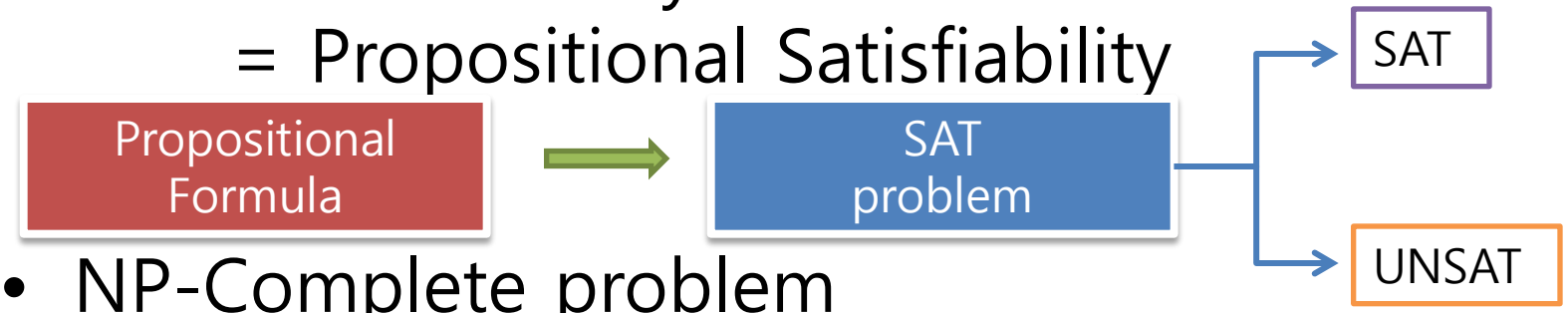
# Application of Propositional Logic - How to Solve Sudoku?

Moonzoo Kim

# SAT Basics (1/2)

- SAT = Satisfiability
  = Propositional Satisfiability




- NP-Complete problem
  - We can use SAT solver for many NP-complete problems
    - Hamiltonian path
    - 3 coloring problem
    - Traveling sales man's problem
- Recent interest as a verification engine

# SAT Basics (2/2)

- A set of propositional variables and Conjunctive Normal Form (CNF) clauses involving variables
  - $(x_1 \vee x_{2'} \vee x_3) \wedge (x_2 \vee x_{1'} \vee x_4)$
  - $x_1$, $x_2$, $x_3$ and $x_4$ are variables (true or false)
- Literals: Variable and its negation
  - $x_1$ and $x_1'$
- A clause is satisfied if one of the literals is true
  - $x_1$=true satisfies clause 1
  - $x_1$=false satisfies clause 2
- Solution: An assignment/interpretation/model that satisfies all clauses

# DIMACS format for CNF

- The file can start with comments, that is lines beginning with the character **c**.
- Right after the comments, there is the line **p cnf <nbvar> <nbclauses>** indicating that the instance is in CNF format;
  - <nbvar> is the exact number of variables appearing in the file;
  - <nbclauses> is the exact number of clauses contained in the file. It is guaranteed that each variable between 1 and nbvar appears at least once in a clause.
- Then the clauses follow.
  - Each clause is a sequence of distinct non-null numbers between -nbvar and nbvar ending with **0** on the same line.
  - Positive numbers denote the corresponding variables.
  - Negative numbers denote the negations of the corresponding variables.
  - A clause is not allowed to contain the opposite literals i and -i simultaneously

```
c
c start with comments
c
c
p cnf 5 3
1 -5 4 0
-1 5 3 4 0
-3 -4
```

