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

Michał Mrowczyk

Department of Computer Science
AGH Kraków

Seminarium Dyplomowe, 2016

Outline

OWA-Winner Problem - One More Time

- Notation used in this presentation:
  - α_k - OWA vector
  - u_{i,a_j}- Utility function value that i -th voter assigns to the j -th item/candidate
  - $x_{i,j,k}$- 1if 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 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 OWA-Winner problem instances to ILP and then to SAT-CNF

ILP Formulation

[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 :

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

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

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

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

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

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

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

SAT Formulation

```
model = OWAModel(length=8)
y = [model.add_var('y' + str(j)) for j in xrange(m)]
x = [[[model.add_var('x' + str(i) + '|' + str(j) + '|' + str(k)) for k in xrange(K)] for j in xrange(m)] for i in xrange(n)]

# (a) Exactly K candidates are chosen
sum([1*yy for yy in y]) == K

# (b) x and y are consistent with each other
for i in xrange(n):
    for j in xrange(m):
        for k in xrange(K):
            model.add_clause([~x[i][j][k], y[j]])

# (c) Only 1 candidate is ranked as k-th best
for i in xrange(n):
    for k in xrange(K):
        model.exactly_one_of([x[i][j][k] for j in xrange(m)])

# (d) Candidate 'j' is ranked precisly on one position for 'i'
for i in xrange(n):
    for j in xrange(m):
        model.at_most_one_of([x[i][j][k] for k in xrange(K)])

# (e) Candidate placed on k-th position is at least as valuable (for particular voter) as one placed on k+1-th position
for i in xrange(n):
    for k in xrange(K-1):
        sum([u[i][j]*x[i][j][k] for j in xrange(m)]) >= sum([u[i][j]*x[i][j][k+1] for j in xrange(m)])

# (objective)
solution, max_val = model.maximize(sum([(alpha[k]*u[i][j])*x[i][j][k] for i in xrange(n) for j in xrange(m) for k in xrange(K)]), lb=0
```

SAT Formulation - Binary Utility and OWA Vector

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

```
model = BinaryOWAModel(length=1)
y = [model.add_var('y' + str(j)) for j in xrange(m)]
x = [[[model.add_var('x' + str(i) + '|' + str(j) + '|' + str(k)) for k in xrange(K)] for j in xrange(m)] for i in xrange(n)]

# (a) Exactly K candidates are chosen
model.exactly_k_of(y, K)

# (b)
for i in xrange(n):
    for j in xrange(m):
        for k in xrange(K):
            model.add_clause([~x[i][j][k], y[j]])

# (c) Only 1 candidate is ranked as k-th best
for i in xrange(n):
    for k in xrange(K):
        model.exactly_k_of([x[i][j][k] for j in xrange(m)], 1)

# (d) Item 'j' is ranked precisely on one position for 'i'
for i in xrange(n):
    for j in xrange(m):
        model.at_most_k_of([x[i][j][k] for k in xrange(K)], 1)

l = [x[i][j][k] for i in xrange(n) for j in xrange(m) for k in xrange(K) if alpha[k]*u[i][j] > 0]

model.at_least_k_of(l, int(sys.argv[1]))
```

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:

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

[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:

```
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
```

General Results

- Tests were conducted by applying the PicoSAT solver to general and binary OWA models
- Bigger OWA-winner problem instances can be solved by using binary OWA model (because 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]
```

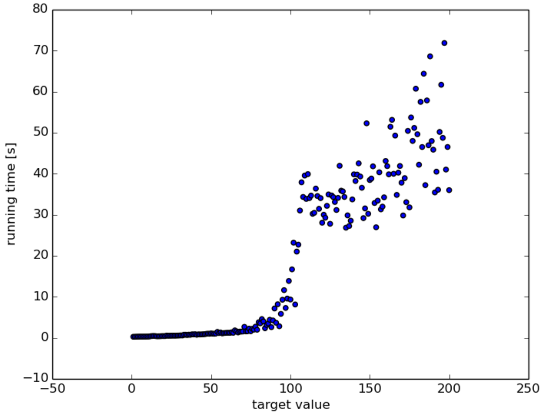
Example: Binary (Approval) OWA-winner Instance

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

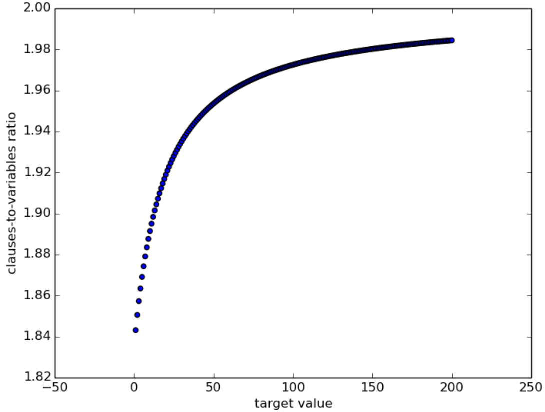


Clauses to Variables Ratio

- 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 OWA-Winner and Integer Factorization instances

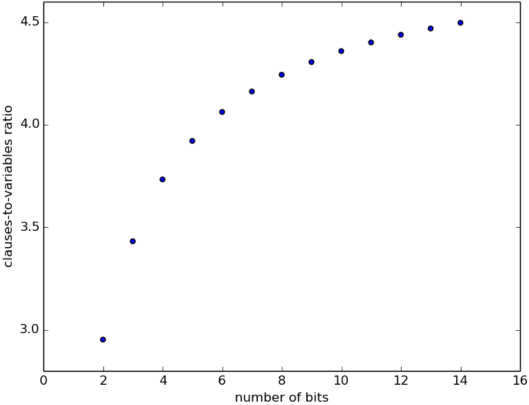
Clauses to Variables for OWA-Winner Instances

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



Summary

- Reduction from OWA-Winner to SAT-CNF was presented with some (but not all) quirks
- In addition to the general OWA-Winner we prepared more efficient reduction for 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
- TODO: Consider other metrics of hardness e.g. proximity to XORSAT, 2SAT,..

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´er^ome Lang Universit´e Paris-Dauphine Paris, France