# Modeling selected computational problems as SAT-CNF and analyzing (some) structural properties of obtained formulas

Michał Mrowczyk

Department of Computer Science
AGH Kraków

M.Sc. Defense, 2017

## Goals

- Reduce selected computational problems i.e. the Integer-Factorization and the OWA-Winner to SAT-CNF
- Investigate properties of the obtained formulas (using SAT solvers such as PicoSAT to solve these formulas)
  - running time of the solver
  - clauses to variables ratio
  - ...

## SAT Decision Problem

- Given a boolean formula $F$ decide (answer yes/no) whether there exists a satisfying assignment of True/False to variables (so that the formula evaluates to True).
- $x_1 \wedge (\overline{x_1} \vee x_2)$
- Setting $x_1 = 1$ and $x_2 = 1$ makes the formula above to evaluate to 1 (True), so we call it satisfiable (SAT)
- What about this one: $(\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge (x_1 \vee x_2)$ ?
- It turns out that this one always evaluates to 0 (False), so we call it unsatisfiable (UNSAT)

## SAT Decision Problem

- Given a boolean formula $F$ decide (answer yes/no) whether there exists a satisfying assignment of True/False to variables (so that the formula evaluates to True).
- $x_1 \wedge (\overline{x_1} \vee x_2)$
- Setting $x_1 = 1$ and $x_2 = 1$ makes the formula above to evaluate to 1 (True), so we call it satisfiable (SAT)
- What about this one: $(\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge (x_1 \vee x_2)$ ?
- It turns out that this one always evaluates to 0 (False), so we call it unsatisfiable (UNSAT)

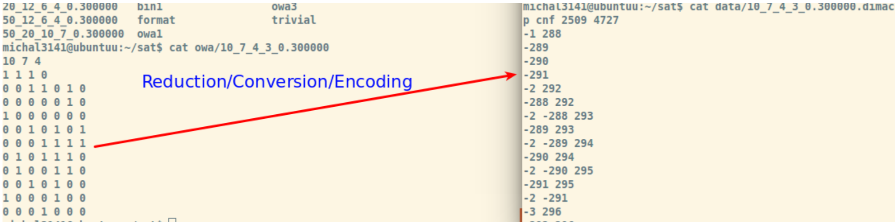
## Pseudo DIMACS Format

- A way of writing (format) boolean formulas
- Example:
  $(x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee x_4) \wedge (\overline{x_3} \vee x_5) \rightarrow (1, -2, 3), (2, 4), (-3, 5)$

$$\begin{array}{lll} 1 & -2 & 3 \\ 2 & 4 & \\ -3 & 5 & \end{array}$$

## Big Picture

- We want to transform the instances of both OWA-Winner and Integer-Factorization problems into the boolean formulas
- ... And investigate properties of these formulas

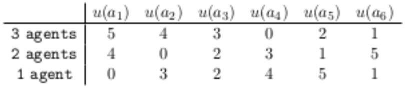


```
20_12_6_4_0.300000   bin1                owa3
50_12_6_4_0.300000   format              trivial
50_20_10_7_0.300000  owa1
michal3141@ubuntuu:~/sat$ cat owa/10_7_4_3_0.300000
10 7 4
1 1 1 0
0 0 1 1 0 1 0
0 0 0 0 0 1 0
1 0 0 0 0 0 0
0 0 1 0 1 0 1
0 0 0 1 1 1 1
0 1 0 1 1 1 0
0 1 0 0 1 1 0
0 0 1 0 1 0 0
1 0 0 0 1 0 0
0 0 0 1 0 0 0
```

```
michal3141@ubuntuu:~/sat$ cat data/10_7_4_3_0.300000.dimac
p cnf 2509 4727
-1 288
-289
-290
-291
-2 292
-288 292
-2 -288 293
-289 293
-2 -289 294
-290 294
-2 -290 295
-291 295
-2 -291
-3 296
```

## OWA-Winner Problem Example

- We want to select $K = 3$ items
- We use the following OWA vector: $\alpha = (2, 1, 0)$
- What is the score of $\{a_1, a_2, a_6\}$ ?