# Example

- DIMACS SAT Format
  - Ex. $(x_1 \vee x_2' \vee x_3)$

    $\wedge (x_2 \vee x_1' \vee x_4)$

```
p cnf 4 2
1 -2 3 0
2 -1 4 0
```

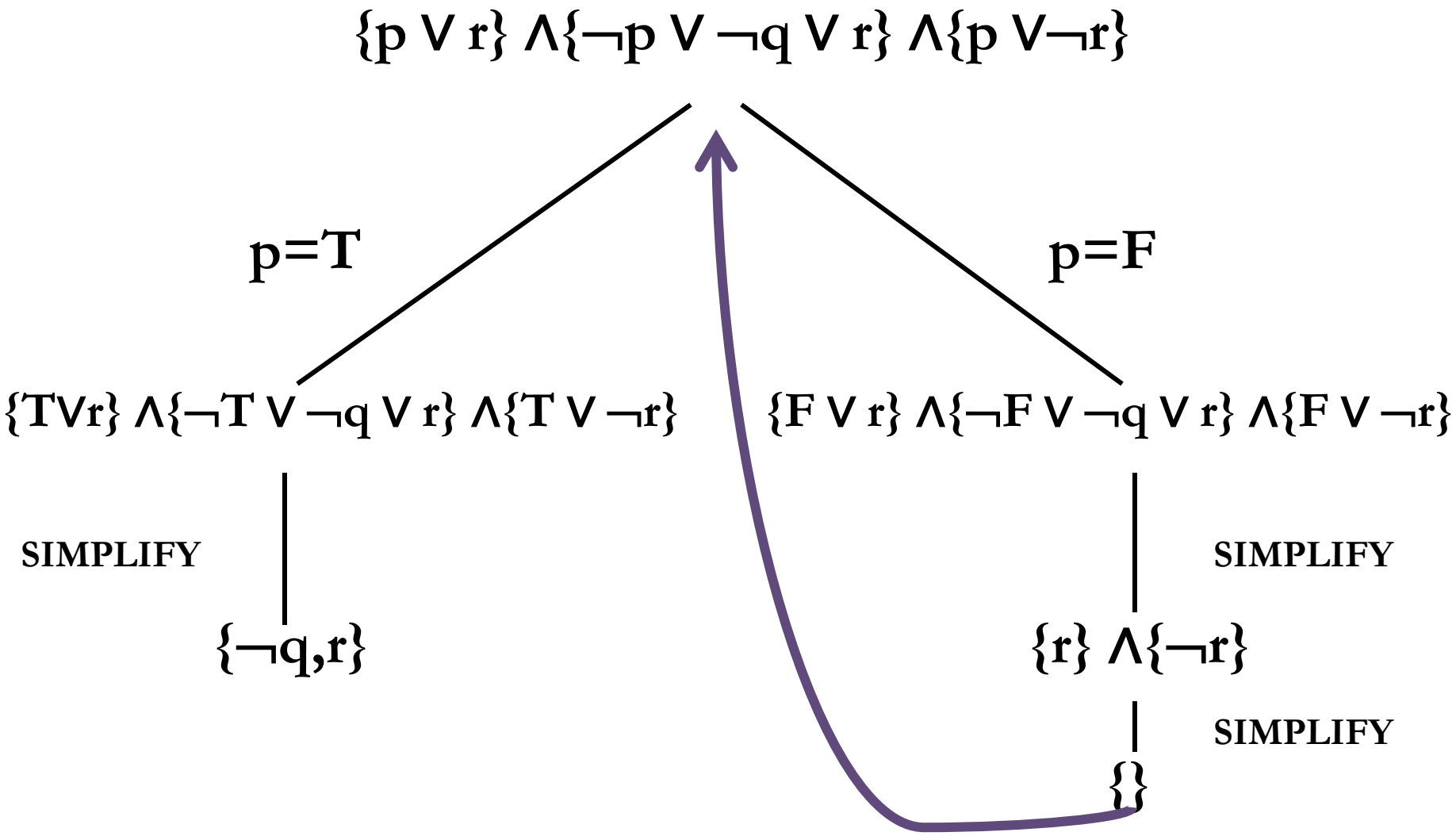
Model/ solution

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | Formula |
|---|---|---|---|---|
| T | T | T | T | T |
| T | T | T | F | T |
| T | T | F | T | T |
| T | T | F | F | T |
| T | F | T | T | T |
| T | F | T | F | F |
| T | F | F | T | T |
| T | F | F | F | F |
| F | T | T | T | T |
| F | T | T | F | T |
| F | T | F | T | F |
| F | T | F | F | F |
| F | F | T | T | T |
| F | F | T | F | T |
| F | F | F | T | T |
| F | F | F | F | T |

# DPLL (Davis-Putnam-Logemann-Lveland) Algorithm

```
/* The Quest for Efficient Boolean Satisfiability Solvers
 *  by L.Zhang and S.Malik, Computer Aided Verification 2002 */
DPLL(a formula φ, assignment) {
    necessary = deduction(φ, assignment);
    new_asgnment = union(necessary, assignment);
    if (is_satisfied(φ, new_asgnment))
        return SATISFIABLE;
    else if (is_conflicting(φ, new_asgnmnt))
        return UNSATISFIABLE;
    var = choose_free_variable(φ, new_asgnmnt);
    asgn1 = union(new_asgnmnt, assign(var, 1));
    if (DPLL(φ, asgn1) == SATISFIABLE)
        return SATISFIABLE;
    else {
        asgn2 = union (new_asgnmnt, assign(var,0));
        return DPLL (φ, asgn2);
    }
}
```

