Automatic Evaluation of Reductions between NP-Complete Problems (Tool Paper)

Carles Creus, Pau Fernández, Guillem Godoy

Universitat Politècnica de Catalunya

July 16, 2014

The RACSO Online Judge (http://racso.lsi.upc.edu/juez)

We have developed an Online Judge for Theory of Computation.
The users are intended to be students of this subject.
Each kind of problem needs a specific evaluator.
The evaluators give a verdict for the submitted solution proposal (ACCEPT or REJECT, with a counterexample in the latter case).
The desirable execution time is ≤ 5 seconds.

Evaluator for Reductions between NP-Complete Problems

Recall that, given two problems P_1, P_2, a polynomial time reduction $R : P_1 \leq P_2$ is a polynomial-time transformation from instances I_1 of P_1 into instances I_2 of P_2 holding $I_1 \in P_1 \Leftrightarrow I_2 = R(I_1) \in P_2$.
If P_1 is NP-hard, then P_2 is NP-hard.
We have exercises describing NP-complete problems P_1, P_2 that ask for $R : P_1 \leq P_2$.
The evaluator tries to determine whether R is correct or not.
Recall that this problem is undecidable.

Evaluator for Reductions between NP-Complete Problems

How to check correctness of R?
The evaluator has a set of tests that are inputs I_1 of P_1 and two additional reductions $R_1 : P_1 \leq \texttt{SAT}$, $R_2 : P_2 \leq \texttt{SAT}$ provided by the problem setter.
For each I_1 the evaluator computes:

$$\begin{array}{ccc} I_1 & \xrightarrow{R} & I_2 \\ \downarrow_{R_1} & & \downarrow_{R_2} \\ S_1 & & S_2 \end{array}$$

and checks equivalence of S_1, S_2 using a SAT-solver (MiniSat).
Problems:

1. The set of tests may not be exhaustive enough.
2. The composition $R_2(R(I))$ produces big SAT instances.
3. Is polynomiality of the reduction tested?

The problem setter is responsible for 1. For item 2 we need small instances. For item 3 we need big instances.

The REDNP programming language

Our solution: we define a programming language called REDNP with the following advantages:

- Defining several reductions between NP-complete problems is easy.
- Using intermediate structured memory is not possible. Thus, performing an exhaustive search over an exponential number of combinations is difficult.

Hence, we can use a set of small instances.

Polynomiality is not 100% assured, but, in principle, only reasonable submissions will be accepted.

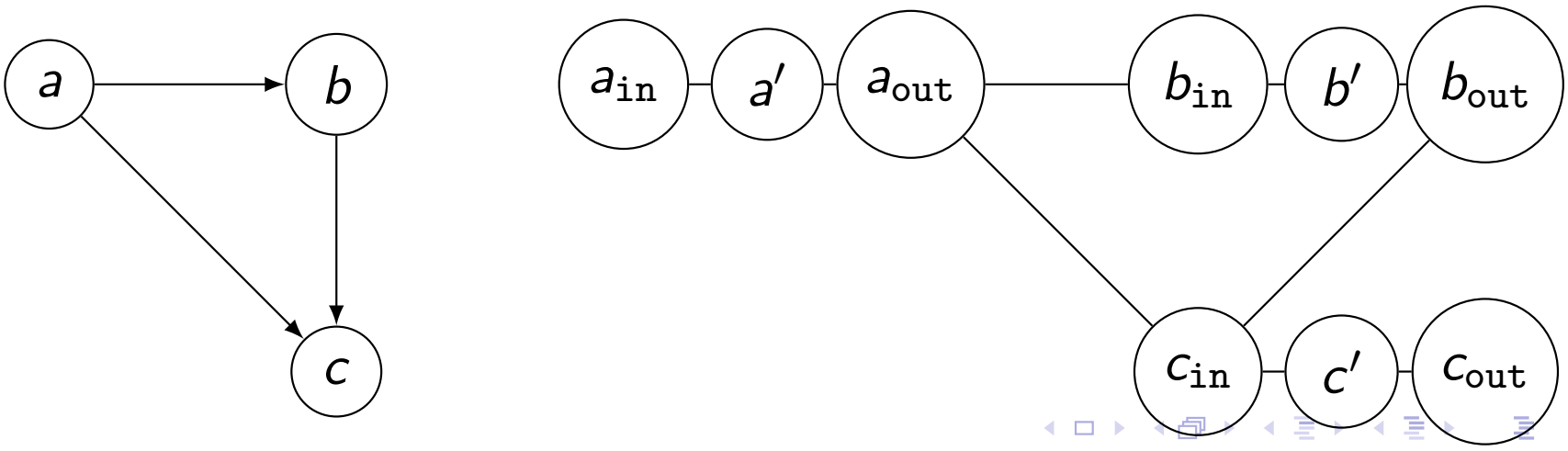
With REDNP, the only structured data is the input and the output. Each exercise defines them according to P_1, P_2.

