between sequence and integer $X \leq I$

$X =< x_1, x_2, .., x_n >$ - x_1 is the least significant digit, x_n is the most significant digit

$I =< i_1, i_2, .., i_n >$ - binary encoding of integer I. $i_1, i_2, .., i_n$ - bits

$X \leq I$ is a special case of $X \leq Y$. Because of that we can obtain more efficient encoding of $X \leq I$

Formula experssing $X \leq I$ can be generated using Algorithm 2.3. We can employ Tseytin transformation to convert it to CNF.

2.3.3. Encoding boolean cardinality constraints

In this section we will consider different boolean cardinality constraints and their encodings. We will show an efficient implementation of those constraints based on the work in [Sin]

[2]In fact OWA-Winner is an optimization problem, so we will consider it's decision version.

Algorithm 2.3 Encoding $X \leq I$

```
1: if i_1 = 0 then
2:    f ← x̄_1
3: else if i_1 = 1 then
4:    f ← x_1 ∨ x̄_1
5: end if
6: for j = 2 to n do
7:    if i_j = 0 then
8:       f ← x̄_j ∧ f
9:    else if i_j = 1 then
10:      f ← x̄_j ∨ (x_j ∧ f)
11:   end if
12: end for
13: return f
```

Definition 17. Let $X = \{x_1, x_2, .., x_n\}$ be a set of boolean variables. We define at most k of constraint $_{\leq k}(X)$ by demanding that at most k variables from X are set to $TRUE$

Example 18. Let $X = \{x_1, x_2, x_3\}$. At most 1 of X constraint $_{\leq 1}(\{x_1, x_2, x_3\})$ can be represented as a following boolean formula: $(\overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3}) \vee (x_1 \wedge \overline{x_2} \wedge \overline{x_3}) \vee (\overline{x_1} \wedge x_2 \wedge \overline{x_3}) \vee (\overline{x_1} \wedge \overline{x_2} \wedge x_3)$. It enforces that there are no 2 variables set to $TRUE$ at the same time.

Definition 19. Let $X = \{x_1, x_2, .., x_n\}$ be a set of boolean variables. We define at least k of constraint $_{\geq k}(X)$ by demanding that at least k variables from X are set to $TRUE$

Definition 20. Let $X = \{x_1, x_2, .., x_n\}$ be a set of boolean variables. We define exactly k of constraint $_{=k}(X)$ by demanding that exactly k variables from X are set to $TRUE$

Remark 21. Let $k \in \mathbb{N}$ and X be a set of propositional (boolean) variables. Let $CNF(_{\leq k}(X))$ be a CNF encoding of $_{\leq k}(X)$, $CNF(_{\geq k}(X))$ be a CNF encoding of $_{\geq k}(X)$ and $CNF(_{=k}(X))$ be a CNF encoding of $_{=k}(X)$. The following holds:

$$CNF(_{=k}(X)) = CNF(_{\leq k}(X)) \wedge CNF(_{\geq k}(X))$$

Theorem 22. *Encoding* $LT^{n,k}_{SEQ}$ *expressing* $_{\leq k}(\{x_1, x_2, .., x_n\})$ $n > 1, k > 0$ *can be stated as follows:*

$(\overline{x_1} \vee s_{1,1})$	
$(\overline{s_{1,j}})$	$1 < j \leq k$
$(\overline{x_i} \vee s_{i,1})$	$1 < i < n$
$(\overline{s_{i-1,1}} \vee s_{i,1})$	$1 < i < n$
$(\overline{x_i} \vee \overline{s_{i-1,j-1}} \vee s_{i,j})$	$1 < i < n, 1 < j \leq k$
$(\overline{s_{i-1,j}} \vee s_{i,j})$	$1 < i < n, 1 < j \leq k$
$(\overline{x_i} \vee \overline{s_{i-1,k}})$	$1 < i < n$
$(\overline{x_n} \vee \overline{s_{n-1,k}})$	

Proof of theorem 22 is available in [Sin]

Corollary 23. *Encoding $GT^{n,k}_{SEQ}$ expressing $_{\geq k}(\{x_1, x_2, .., x_n\})$ $n > 1, k > 0$ can be stated as follows:*

$(x_1 \vee \overline{s_{1,1}})$	
$(\overline{s_{1,j}})$	$1 < j \leq k$
$(\overline{s_{i,j}}, s_{i-1,j-1})$	$1 < i \leq n, 1 < j \leq k$
$(\overline{s_{i,j}}, s_{i-1,j}, x_i)$	$1 < i \leq n, 1 < j \leq k$
$(\overline{s_{i,1}}, s_{i-1,1}, x_i)$	$1 < i \leq n$
$(s_{n,k})$	