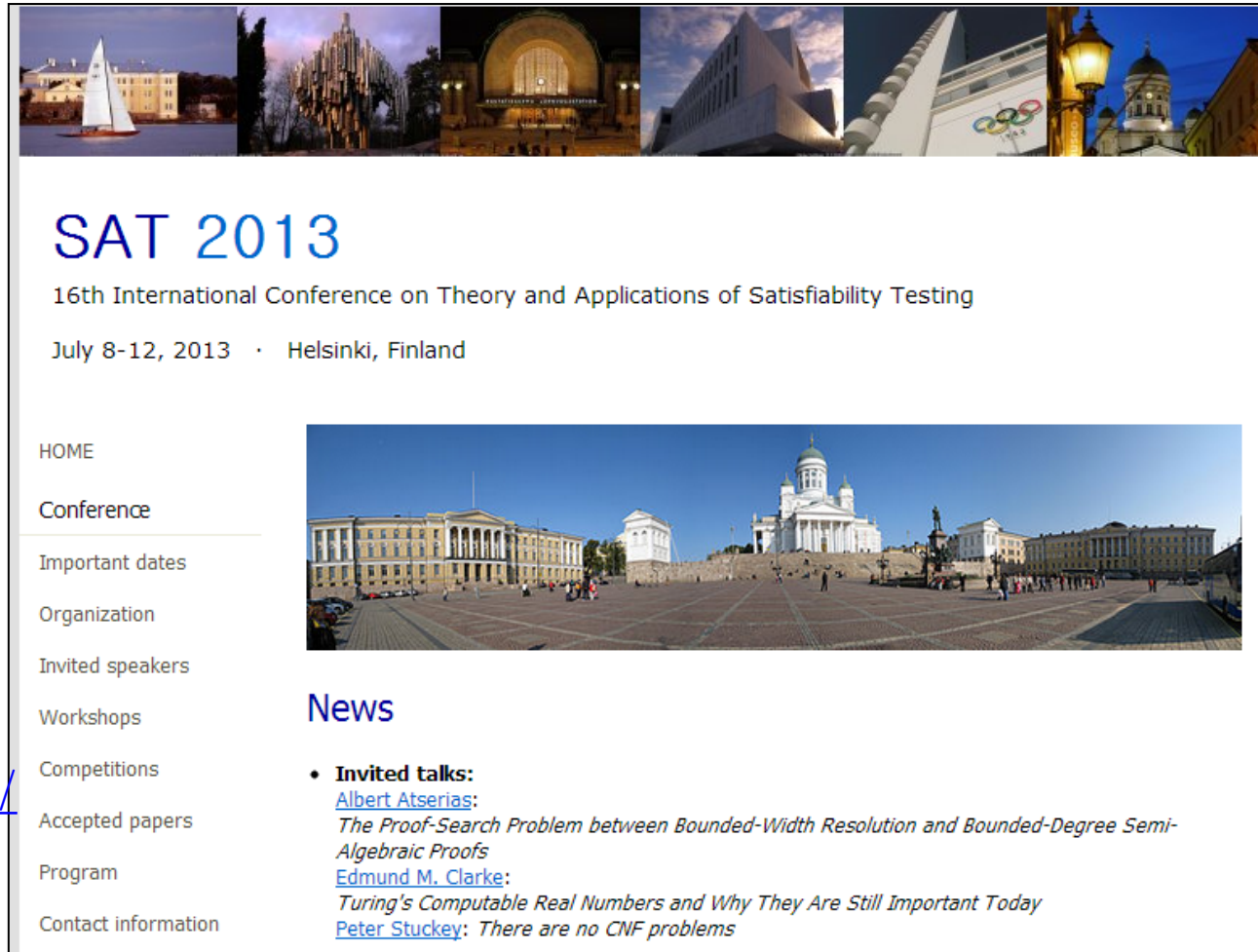
# Example


{p ∨ r} ∧{¬p ∨ ¬q ∨ r} ∧{p ∨¬r}
p=T
p=F
{T∨r} ∧{¬T ∨ ¬q ∨ r} ∧{T ∨ ¬r}
{F ∨ r} ∧{¬F ∨ ¬q ∨ r} ∧{F ∨ ¬r}
SIMPLIFY
SIMPLIFY
{¬q,r}
{r} ∧{¬r}
SIMPLIFY
{}

# SAT Solvers

- Most SAT solvers receives DIMACS CNF formulas
- Dozens of industry-strength SAT solvers available
  - MiniSAT
  - PicoSAT
  - SAT4J
  - borg-sat
  - clasp
  - sathys
  - tts
  - …

http://sat2013.cs.helsinki.fi/



SAT 2013

16th International Conference on Theory and Applications of Satisfiability Testing