Example: Directed-Hamiltonian $\leq$ Undirected-Hamiltonian

Problem: Directed-Hamiltonian-Circuit
Input: A directed graph G.
Question: Is there a Hamiltonian cycle in G ?

Problem: Undirected-Hamiltonian-Circuit
Input: An undirected graph G.
Question: Is there a Hamiltonian cycle in G ?

$$R : G = \langle V, E \rangle \mapsto \left\langle \begin{array}{l} \{u_{\mathtt{in}}, u_{\mathtt{out}}, u' \mid u \in V\}, \\ \{(u_{\mathtt{in}}, u'), (u', u_{\mathtt{out}}) \mid u \in V\} \cup \\ \{(u_{\mathtt{out}}, v_{\mathtt{in}}) \mid (u, v) \in E\} \end{array} \right\rangle$$



Example: Directed-Hamiltonian $\leq$ Undirected-Hamiltonian

In this part of the talk the reduction is described with REDNP inside the answer block text of the corresponding problem of the RACSO online judge as follows:

```
main
{
  for (i=1;i<=in.numnodes;i++) {
    out.edges.push="in{i}","aux{i}";
    out.edges.push="out{i}","aux{i}";
  }
  foreach (edge;in.edges)
    out.edges.push="out{edge[0]}","in{edge[1]}";
}
```