| | $u(a_1)$ | $u(a_2)$ | $u(a_3)$ | $u(a_4)$ | $u(a_5)$ | $u(a_6)$ |
|---|---|---|---|---|---|---|
| 3 agents | 5 | 4 | 3 | 0 | 2 | 1 |
| 2 agents | 4 | 0 | 2 | 3 | 1 | 5 |
| 1 agent | 0 | 3 | 2 | 4 | 5 | 1 |

score $= 3 \cdot (2 \cdot 5 + 1 \cdot 4 + 0 \cdot 1) + 2 \cdot (2 \cdot 5 + 1 \cdot 4 + 0 \cdot 0) + 1 \cdot (2 \cdot 3 + 1 \cdot 1 + 0 \cdot 0) = 42 + 28 + 7 = 77$

## OWA-Winner Problem

- The notation used in this presentation:
  - $\alpha_k$ is an OWA vector
  - $u_{i,a_j}$ is a utility function value that $i$-th voter assigns to the $j$-th item/candidate
  - $x_{i,j,k}$ is 1 if the $i$-th voter views the $j$-th item/candidate on the $k$-th position from items taken into the solution
- The goal of the OWA-Winner problem is to determine a comitee (set of items/candidates) of cardinality $K$ maximizing the following expression:
  - $$\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{K}\alpha_k u_{i,a_j} x_{i,j,k}$$
- We convert the OWA-Winner problem instances to ILP and then to SAT-CNF

## ILP Formulation of the OWA-Winner

[2] Source: Finding a Collective Set of Items: From Proportional Multirepresentation to Group Recommendation

$$\text{maximize} \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{K} \alpha_k u_{i,a_j} x_{i,j,k}$$

subject to :

$$(a): \sum_{i=1}^{m} y_i = K$$

$$(b): x_{i,j,k} \le y_j \qquad ,i \in [n]; j \in [m]; k \in [K]$$

$$(c): \sum_{j=1}^{m} x_{i,j,k} = 1 \qquad ,i \in [n]; k \in [K]$$

$$(d): \sum_{k=1}^{K} x_{i,j,k} \le 1 \qquad ,i \in [n]; j \in [m]$$

$$(e): \sum_{j=1}^{m} u_{i,a_j} x_{i,j,k} \ge \sum_{j=1}^{m} u_{i,a_j} x_{i,j,(k+1)} \qquad ,i \in [n]; k \in [K-1]$$

$$(f): x_{i,j,k} \in \{0,1\} \qquad ,i \in [n]; j \in [m]; k \in [K]$$

$$(g): y_j \in \{0,1\} \qquad ,j \in [m]$$

## SAT Formulation of the OWA-Winner

$$(m): CNF(\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{K}\alpha_k u_{i,a_j} x_{i,j,k} \geq L) \qquad L \in \mathbb{N}$$

$$(a): CNF(=_K(\{y_j | j \in [m]\}))$$

$$(b): (\overline{x_{i,j,k}}, y_j) \qquad , i \in [n]; j \in [m]; k \in [K]$$

$$(c): CNF(=_1(\{x_{i,j,k} | j \in [m]\})) \qquad , i \in [n]; k \in [K]$$

$$(d): CNF(\leq_1(\{x_{i,j,k} | k \in [K]\})) \qquad , i \in [n]; j \in [m]$$

$$(e): CNF(\sum_{j=1}^{m} u_{i,a_j} x_{i,j,k} \geq \sum_{j=1}^{m} u_{i,a_j} x_{i,j,(k+1)}) \qquad , i \in [n]; k \in [K-1]$$

$$(f): x_{i,j,k} \in \{0,1\} \qquad , i \in [n]; j \in [m]; k \in [K]$$

$$(g): y_j \in \{0,1\} \qquad , j \in [m]$$

## SAT Formulation - Binary Utility and OWA Vector

Both the utility function and the OWA vector are binary. In addition to this the OWA vector is nonincreasing.