July 8-12, 2013 · Helsinki, Finland

HOME

Conference

Important dates

Organization

Invited speakers

Workshops

Competitions

Accepted papers

Program

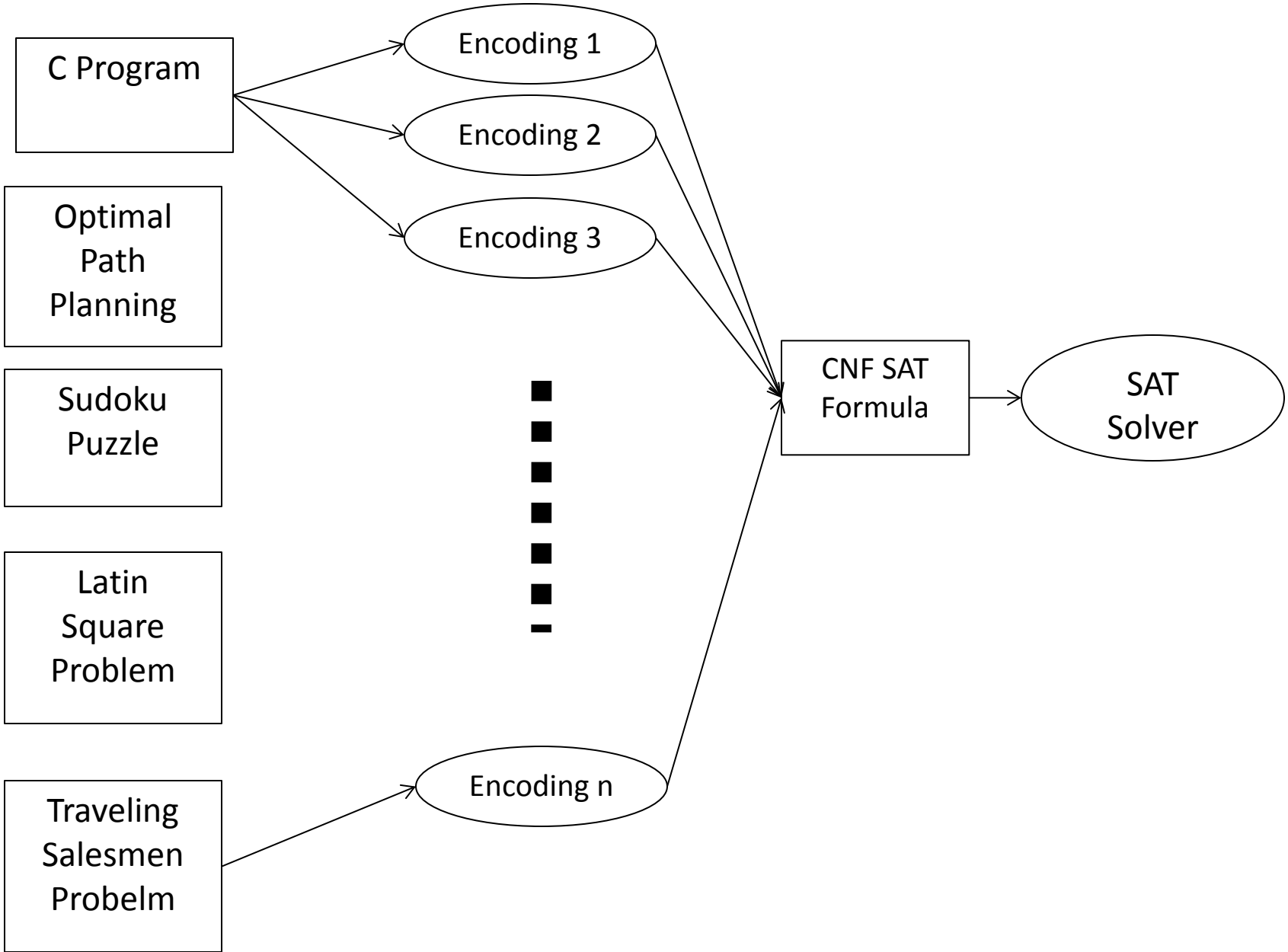
Contact information



News

- **Invited talks:**
  Albert Atserias:
  *The Proof-Search Problem between Bounded-Width Resolution and Bounded-Degree Semi-Algebraic Proofs*
  Edmund M. Clarke:
  *Turing's Computable Real Numbers and Why They Are Still Important Today*
  Peter Stuckey: *There are no CNF problems*

# Solving Various Problems using SAT Solver



# What is Sudoku ?

Problem

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 6 | 1 | | 2 | 5 | | |
| | 3 | 9 | | | | 1 | 4 | |
| | | | | 4 | | | | |
| 9 | | 2 | | 3 | | 4 | | 1 |
| | 8 | | | | | | 7 | |
| 1 | | 3 | | 6 | | 8 | | 9 |
| | | | | 1 | | | | |
| | 5 | 4 | | | | 9 | 1 | |
| | | 7 | 5 | | 3 | 2 | | |

Given a problem, the objectvie is to find a satisfying assignment w.r.t. Sudoku rules.