Features of REDNP for defining reductions to SAT

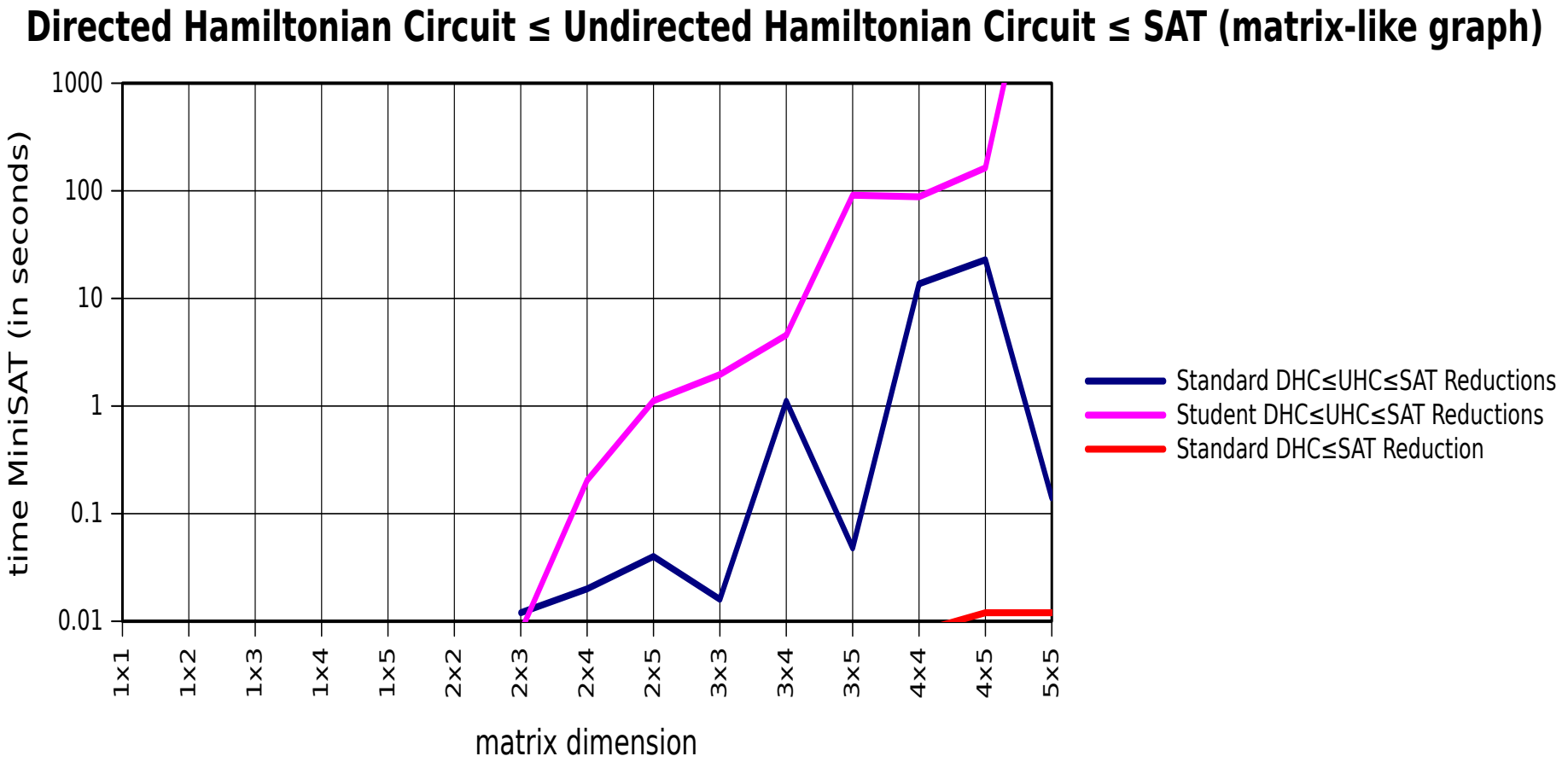
REDNP has the insertsat function predefined, that makes the insertion of clauses from an arbitrary propositional formula over and, or, not, =>, <=, <=> by using the Tseitin transformation. Also, we can use atmost, atleast, exactly inside insertsat.

