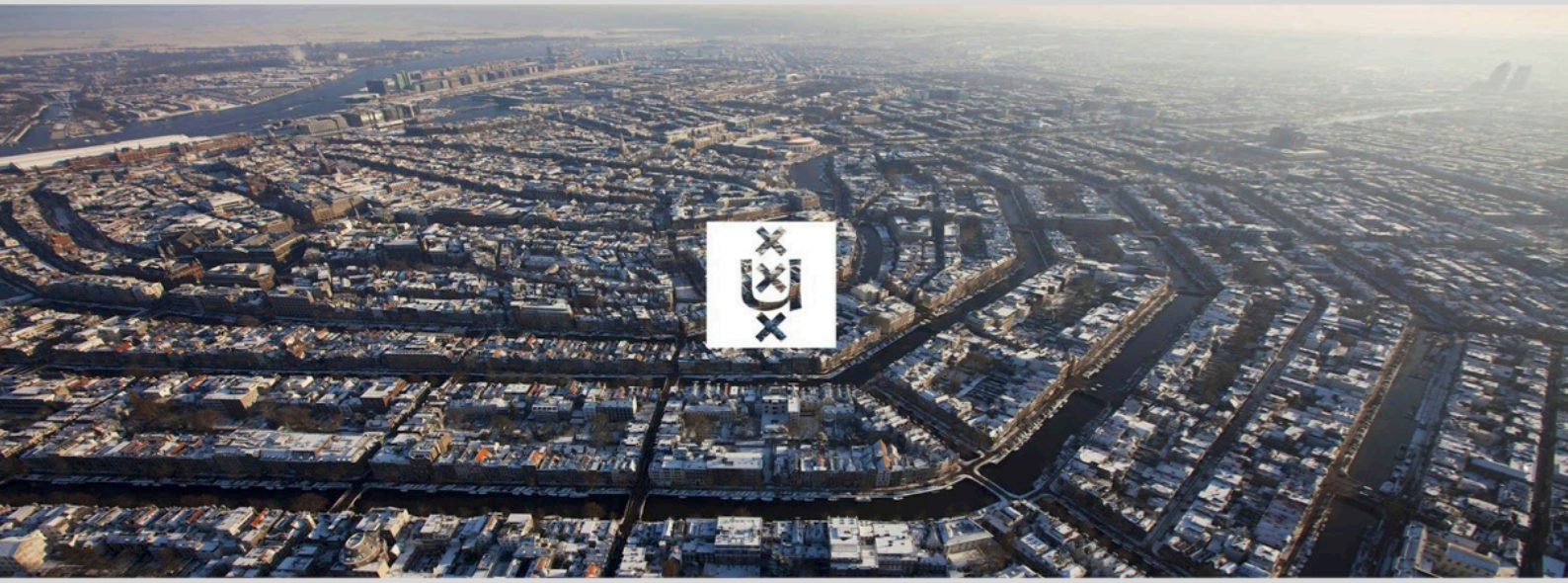

Institute for Logic, Language and Computation

Encoding of Social Choice Problems in SMT solvers (Lab Report 5)

Shuai Wang

UID: 11108339

ILLC, UvA

Contents

ABSTRACT

In this assignment, I explored the possibility of encoidng social choice problems in SAT and SMT solver. More specifically, this project takes advantage of modern SMT solver, Yices. Section 2 presents the Stable Marriage Problem. Section 3 discuss about the Stable Roommate Problem and finally we give a design for the Dutch School Assignment System using SMT solvers in Section 4.

1 INTRODUCTION TO YICES

Satisfiability Modulo Theory is combinint Satisfiability with Theories such as bit vectors, uninterpreted functions and arithematics (integer programming). Yices[3] is a SMT (Satisfiability Modulo Theories) solver for such problems. In this project we used an older version[3, 1] of Yices (1.0.40). This is because the only working Haskell binding [1] is not working with any other version of Yices and due to the limit of time, this older version of Yices was used. Yices is not a good choice for MaxSAT since its optimisation part has not been maintained for years.