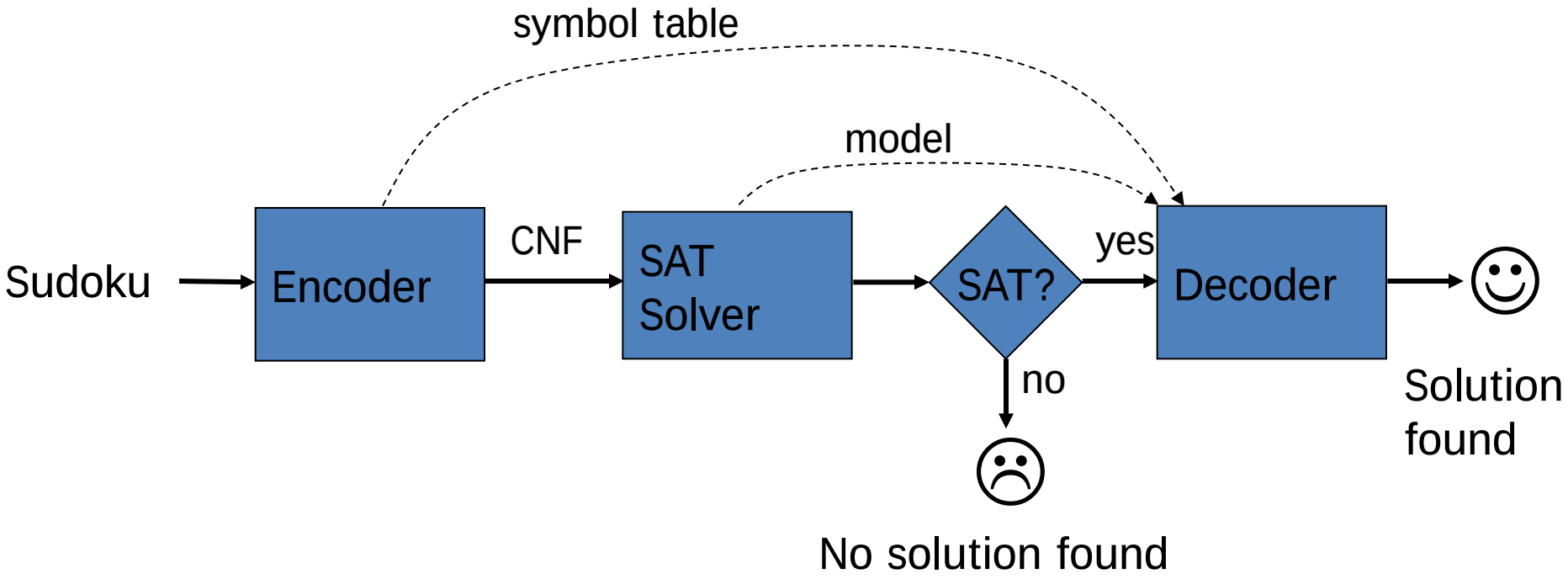
Solution

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 8 | 4 | 6 | 1 | 7 | 2 | 5 | 9 | 3 |
| 7 | 3 | 9 | 6 | 5 | 8 | 1 | 4 | 2 |
| 5 | 2 | 1 | 3 | 4 | 9 | 7 | 6 | 8 |
| 9 | 6 | 2 | 8 | 3 | 7 | 4 | 5 | 1 |
| 4 | 8 | 5 | 9 | 2 | 1 | 3 | 7 | 6 |
| 1 | 7 | 3 | 4 | 6 | 5 | 8 | 2 | 9 |
| 2 | 9 | 8 | 7 | 1 | 4 | 6 | 3 | 5 |
| 3 | 5 | 4 | 2 | 8 | 6 | 9 | 1 | 7 |
| 6 | 1 | 7 | 5 | 9 | 3 | 2 | 8 | 4 |

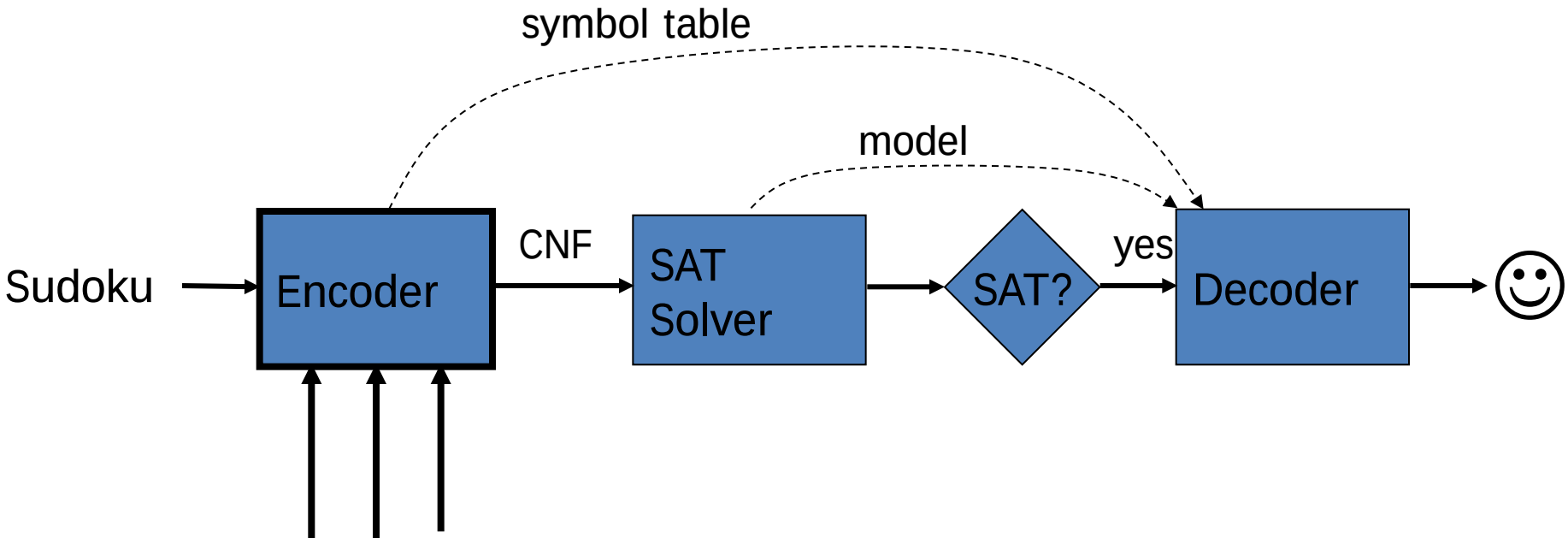
## Sodoku rules

- ✓ There is a number in each **cell**.
- ✓ A number appears once in each **row**.
- ✓ A number appears once in each **column**.
- ✓ A number appears once in each **block**.

# Sudoku as SAT Problem



# Previous Encodings




**Minimal** encoding [Lynce & Ouaknine, 2006]

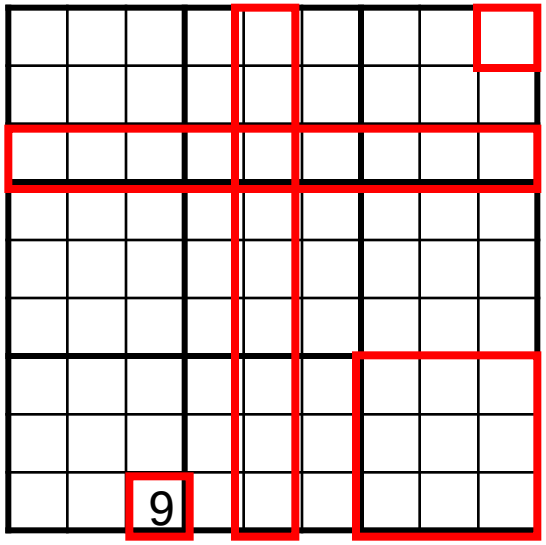
**Extended** encoding [Lynce & Ouaknine, 2006]