Example: Vertex-Cover $\leq$ SAT

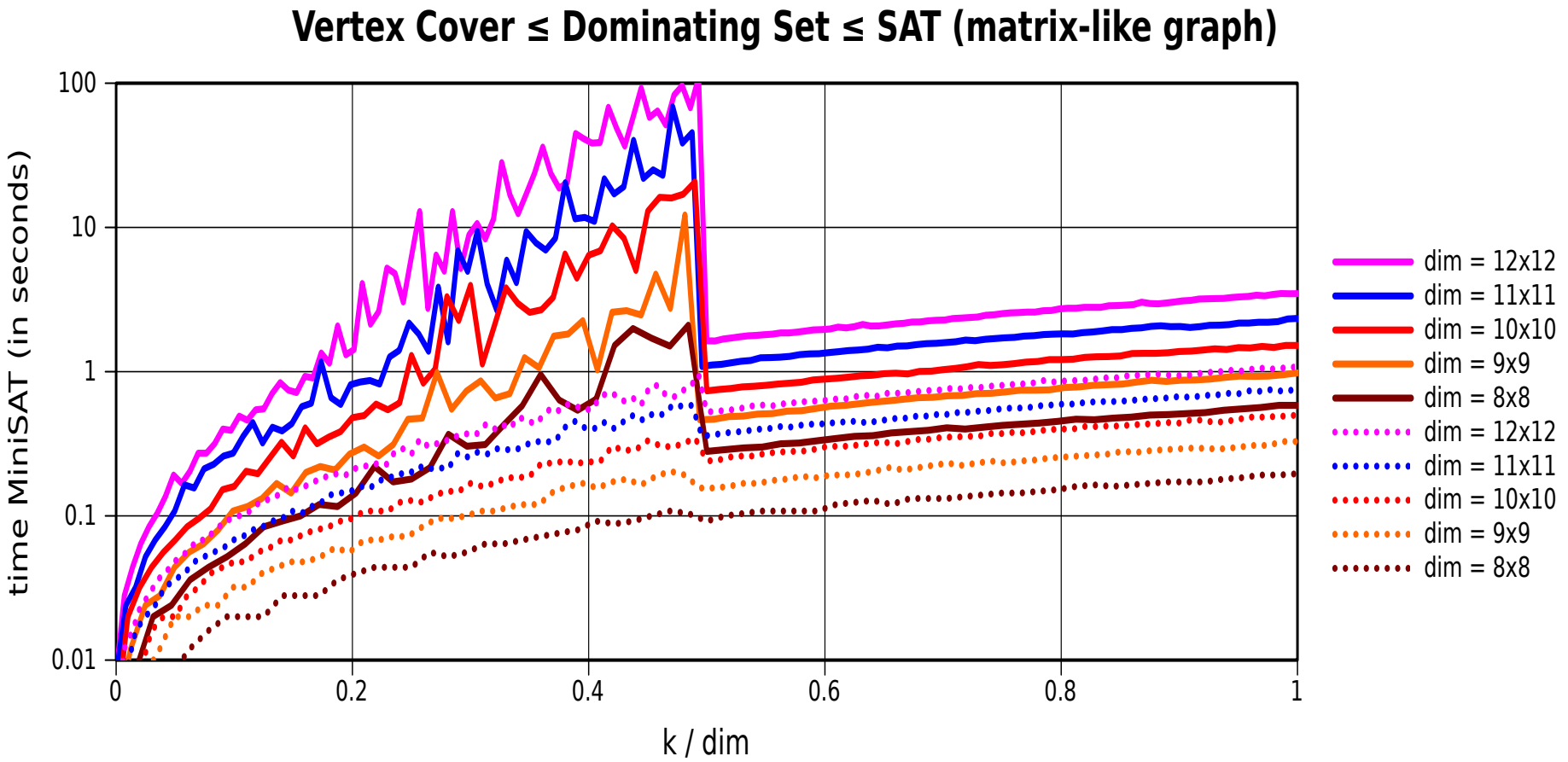
In this part of the talk the reduction is described with REDNP inside the answer block text of the corresponding problem of the RACSO online judge as follows:

```
main
{
  foreach (edge;in.edges)
    insertsat("atleast 1 chosen{edge[0]} chosen{edge[1]}");
  formula="exactly {in.k}";
  for (i=1;i<=in.numnodes;i++)
    formula=formula+" chosen{i}";
  insertsat(formula);
}
```

Performance of composition of reductions



Performance of composition of reductions



Conclusion and further work

- Small conceptually difficult instances produce big difficult SAT instances through $R_2 \circ R$.
- Practice shows that REDNP is confortable enough to formalize reductions (students are familiar with C-like programming languages).
- Very useful: The judge can be used as a support-learning tool and for evaluating practical exams. Students of our subject (not including NP-complete problems) are motivated and work hard (more than 200 exercises solved per person).
- More useful: There are global exams in the judge. Professors can participate or collaborate in preparing exams.
- The SAT instances obtained by composing reductions could be added as benchmarks in SAT-competitions.
- Alternative methods for evaluating reductions?
- Alternative programming languages for defining reductions (easily and forbiding exhaustive exponential searches)?