$(m) : CNF(\geq_L(\{x_{i,j,k} | i \in [n], j \in [m], k \in [K], \alpha_k u_{i,a_j} > 0\}))$

$(a) : CNF(=_K(\{y_j | j \in [m]\}))$

$(b) : (\overline{x_{i,j,k}}, y_j) \quad , i \in [n]; j \in [m]; k \in [K]$

$(c) : CNF(=_1(\{x_{i,j,k} | j \in [m]\})) \quad , i \in [n]; k \in [K]$

$(d) : CNF(\leq_1(\{x_{i,j,k} | k \in [K]\})) \quad , i \in [n]; j \in [m]$

$(f) : x_{i,j,k} \in \{0, 1\} \quad , i \in [n]; j \in [m]; k \in [K]$

$(g) : y_j \in \{0, 1\} \quad , j \in [m]$

## General Results

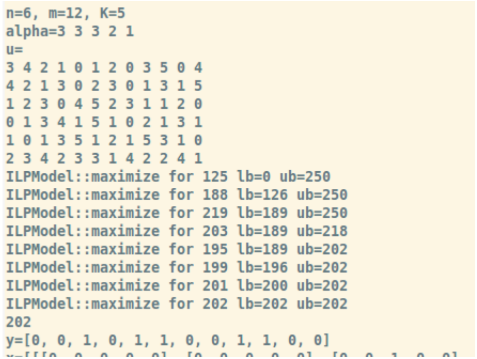
- Tests were conducted by applying the PicoSAT solver to general and binary OWA-Winner models
- Bigger OWA-Winner problem instances can be solved by using the binary OWA model (because the binary OWA model is more efficient, but at the same time more restricting)
- As $K$ gets bigger and bigger, the produced boolean formulas are becoming larger and larger very rapidly

## Example: General OWA-Winner Instance

The optimization run below lasted for about 10 minutes.

p cnf 26199 124996

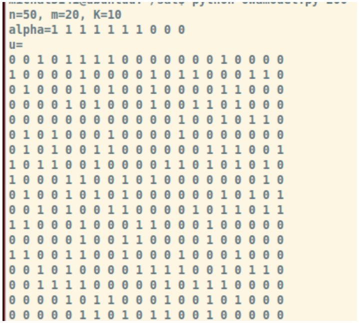
Produced boolean formula is available at: owa3.dimacs



```
n=6, m=12, K=5
alpha=3 3 3 2 1
u=
3 4 2 1 0 1 2 0 3 5 0 4
4 2 1 3 0 2 3 0 1 3 1 5
1 2 3 0 4 5 2 3 1 1 2 0
0 1 3 4 1 5 1 0 2 1 3 1
1 0 1 3 5 1 2 1 5 3 1 0
2 3 4 2 3 3 1 4 2 2 4 1
ILPModel::maximize for 125 lb=0 ub=250
ILPModel::maximize for 188 lb=126 ub=250
ILPModel::maximize for 219 lb=189 ub=250
ILPModel::maximize for 203 lb=189 ub=218
ILPModel::maximize for 195 lb=189 ub=202
ILPModel::maximize for 199 lb=196 ub=202
ILPModel::maximize for 201 lb=200 ub=202
ILPModel::maximize for 202 lb=202 ub=202
202
y=[0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0]
x=[[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 1, 0, 0]
```

## Example: Binary (Approval) OWA-Winner Instance

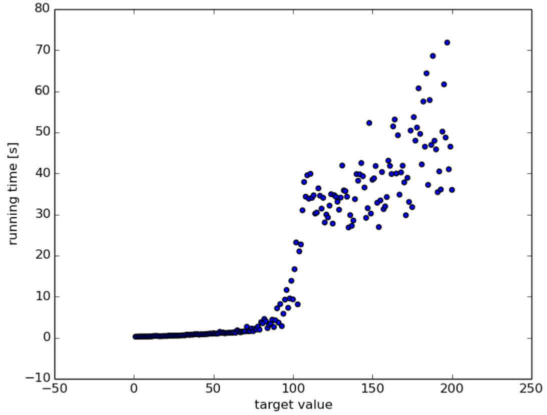
To solve this one below to optimality it took roughly 90 minutes on my machine... (there are $\binom{20}{10} = 184756$ possible comitees to check when using pure brute-force)



```
n=50, m=20, K=10
alpha=1 1 1 1 1 1 1 0 0 0
u=
0 0 1 0 1 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0
1 0 0 0 0 1 0 0 0 0 1 0 1 1 0 0 0 1 1 0
0 1 0 0 0 1 0 1 0 0 1 0 0 0 0 1 1 0 0 0
0 0 0 0 1 0 1 0 0 0 1 0 0 1 1 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 0
0 1 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0
0 1 0 1 0 0 1 1 0 0 0 0 0 0 1 1 1 0 0 1
1 0 1 1 0 0 1 0 0 0 0 1 1 0 1 0 1 0 1 0
1 0 0 0 1 1 0 0 1 0 1 0 0 0 0 0 0 0 1 0
0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 1 0 1
0 0 1 0 1 0 0 1 1 0 0 0 0 1 0 1 1 0 1 1
1 1 0 0 0 1 0 0 0 1 1 0 0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 0 0
1 1 0 0 1 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0
0 0 1 0 1 0 0 0 0 1 1 1 1 0 0 1 0 1 1 0
0 0 1 1 1 1 0 0 0 0 0 1 0 1 1 1 0 0 0 0
0 0 0 0 1 0 1 1 0 0 0 1 0 0 1 0 1 0 0 0
0 0 0 0 0 1 1 0 1 0 1 1 0 0 1 0 0 0 0 0
```