**Efficient** encoding [Weber, 2005]

# Encoding

- Kowledge compilation into a **target language**
  problem knowlege ➡ CNF

- Knowlede about Sudoku




A number appears once in each **cell**

A number appears once in each **row**

A number appears once in each **col**

A number appears once in each **block**

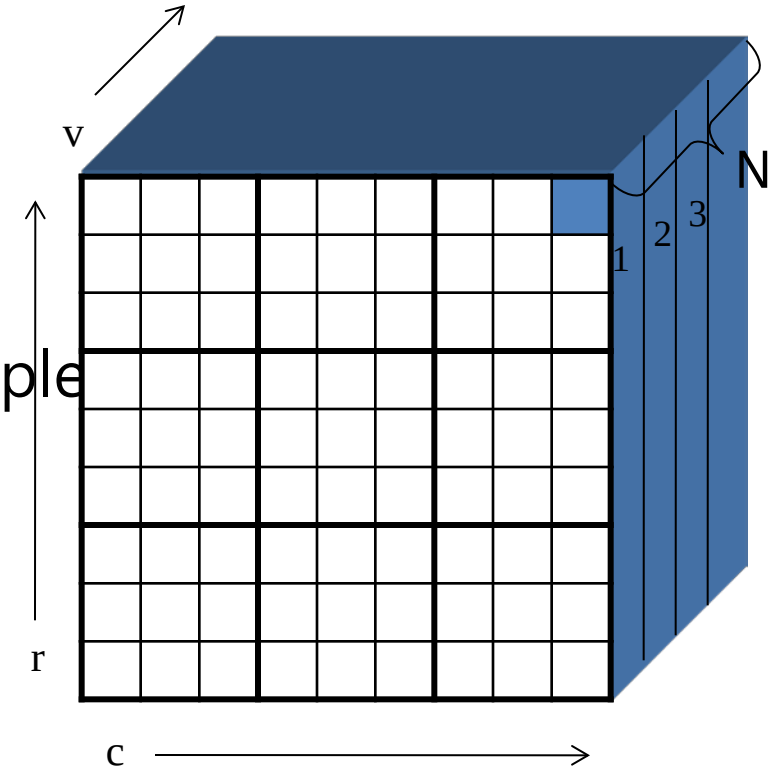
**rules** ➡ **CNF**

A pre-assigned number

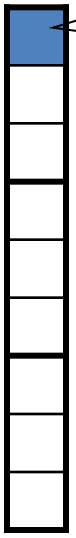
**facts** ➡ **CNF**

# Variables

- Each cell has one number from 1..N
  - [1,1]=1 or [1,1]=2 or …… or [1,1]=N
  - Each cell needs N boolean variables to consider all cases
- Total number of variables
  - $N^3$
- Boolean variable name as a triple
  - (r,c,v) (i.e., $x_{rcv}$ ) iff [r,c] = v
  - $\neg$(r,c,v) (i.e.,$\neg x_{rcv}$ ) iff [r,c] $\neq$ v



# Cell Rule ➔ CNF



A number appears once in each **cell**

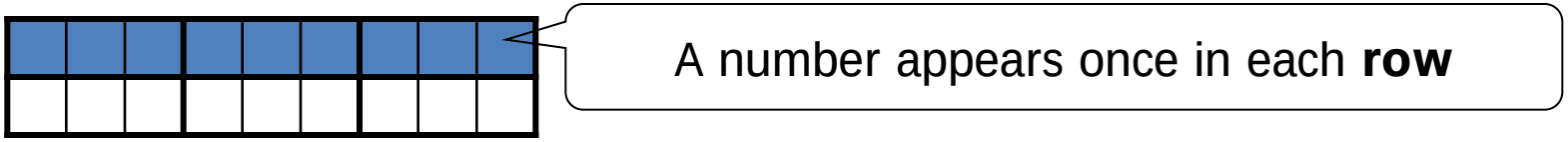
There is **at least** one number in each cell (definedness)

$$Cell_d = \wedge_{r=1}^{N} \wedge_{c=1}^{N} \vee_{v=1}^{N} (r, c, v)$$

There is **at most** one number in each cell (uniqueness)

$$Cell_u = \wedge_{r=1}^{N} \wedge_{c=1}^{N} \wedge_{v_i=1}^{N-1} \wedge_{v_j=v_i+1}^{N} \neg((r, c, v_i) \wedge (r, c, v_j))$$

# Row Rule ➔ CNF



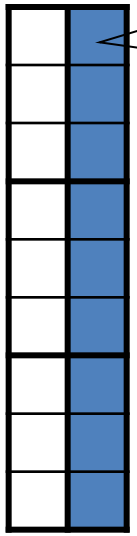

Each number appears **at least** once in each row (definedness)

$$Row_d = \wedge_{r=1}^{N} \wedge_{v=1}^{N} \vee_{c=1}^{N} (r, c, v)$$

Each number appears **at most** once in each row (uniqueness)

$$Row_u = \wedge_{r=1}^{N} \wedge_{v=1}^{N} \wedge_{c_i=1}^{N-1} \wedge_{c_j=c_i+1}^{N} \neg((r, c_i, v) \wedge (r, c_j, v))$$

