184.701 UE Declarative Problem Solving
Exercise 1: Satisfiability

Uwe Egly Hans Tompits Christoph Redl
Institute of Information Systems, Knowledge-Based Systems Group 184/3
Vienna University of Technology

Summer term 2017

`dps-2017s@kr.tuwien.ac.at`

1 Introduction

This exercise deals with a well-known combinatorial design problem: the *round robin problem*. The round robin problem represents a sports scheduling problem. The task is to find a feasible schedule for an even number of n teams, where every team must play against every other team exactly once. Considering constraints on how the competing teams can be paired and how each team's games are distributed over the entire schedule makes this a hard combinatorial problem. One possible approach to find solutions for this problem is to encode it as a satisfiability problem (SAT), i.e., as a propositional formula, such that models of the formula represent solutions to the problem. The problem solution is computed by a SAT solver. A model for the formula represents a feasible schedule.

Your task is to solve the round robin problem by reducing it to SAT and to invoke a SAT solver to compute the solution(s). The reduction is implemented by a program which takes the number of teams as input and outputs a correct SAT encoding for the given number of teams in DIMACS format. You can find a short and straightforward explanation of the DIMACS format at [2]. A further task is to compare the runtimes of different SAT solvers (`picosat`, `plingeling`, `ubcsat`) for different problem instances (for an increasing number of teams). Additionally you will have to provide a solution printer that decodes the output of `picosat` and `plingeling` and prints the resulting schedule in a user-friendly way as described below.

In the next sections, we describe the general problem and discuss the subtasks to be solved.

2 General Problem Description

The n-team round robin problem is formally defined as follows [1]:

1. There are n teams (n even) and every distinct pair of teams meets exactly once.
2. The season lasts $n - 1$ weeks.
3. Every team plays one game in each week of the season.
4. There are $n/2$ fields and, each week, every field is scheduled for one game.
5. No team plays more than twice on the same field over the course of the season.

You will find a SAT encoding of this problem in [1]. Read this paper carefully (you will find it in the TUWEL course) and use the given SAT encoding to solve Subtask 1.

3 Subtask 1

Use the given SAT encoding of the round robin problem from [1] to develop and implement a program that requires an even number n of teams as command line argument and that prints a propositional formula in DIMACS format [2] to standard output. The rules for the SAT encoding are described in step 1 to 6 in [1]. Use every single rule to implement a complete encoding of the problem.

You may either use `Perl` version 5.24.1 [4], or `Python` version 3.6.0 available at [5], to implement the translation into a propositional formula in conjunctive normal form in DIMACS format (see [2]). According to your prefered programming language write your program in a file called `roundrobin_to_sat.pl` (for `Perl`) or `roundrobin_to_sat.py` (for `Python`).

Use the `Makefile` that was provided in the TUWEL course to interpret your source code. Put the `Makefile` to your source code working directory. Have a look at the `Makefile` and adjust the variables to your preferences, e.g., set the variable `MNR` to your immatriculation number, choose your prefered programming language and make sure that `make` is able to find the correct version of the compiler/interpreter for `Perl` or `Python` and the correct version of `picosat`.

To generate the CNF encodings call your program for every even number $n \in \{4, \ldots, 16\}$ using the `Makefile`; e.g., for a number of 12 teams, invoke

```
make run_task1 NO_TEAMS=12
```

The Makefile pipes the output of your program into a file, e.g., `roundrobin_12.cnf`.

For this subtask download and install the SAT solver `picosat` version 965 from [7] (it is already included in Debian/Ubuntu packages but make sure you have the correct version). `picosat` computes a model for satisfiable problem instances. If you use the Makefile as described below, a file, e.g., named `roundrobin_12.sat`, will be produced containing the solution of `picosat` in the following form (e.g., for four variables):

```
s SAT
v 1 -2 3 -4
```

Please have a look at [3] to find out more about the standard output format of state of the art SAT solvers. Implement a second program (in the same programming language) that takes the file containing the output of `picosat` as command line argument, and prints the solution in a textual way to standard output. According to your prefered programming language write this program in a file called `sat_to_text.pl` (for `Perl`) or `sat_to_text.py` (for `Python`).

To run `picosat` for a certain problem instance and to generate the textual representations call this program using the `Makefile`; e.g., for a number of 6 teams, invoke

```
make run_task2 NO_TEAMS=6
```

Do not forget to set the variable `PICOSAT` to a correct value. This will produce an intermediate file `roundrobin_6.sol` containing the standard output of your program and call your solution printer `sat_to_text.{pl,py}` with that file as command line argument.

The output of your program (printed to stdout) has to be of the following format. Columns represent weeks and rows represent fields. Columns are tab-separated, rows are newline-separated. Each entry consists of the team-numbers playing against each other separated by : (colon).

Take the 6-team round robin problem as example. The timetable has be of the following form:

```
1:2 2:3 1:5 4:6 3:5
3:4 1:6 3:6 2:5 1:4
5:6 4:5 2:4 1:3 2:6
```

Analyze the output of `picosat` and provide a file named `documentation.pdf`: For the problem instances of size 4, 8, 10 and 14 take the models that are provided by `picosat`, extract the represented round robin schedules and present them in a readable way (e.g., like Table 1 in [1]). Additionally provide runtime analysis for growing instances, i.e., draw a diagram showing the runtime of `picosat` for all instances from 4 up to 16 teams. Include this diagram in `documentation.pdf` and provide a short discussion. What happens if you increase the number of teams to 18 or even 20?