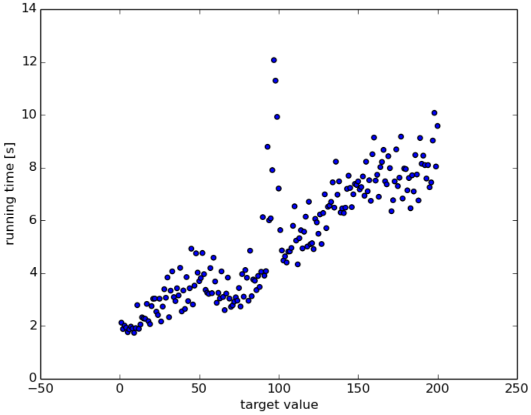
## Running Time vs Solution Quality (target value)

kBestOWAApprovalWinner$(50, 12, 6, \mu, 4, 0.3, v)$; Unsatisfiability for target value=108



## Running Time vs Solution Quality (target value) 2

kBestOWAApprovalWinner$(100, 24, 10, \mu, 1, 0.3, v)$; Unsatisfiability for target value=101

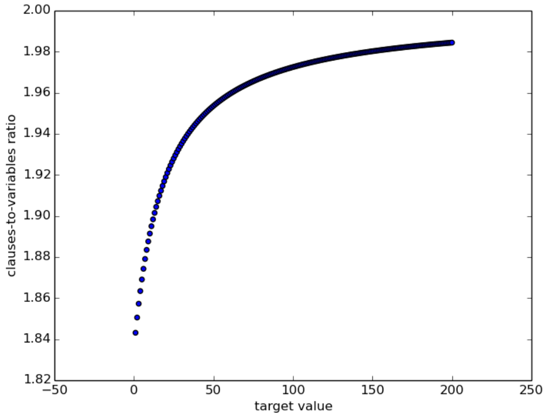


## Clauses to Variables Ratio

- clauses-to-variables-ratio $cv = \frac{\#clauses}{\#variables}$
- Example: $F \equiv (x_1 \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2})$. $cv = 2$
- Randomly generated SAT-CNF formulas with $cv > 4.26$ are mostly UNSAT, but formulas with $cv < 4.26$ are mostly SAT
- Studying $cv$ for various formulas generated for both the OWA-Winner and Integer Factorization instances

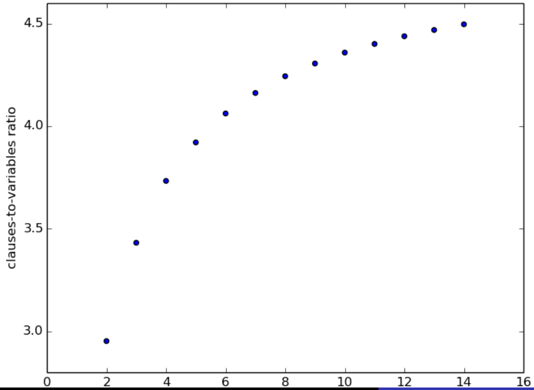
## Clauses to Variables for OWA-Winner Instances

kBestOWAApprovalWinner$(50, 12, 6, \mu, 4, 0.3, v)$



## Clauses to Variables for Integer Factorization Instances

All $n$-bit integers Factorization problems corresponds to the formulas with the same $cv$ ratio $\rightarrow$ we consider how $cv$ varies depending on number of bits. We show it tends to $\frac{39}{8}$.