Proof: Theorem 23 is simply obtained by applying the same technique used to construct 22

2.3.4. Encoding decision version of OWA-Winner problem

Let's state decision version of OWA-Winner problem based on ILP formulation from Chapter 1

Definition 24. Decision version of OWA-Winner problem reduces to checking feasibility of following integer linear program:

$$
\begin{aligned}
&(a): \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{K} \alpha_k u_{i,a_j} x_{i,j,k} \geq L && L \in \mathbb{N}\\
&(b): \sum_{i=1}^{m} y_i = K\\
&(c): x_{i,j,k} \leq y_j && ,i \in [n]; j \in [m]; k \in [K]\\
&(d): \sum_{j=1}^{m} x_{i,j,k} = 1 && ,i \in [n]; k \in [K]\\
&(e): \sum_{k=1}^{K} x_{i,j,k} \leq 1 && ,i \in [n]; j \in [m]\\
&(f): \sum_{j=1}^{m} u_{i,a_j} x_{i,j,k} \geq \sum_{j=1}^{m} u_{i,a_j} x_{i,j,(k+1)} && ,i \in [n]; k \in [K-1]\\
&(g): x_{i,j,k} \in \{0,1\} && ,i \in [n]; j \in [m]; k \in [K]\\
&(h): y_j \in \{0,1\} && ,j \in [m]
\end{aligned}
$$

Having stated what we mean by decison version of OWA-Winner problem we can finally present a way of encoding arbitrary OWA-Winner problem instances as a SAT-CNF formula.

Theorem 25. *Encoding Decision OWA-Winner problem instances as SAT-CNF formulas*

$$
\begin{aligned}
&(a): \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{K} \alpha_k u_{i,a_j} x_{i,j,k} \geq L && L \in \mathbb{N}\\
&(b): :_{=K} (\{y_j | j \in [m]\})\\
&(c): (\overline{x_{i,j,k}}, y_j) && ,i \in [n]; j \in [m]; k \in [K]\\
&(d): :_{=1} (\{x_{i,j,k} | j \in [m]\}) && ,i \in [n]; k \in [K]\\
&(e): :_{\leq 1} (\{x_{i,j,k} | k \in [K]\}) && ,i \in [n]; j \in [m]\\
&(f): \sum_{j=1}^{m} u_{i,a_j} x_{i,j,k} \geq \sum_{j=1}^{m} u_{i,a_j} x_{i,j,(k+1)} && ,i \in [n]; k \in [K-1]\\
&(g): x_{i,j,k} \in \{0,1\} && ,i \in [n]; j \in [m]; k \in [K]\\
&(h): y_j \in \{0,1\} && ,j \in [m]
\end{aligned}
$$

Proof. We need to show that constraints (a) - (h) are expressible using SAT-CNF encodings constructed so far. (g) and (h) are clearly just declaring sets of propositional variables: x and y, and therefore producing no clauses in a CNF encoding.

Constraint (a) is simply an inequality between sequence constructed from **sum of products** and integer ($S \geq L$ and $S = \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{k=1}^{K} \alpha_k u_{i,a_j} x_{i,j,k}$) so in spite of being quite costly (in terms of number of variables and clauses) it is expressible in SAT-CNF format.

Similarly for (f) we can write $S_1 \geq S_2$ where S_1 is a sequence ($S_1 = \sum_{j=1}^{m} u_{i,a_j} x_{i,j,k}$) and S_2 is a sequence ($S_2 = \sum_{j=1}^{m} u_{i,a_j} x_{i,j,(k+1)}$) . (c) is a simple clause logically equivalent to: $(x_{i,j,k} \Rightarrow y_j)$, which behaves as $x_{i,j,k} \leq y_j$. (b), (d), (e) are all boolean cardinality constraints for which we have already shown one efficient encoding. □

(a) and (f) are the most costly constraints in the model. In the next section we will look at a slightly restricted version of decision OWA-Winner problem.

2.3.5. Encoding decision version of Binary OWA-Winner problem

As we saw in the previous subsection it is possible to convert any Decision OWA-Winner problem instance to SAT-CNF formula. It is prohibitevely expensive to encode constraints (a) and (f) (requiring lots of sequence multiplications). In this subsection we will present more restricted yet still computationally demanding version of Decision OWA-Winner problem.