# Column Rule ➔ CNF



A number appears once in each **column**

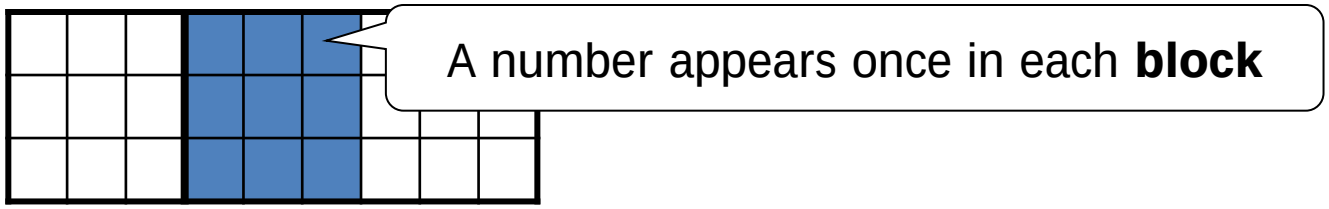
Each number appears **at least once** in each column (definedness)

$$Col_d = \wedge_{c=1}^{N} \wedge_{v=1}^{N} \vee_{r=1}^{N} (r, c, v)$$

Each number appears **at most** once in each column (uniqueness)

$$Col_u = \wedge_{c=1}^{N} \wedge_{v=1}^{N} \wedge_{r_i=1}^{N-1} \wedge_{r_j=r_i+1}^{N} \neg((r_i, c, v) \wedge (r_j, c, v))$$

# Block Rule ➔ CNF




Each number appears **at least** once in each block (definedness)

$$Block_d = \wedge_{r_{offs}=1}^{subN} \wedge_{c_{offs}=1}^{subN} \wedge_{v=1}^{N} \vee_{r=1}^{subN} \vee_{c=1}^{subN} ((r_{offs}-1)*subN+r,(c_{offs}-1)*subN+c,v)$$

where subN =3 for the above example

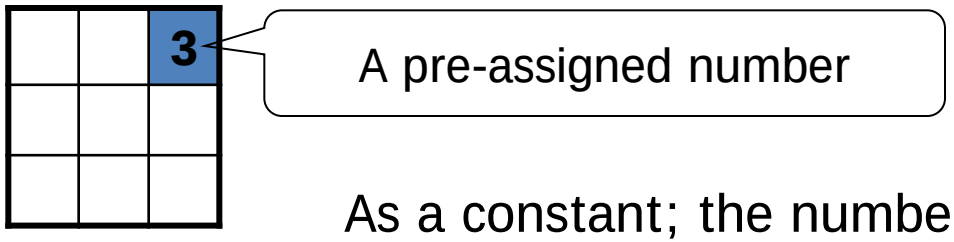
Each number appears **at most** once in each block (uniqueness)

$$Block_u = \wedge_{r_{offs}=1}^{subN} \wedge_{c_{offs}=1}^{subN} \wedge_{v=1}^{N} \wedge_{r=1}^{N} \wedge_{c=r+1}^{N}$$

$$\neg(((r_{offs}-1)*subN+(r \mod subN),(c_{offs}-1)*subN+(r \mod subN),v)$$

$$\wedge((r_{offs}-1)*subN+(c \mod subN),(c_{offs}-1)*subN+(c \mod subN),v))$$

# Pre-Assigned Fact ➔ CNF




As a constant; the number is never changed

It can be represented as a **unit clause**

$$Assigned = \wedge_{i=1}^{k}\{(r,c,a) \mid \exists_{1 \le a \le N} \bullet [r,c] = a\}$$

where $k$ is a number of pre-assigned numbers

# Previous Encodings

**Minimal encoding** [Lynce & Ouaknine, 2006]

$$\phi = Cell_d \cup Row_u \cup Col_u \cup Block_u \cup Assigned$$

sufficient to characterize the puzzle

**Extended encoding** [Lynce & Ouaknine, 2006]

$$\phi = Cell_d \cup Cell_u \cup Row_d \cup Row_u \cup Col_d \cup Col_u \cup Block_d \cup Block_u \cup Assigned$$

minimal encoding with redundant clauses

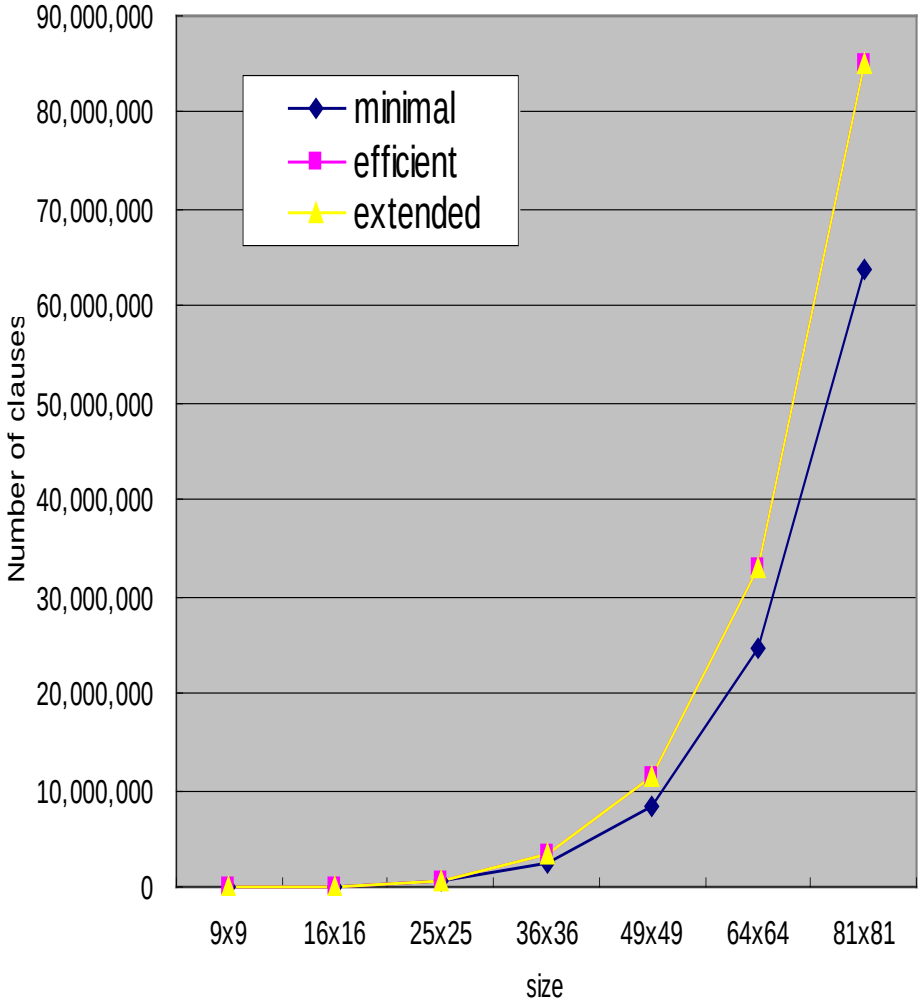
**Efficient encoding** [Weber, 2005]