## Summary

- Reduction from the OWA-Winner to SAT-CNF was presented with some (but not all) quirks
- In addition to the general OWA-Winner, we prepared a more efficient reduction for the OWA-Winner with binary utility and OWA-vector (+ nonincreasing OWA vector)
- Running Time and Clauses to Variables Ratio were considered as measures of hardness for selected SAT-CNF instances

## Thank you

Thank you!

## Boolean Cardinality Constraints

- Given a set of boolean variables $\{x_1, x_2, ..., x_n\}$ we want to ensure that exactly (at least, at most,...) $k \leq n$ of them are set to True
- At least one of the variables is set to True ?
- Easy: $x_1 \vee x_2 \vee ... \vee x_n$
- At most one of the variables is set to True ?
- Idea: Ensure that for every pair of variables at least one of the variables is False !
- How to model generic 'at least $k$ of' and 'at most $k$' of constraints ?

## Encoding at most k of Constraint

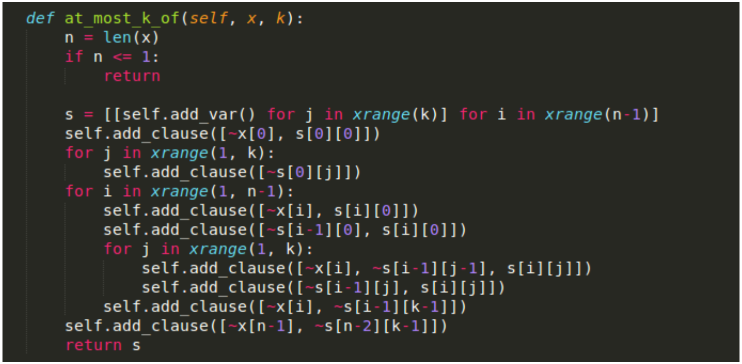
Below is the encoding that requires additional $k * n$ variables: $s$ to ensure that at most $k$ of $n$ variables from $x$ sequence are chosen:

$$
\begin{array}{l}
(\neg x_1 \vee s_{1,1}) \\
(\neg s_{1,j}) \qquad \text{for } 1 < j \leq k \\
\left.\begin{array}{l}
(\neg x_i \vee s_{i,1}) \\
(\neg s_{i-1,1} \vee s_{i,1}) \\
\left.\begin{array}{l}
(\neg x_i \vee \neg s_{i-1,j-1} \vee s_{i,j}) \\
(\neg s_{i-1,j} \vee s_{i,j})
\end{array}\right\} \quad \text{for } 1 < j \leq k \\
(\neg x_i \vee \neg s_{i-1,k})
\end{array}\right\} \text{for } 1 < i < n \\
(\neg x_n \vee \neg s_{n-1,k})
\end{array}
$$

[1] Source: Towards an Optimal CNF Encoding of Boolean Cardinality Constraints

## Implementation of at most k of Constraint

Below is my implementation of at most *k* of encoding proposed by Sinz:



```python
def at_most_k_of(self, x, k):
    n = len(x)
    if n <= 1:
        return

    s = [[self.add_var() for j in xrange(k)] for i in xrange(n-1)]
    self.add_clause([~x[0], s[0][0]])
    for j in xrange(1, k):
        self.add_clause([~s[0][j]])
    for i in xrange(1, n-1):
        self.add_clause([~x[i], s[i][0]])
        self.add_clause([~s[i-1][0], s[i][0]])
        for j in xrange(1, k):
            self.add_clause([~x[i], ~s[i-1][j-1], s[i][j]])
            self.add_clause([~s[i-1][j], s[i][j]])
        self.add_clause([~x[i], ~s[i-1][k-1]])
    self.add_clause([~x[n-1], ~s[n-2][k-1]])
    return s
```

## For Further Reading I

Handbook of Satisfiability, Biere, A., Heule, M., Van Maaren, H., Walsh, T, February 2009

- Towards an Optimal CNF Encoding of Boolean Cardinality Constraints - Carsten Sinz
- Finding a Collective Set of Items: From Proportional Multirepresentation to Group Recommendation ∗ Piotr Skowron University of Warsaw Warsaw, Poland Piotr Faliszewski AGH University Krakow, Poland J´erˆome Lang Universit´e Paris-Dauphine Paris, France