2 STABLE MRRIAGE PROBLEM

2.1 EXERCISE 1

Show that if there are equal numbers of men and women, the algorithm always terminates. Hint: analyze what happens to the preference lists of the men. Observe that no man proposes to the same woman twice.

To show that no man propose to the same woman twice, we need to consider the preference list is in strict order. It is impossible for a man i to propose to j again since he only propose to women in the rest of the list [4]. Assume that he would propose to this woman again, this would make the algorithm not terminating.

2.2 EXERCISE 2

To show stability, we have to show that each step through the step function preserves the invariant "the set of currently engaged men and women is stable." Prove this.

To show that the algorithm maintain stableness at each step, we need to look into the algorithm. For the first step, a woman would simply accept the proposal. If there are multiple men proposing to the same woman, she would compare the choose the one ranking higher in the prefernce list. If the algorithm is not stable then there is a woman engaged with a man lower in the prefernece list. From the description of the algorithm, we know this is impossible. For each step, the algorithm go through each preference list and only if there is a preferred choice would this person change the a new proposer. Thus each man and woman would therefore get matched in each step with their favourite partner in their prefernece list [4].

2.3 AN INTRODUCTION TO MAXSAT

One problem of the algorithm presented above is that if the females are considered first, the engagement would then lose the fairness for men (i.e. $woman - optimal$) [4]. Next I present a

[1] `https://github.com/airobert/yices_haskell`

simple but effecient encoding for this problem to a MaxSAT problem after a brief introduction to MaxSAT.

Satisfiablility is to determine if there exist an assignment making all propositional constraints (clauses) true. In contrast, Maximum Satisfiabiliy (MaxSAT) problem is to try to find such assignment to satisfy as many clauses as possible. With the definition above, we can then define weighted maximum satisfiabiliy problems (Weighted MaxSAT). In these problems, each clause carries a $weight$ (usually as an integer). The problem is to maximise the sum of the weight of satisfied clauses. A special case of such problem is the Partial Weighted MaxSAT (PWMaxSAT) problem where $hardclauses$ must be satisfied and $softclauses$ which carries the weight. The model should maximise the weight carried. As is known, such problem is a combination of satisfiabiliy and optimisation and is NP-complete. All problems of SAT, MaxSAT and Weighted MaxSAT can be regarded as special cases of Partial Weighted MaxSAT [5]. The solver for such problems are usually referred to as MaxSAT solver in short.

Examples of SAT, MaxSAT, Weighted MaxSAT were illustrated as in the following table.

SAT problems and Weighted SAT problems			
SAT	MaxSAT	Weighted MaxSAT	PWMaxSAT
$\neg p \wedge q$	$\neg p \wedge q$	$\neg p \wedge q$(weight = 2)	$\neg p \wedge q$ (hard clause)
	$\neg p \wedge \neg q$	$\neg p \wedge \neg q$ (weight = 3)	$\neg p \wedge \neg q$ (weight = 6)
$\neg q$	$p \wedge \neg r$	$p \wedge \neg r$ (weight = 5)	$p \wedge \neg r$ (weight = 5)
	$p \wedge \neg r$	$q \wedge \neg r$ (weight = 1)	$q \wedge \neg r$ (hard clause)

TABLE 1: EXAMPLES OF SAT, MAXSAT, WEIGHTED MAXSAT AND PWMAXSAT

2.4 ENCODING OF THE STABLE MARRIAGE PROBLEM

We first introduce a matrix of propositional variables. A variable in row i and column j represents the engagement of a man i and a woman j. Also we need a function to convert preference to list of integers as weight. The weight of this propositional variable is the sum of the weight function for both man and woman.

Hard Clauses Constraints that must be satisfied in the model are the followngs:

1. Each man is engaged with only one woman
2. Each woman is engaged with only one man
3. Each man must get engated with some woman
4. Each woman must get engated with some man

Soft Clauses 1. Each engagement carries the weight of the sum of the weight of preference of man and woman.