Definition 26. Decision version of Binary OWA-Winner problem is obtained from Decision version of OWA-Winner problem by:

- Forcing α - OWA vector and u - derived utility to be binary ($\alpha_i \in \{0,1\}, u_{i,a_j} \in \{0,1\}$)
- Removing following constraint: $(f) : \sum_{j=1}^{m} u_{i,a_j} x_{i,j,k} \geq \sum_{j=1}^{m} u_{i,a_j} x_{i,j,(k+1)}, i \in [n]; k \in [K-1]$

SAT-CNF encoding of Decision Binary OWA-Winner problem follows:

Theorem 27. *Encoding Decision Binary OWA-Winner problem instances as SAT-CNF formulas*

$$
\begin{aligned}
&(a) :_{\geq L} (\{x_{i,j,k} | i \in [n], j \in [m], k \in [K], \alpha_k u_{i,a_j} > 0\}) && \\
&(b) :_{=K} (\{y_j | j \in [m]\}) && \\
&(c) : (\overline{x_{i,j,k}}, y_j) && , i \in [n]; j \in [m]; k \in [K] \\
&(d) :_{=1} (\{x_{i,j,k} | j \in [m]\}) && , i \in [n]; k \in [K] \\
&(e) :_{\leq 1} (\{x_{i,j,k} | k \in [K]\}) && , i \in [n]; j \in [m] \\
&(f) : x_{i,j,k} \in \{0,1\} && , i \in [n]; j \in [m]; k \in [K] \\
&(g) : y_j \in \{0,1\} && , j \in [m]
\end{aligned}
$$

Proof. We remove $\sum_{j=1}^{m} u_{i,a_j} x_{i,j,k} \geq \sum_{j=1}^{m} u_{i,a_j} x_{i,j,(k+1)}, i \in [n]; k \in [K-1]$ constraints. We can easily see that $\alpha_k u_{i,a_j}$ has to be either 0 or 1 (α and u are binary). This fact allows us to transform $\sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{k=1}^{K} \alpha_k u_{i,a_j} x_{i,j,k} \geq L$ into ${}_{\geq L}(\{x_{i,j,k} | i \in [n], j \in [m], k \in [K], \alpha_k u_{i,a_j} > 0\})$ □

3. Experimental analysis of structure of obtained formulas

3.1. Structure of Integer Factorization formulas

3.2. Structure of OWA-Winner formulas

3.3. Structure of randomly-generated hard formulas

Acknowledgments

Firstly I would like to thank my family for all the love and support.

I wish to thank my supervisor Prof. Dr. Piotr Faliszewski for his suggestions and advices.

Last but not least, I am really grateful to all the people who inspired me including colleagues and teachers.

A. Appendix

A.1. Overview

bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla roughness parameter R_a bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla, see [?].

A.2. The next section

Bibliography

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures, 1971.

[GJNS02] R. A. Gi-Joon Nam; Sakallah, K. A.; Rutenbar. A new fpga detailed routing approach via search-based boolean satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 21 (6): 674*, 2002.

[HZS] Dapeng Li Hantao Zhang and Haiou Shen. A sat based scheduler for tournament schedules. `http://www.satisfiability.org/SAT04/programme/74.pdf`.

[Kau] Henry Kautz. Deconstructing planning as satisfiability. `https://www.cs.washington.edu/ai/planning/papers/AAAI0609KautzA.pdf`.

[Lar] Tracy Larrabee. Test pattern generation using boolean satisfiability. `https://users.soe.ucsc.edu/~larrabee/ce224/tcad.sat.pdf`.

[PF] Jerome Lang Piotr Faliszewski, Piotr Skowron. Finding a collective set of items: From proportional multirepresentation to group recommendation.

[REBV99] S. M. German R. E. Bryant and M. N. Velev. Microprocessor verification using efficient decision procedures for a logic of equality with uninterpreted functions. volume Automated Reasoning with Analytic Tableaux and Related Methods, pages 1–13, 1999.

[Sin] Carsten Sinz. Towards an optimal cnf encoding of boolean cardinality constraints. `http://www.carstensinz.de/papers/CP-2005.pdf`.

[Sre04] Mateusz Srebrny. Factorization with sat - classical propositional calculus as a programming environment. `http://www.mimuw.edu.pl/~mati/fsat-20040420.pdf`, April 2004.

[Tse68] G. Tseytin. On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic*, pages 115–125, 1968.

Nomenclature

R_a	arithmetic average roughness