4 Subtask 2

After you have tested and documented your SAT encodings for the round robin problem with `picosat`, the next subtask is to try two more SAT solvers.

Download `plingeling-ayv-86bf266-140429.zip` from [8] and install `plingeling`. Use `plingeling` to produce models for all your problem instances `roundrobin_`n`.cnf` (again from 4 to 16 teams). For the problem instances of size 8 and 14 analyze if the computed schedules are the same as those computed by `picosat`. Since the output format of `plingeling` is the same as of `picosat` your solution printer again should be able to print the solution in a readable way. Further compare the runtime of `plingeling` for all instances with the runtime of `picosat` (e.g., draw another curve in your diagram) and give a brief summary of your conclusions.

As next step, download and install the stochastic SAT solver `ubcsat` version 1.1 from [9]. You can find some further information about `ubcsat` at [10]. For testing `roundrobin_`n`.cnf` with `ubcsat` you have to select an algorithm: use the command line option `-alg saps` to create models for all problem instances. Again compare the produced schedules for 8 and 14 teams to the results of `picosat` and `plingeling` and document the results in `documentation.pdf`. As for the previous subtask collect the run times for the single instances and update your run time diagram with a third curve showing the results of `ubcsat`. Does `ubcsat` always find a model and why (not)?

Remark:
Another very famous SAT solver ist `Minisat` (see [6]). You are encouraged to download it and try it out.

5 Subtask 3

Based on the given SAT encoding of the round robin problem from [1], encode a variant of the round robin problem, where the numbers of teams and fields are n and $n/2$ again, respectively, but where

- the season lasts $2n - 2$ weeks,
- every two teams play each other exactly twice,
- no team plays more than four times in the same field over the course of the season, and additionally
- for each field i, with $1 \leq i \leq n/2$, and each week j, with $1 \leq j \leq n - 1$, the two teams playing each other in this slot, must not play each other in field i and week $j + n - 1$ again (i.e., at most one of them may play in that slot).

Document the adapted set of propositional variables and the adapted clauses in `documentation.pdf`, and save the program in `roundrobin_to_sat_2nd_leg.pl` (for `Perl`) or `roundrobin_to_sat_2nd_leg.py` (for `Python`).
Again your program should print the propositional formula in DIMACS format to standard output. Invoke, e.g.,

```
make run_task3 NO_TEAMS=6
```

for the problem instance with 6 teams to produce the file `roundrobin_2nd_leg_6.cnf`. Compute solutions with all three SAT solvers again, for the instances $n \in \{4, 6\}$. Finally compare the runtimes for finding a schedule for 4 and 6 teams and document the results in `documentation.pdf`.

Your solution printer `sat_to_text.pl` (for `Perl`) or `sat_to_text.py` (for `Python`) from subtask 1 should again be able read and interpret the output of `picosat` and `plingeling`.

To run `picosat` for a certain problem instance and to generate the textual representations call this program for the adapted version using the `Makefile`; e.g., for a number of 6 teams, invoke

```
make run_task4 NO_TEAMS=6
```

This will run `picosat` on your adapted encoding and produce the according output file `roundrobin_2nd_leg_6.sol` containing the standard output of your program `sat_to_text.{pl,py}`.

6 Submission

Deadline for the submission of this exercise is **Sunday April 23 2017 23:55**. Upload a ZIP-file with your solution to the TUWEL system. Before the deadline you can upload your solution several times. Submissions after the deadline will be rejected. Use the target

```
make zip
```

from `Makefile` to create your ZIP-file. The result will be `XXXXXXX_ex1.zip`, where `XXXXXXX` is replaced by your immatriculation number (make sure you have set the `MNR` variable in the `Makefile` to the correct value). Make sure the ZIP-file contains 5 files:

- the adjusted `Makefile`,
- `documentation.pdf`, and either
- `roundrobin_to_sat.pl`, `sat_to_text.pl` and `roundrobin_to_sat_2nd_leg.pl`, or

- `roundrobin_to_sat.py`, `sat_to_text.py` and `roundrobin_to_sat_2nd_leg.py`.

7 Questions

Please post questions of general interest to the TISS course forum. In case of technical questions, in particular those revealing (parts of) your solution of this exercise, send an email to `dps-2017s@kr.tuwien.ac.at`.

References

[1] R. Bejar and F. Manya. Solving the Round Robin Problem Using Propositional Logic. In *Proceedings of the National Conference on Artificial Intelligence (AAAI'00)*, pages 262-266, 2000.

[2] `http://logic.pdmi.ras.ru/~basolver/dimacs.html`

[3] `http://www.satcompetition.org/2011/rules.pdf`

[4] `http://www.perl.org`

[5] `http://www.python.org`

[6] `http://minisat.se/MiniSat.html`

[7] `http://fmv.jku.at/picosat`

[8] `http://fmv.jku.at/lingeling`

[9] `https://github.com/dtompkins/ubcsat`

[10] `http://ubcsat.dtompkins.com/`