The problem is then to obtain the best score of sum of weight by satisfyin as many clauses with consideration of their weight. We encode this problem in Yices. There should be two possible out comes: SAT and Unknown. The followng function take two preference list and return an engagement.

```
obtain_eng :: WprefY -> WprefY -> IO Engaged
obtain_eng wpl mpl =
  do yp@(Just hin, Just hout, Nothing, p)
  <- createYicesPipe yicesPath []
     let n = length wpl
     let m = get_name "m" n
     let w = get_name "w" n
     let vl_w = var_list_w w m
     let vl_m = var_list_m vl_w
     let all_var = var_all_list vl_w
     --obtain a list of men and women
     let men = [1..n]
     let women = [1..n]

     let unique_eg = unique_engate vl_w vl_m
     print "-----unique engage-----"
     print unique_eg
     print "-----must_engate-------"
     let must_eg = must_engate vl_w vl_m
     print must_eg
     print "------test sat--------"
     runCmdsY yp ((defs w m) ++ unique_eg ++ must_eg)
     -- hard clauses must be satisfiable
     --Sat ss <- checkY yp
     --return ss
     let mymax = (encode_max vl_w wpl vl_m mpl)
     print "---------maxsat--------------------"
     print mymax
     runCmdsY yp (mymax  ++ [MAXSAT])
     print "-------result<sat or unknown, can't be unsat>------"
     Sat ss' <- checkMAX yp
     print ss'
     let model = obtain_bool_value all_var ss'
     print "--------and model is ---------------------"
     print model
```

```
--return model
let eng = get_eng women men model
return eng
```

2.5 PROOF OF STABLE MATCHING ENCODING

Here I give the proof of the encoding in brief. It would be unstable if there is such a case that a man i is engaged with a woman s but i prefers a woman t and s prefers a man j and vice versa. However j and t are engated. This would not be possible since we introduce a linear map from weight to preference. The weight of preferene of j and t plus the weight of preference of i and s is less than the weight of preferene of i and t plus the weight of preference of j and s. If the MaxSAT solver gives an optimal solution then the answer is guaranteed to be correct.

2.6 EVALUATION

To run, simply type **test_stable_maxsat** for the first example and run simply type **test_stable_maxsat2** for the second example.

It takes almost no time to obtain a solution for 3 women and 3 men. However, Yices didn't manage to solve problem of 10 women and 10 men and returned the following error:

```
"unknown" *** Exception: user error (Pattern match failure
  in do expression at 5.hs:348:8-14)
```

This showed that the Yices's MaxSAT was not stable. Further attempts should be considered on more recent versions. Due to the limit of time, the student didn't manage to get the binding working for Yices 2.2 and later versions.

3 SOLVING THE STABLE ROOMMATE PROBLEM WITH MAXSAT

3.1 EXERCISE 3

Instead of presenting an implementation of Irving's algorithm, the student decides to attempt solving stable roommate problem by encoding to MaxSAT.

There are four students in this game. Their preference list is as follows:

1. 2 > 3 > 4
2. 1 > 3 > 4
3. 2 > 1 > 4
4. 3 > 2 > 1

First we introduce variables and weights. A matrix of variables is introduced with at position in row i and column j represention student i and j are assigned to the same room. The weight is generated by a (linear) function from the preference list.

Similarly we introduce hard and soft clauses:

Hard Clauses Constraints that must be satisfied in the model are the followngs:

1. Each person is not going to live alone (row i and column j is assigned to be false).
2. Each person must be matched with someone.
3. Each person must not be matched with multiple person.
4. If i match with j then j is also matched with i (a mutual match). This makes variable at row i and colum j has the same value as a variable at row j and colum i.

Soft Clauses 1. Each variable at row i and column j carries the weight of the preference of j for person i.