$$\phi = Cell_d \cup Cell_u \cup Row_u \cup Col_u \cup Block_u \cup Assigned$$

between minimal encoding and extended encoding

# Exponential Growth in Clauses

| size | minimal | efficient | extended |
|---|---|---|---|
| 9x9 | 8829 | 11745 | 11988 |
| 16x16 | 92416 | 123136 | 123904 |
| 25x25 | 563125 | 750625 | 752500 |
| 36x36 | 2450736 | 3267216 | 3271104 |
| 49x49 | 8473129 | 11296705 | 11303908 |
| 64x64 | 24776704 | 33034240 | 33046528 |
| 81x81 | 63779481 | 85037121 | 85056804 |



# Experimental Results

| | | minimal encoding | | | efficient encoding | | | extended encoding | | |
|---|---|---|---|---|---|---|---|---|---|---|
| size | level | vars | clauses | time | vars | clauses | time | vars | clauses | time |
| 9x9 | easy | 729 | 8854 | 0.00 | 729 | 11770 | 0.00 | 729 | 12013 | 0.00 |
| 9x9 | hard | 729 | 8859 | 0.00 | 729 | 11775 | 0.00 | 729 | 12018 | 0.00 |
| 16x16 | easy | 4096 | 92520 | 0.10 | 4096 | 123240 | 0.09 | 4096 | 124008 | 0.01 |
| 16x16 | hard | 4096 | 92514 | 0.46 | 4096 | 123234 | 0.21 | 4096 | 124002 | 0.01 |
| 25x25 | easy | 15625 | 563417 | 9.07 | 15625 | 750917 | 17.48 | 15625 | 752792 | 0.07 |
| 25x25 | hard | 15625 | 563403 | time | 15625 | 750903 | time | 15625 | 752778 | 0.21 |
| 36x36 | easy | 46656 | 2451380 | time | 46656 | 3267860 | time | 46656 | 3271748 | 0.50 |
| 36x36 | hard | 46656 | 2451400 | time | 46656 | 3267880 | time | 46656 | 3271768 | 0.67 |
| 49x49 | easy | 117649 | 8474410 | time | 117649 | 11297986 | time | 117649 | 11305189 | 1.47 |
| 64x64 | easy | 262144 | 24779088 | stack | 262144 | 33036624 | stack | 262144 | 33048912 | stack |
| 81x81 | easy | 531441 | 63783464 | stack | 531441 | 85041104 | stack | 531441 | 85060787 | stack |

# Experimental Results



| | | minimal encoding | | | efficient encoding | | | extended encoding | | |
|---|---|---|---|---|---|---|---|---|---|---|
| size | level | vars | clauses | time | vars | clauses | time | vars | clauses | time |
| 9x9 | easy | 729 | 8854 | 0.00 | 729 | 11770 | | 29 | 12013 | 0.00 |
| 9x9 | hard | 729 | 8859 | 0.00 | 729 | 11775 | | 29 | 12018 | 0.00 |
| 16x16 | easy | 4096 | 92520 | 0.10 | 4096 | 123240 | | 96 | 124008 | 0.01 |
| 16x16 | hard | 4096 | 92514 | 0.46 | 4096 | | | 4002 | | 0.01 |
| 25x25 | easy | 15625 | 563417 | 9.07 | 15625 | 750917 | 17.48 | 15625 | 752792 | 0.07 |
| 25x25 | hard | 15625 | 563403 | time | 15625 | 750903 | time | 15625 | 752778 | 0.21 |
| 36x36 | easy | 46656 | 2451380 | | 6656 | 3267860 | time | 46656 | 3271748 | 0.50 |
| 36x36 | hard | 46656 | 2451400 | | 6656 | 3267880 | time | 46656 | 3271768 | 0.67 |
| 49x49 | easy | 117649 | 8474410 | | 7649 | 11297986 | time | 117649 | 11305189 | 1.47 |
| 64x64 | easy | 262144 | | | | 6624 | stack | 262144 | 33048912 | stack |
| 81x81 | easy | 531441 | 63783464 | stack | 531441 | 85041104 | stack | 531441 | 85060787 | stack |

Solution found

No solution found