A hard-coded function is as follows:

```
test_roommate =
  do yp@(Just hin, Just hout, Nothing, p)
      <- createYicesPipe yicesPath []
     let n = 4
     let unique_eg =  (concat (map ctr_unique val_matrix))
     print "----unique engage ----"
     print unique_eg
     print "----must_engate------"
     let must_eg = (map (\l -> ASSERT (OR l)) val_matrix)
     print must_eg
     let diagnal =
          [ASSERT (NOT ((val_matrix !! (i-1)) !!(i-1)))
          | i <- [1..4]]
     print "-----diagnal ----------"
     print diagnal
     print "-----mirror ----------"
     let mirror =
          [ ASSERT (((val_matrix !! (i-1)) !!(j-1)):=
          ((val_matrix !! (j-1)) !!(i-1)))
          | i <- [1..4], j <- [1..4], i < j ]
     print mirror
     print "-----test sat -------"
     runCmdsY yp ((def_matrix) ++
```

```
      unique_eg ++ must_eg ++ diagnal ++ mirror)
-- hard clauses must be satisfiable
Sat ss <- checkY yp
return ss

--let mymax = (encode_max vl_w wpl vl_m mpl)
print "---------maxsat--------------------"
print room_max
runCmdsY yp (room_max  ++ [MAXSAT])
print "---result<sat or unknown, can't be unsat>---"
Sat ss' <- checkMAX yp
print ss'
let model = obtain_bool_value (concat val_matrix) ss'
print "--------and model is ---------------------"
print model
--return model
let eng = get_eng [1..4] [1..4] model
return eng
```

3.2 EVALUATION

Type **test_roommate** to run the test for the function. The model is: [(1,2),(2,1),(3,4),(4,3)]. The match is therefore 1 with 2 nd 3 with 4. This was checked by hand to be an optimal match. No further test was performed due to the limit of time.

4 DUTCH SCHOOL ASSIGNMENT SYSTEM

4.1 EXERCISE 4

In this section, I present a simple design and encoding of the Dutch School Assignment System using MaxSAT. Due to the limit of time, I didn't manage to implement anything.

There are around 6000 students and 30 school in Amsterdam (area). A Dutch School Assignment System takes a list of preference of schools from each student. We introduce two class of variables: individual variables: each student has 30 propositional variables representing the school he/she got assigned to. Schools have integer variables for capacit limit.

Similarly we introduce hard and soft clauses:

Hard Clauses Constraints that must be satisfied in the model are the followngs:

1. Each person is assigned to only one school.

2. Each school must have at lease one student.
3. Each school has a capacity and the total amount of students can not be more than the capacity for each school.

Soft Clauses

1. Each student has a prefernece list of schools and the corresponding propositional variables are to be added to with the weight calculated from the preference list.

Using this simple encoding, we can implement a Dutch School Assignment System using modern MaxSAT solvers. In particular, SAT4J is an idea solver in consideration [5].

4.2 ADDITIONAL FEATURES

We keep the design flexiable and extensivable for further extension of new features. We can introduce more variable such as distance between school and home and student exam scores and then introduce more constrains.

5 FUTURE WORK

In this project, we used an older version of Yices. However, Yices (2.2) [2] is now more powerful and more stable after years of development. It would be necessary to update the Haskell binding after the project for evaluation for better effeciency and accuracy.

References

[1] Leonardo de Moura and Bruno Dutertre. Yices 1.0: An efficient smt solver. *The Satisfiability Modulo Theories Competition (SMT-COMP)*, page 10, 2006.

[2] Bruno Dutertre. Yices 2.2. In *Computer Aided Verification*, pages 737–744. Springer, 2014.

[3] Bruno Dutertre and Leonardo De Moura. The yices smt solver. *Tool paper at http://yices. csl. sri. com/tool-paper. pdf*, 2(2), 2006.

[4] Dan Gusfield and Robert W Irving. *The stable marriage problem: structure and algorithms*. MIT press, 1989.

[5] Daniel Le Berre, Anne Parrain, et al. The sat4j library, release 2.2, system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.