Where the hard problems are

Toby Walsh

Cork Constraint Computation Centre

http://4c.ucc.ie/~tw

1/25/2008

Where the hard problems are

- That’s easy
  - **Passing your finals**
  - **Finding a job**
  - **…**

Where the hard problems are

- That's easy
  - **Passing your finals**
  - **Finding a job**
  - **…**
- Where are the hard computational problems?
  - **Computational complexity**
  - **Phase transition behaviour**
  - **Exploiting structure**

Computational complexity

- Study of "problem hardness"
  - **Typically worst case**
- Big O analysis
  - **Sorting is easy, O(n logn)**
  - **Chess and GO are hard, EXP-time**
    - "Can I be sure to win?"
    - Need to generalize problem to n by n board

Computational complexity

- Study of "problem hardness"
  - **Typically worst case**
- Big O analysis
  - **Sorting is easy, O(n logn)**
  - **Chess and GO are hard, EXP-time**
    - "Can I be sure to win?"
    - Need to generalize problem to n by n board

Where do things start getting hard?

Computational complexity

- **Hierarchy of complexity classes**
  - Polynomial (P), NP, PSpace, ….
  - NP-complete problems mark boundary of tractability
    - **No known polynomial time algorithm**
    - **Though open if P=/=NP**

Aside: What happens at the top of this hierarchy? Some algorithms may NEVER terminate, e.g. halting problem is undecidable

NP-complete problems

- Non-deterministic Polynomial time
  - **If I guess a solution, I can check it in polynomial time**
  - **But no known easy way to guess solution correctly!**
- Complete
  - **Representative of all problems in this class**
  - **If this problem can be solved in polynomial time, all problems in the class can be solved**
  - **Any NP-complete problem can be mapped into any other**

NP-complete problems

- Many examples
  - **Propositional satisfiability (SAT)**
  - **Graph colouring**
  - **Travelling salesperson problem**
  - **Exam timetabling**
  - **...**

Satisfiability

- 1st problem shown to be NP-complete [Cook 71]
- Decision problem
  - **Given a clausal formula, is it satisfiable?**

    E.g. (X or Y) & (-Y or Z) is satisfied by
    X=true, Y=false …

  - **Clausal = ANDs of some ORs**

Satisfiability

- In NP (easy to prove)
  - **You give me an assignment, I can check it satisfies formula in polynomial time**

- Complete for NP (hard to prove)
  - **Any problem in NP can be reduced to SAT in polynomial time**

Diplomatic problem

- Embassy ball.
  - **King wants to invite PERU or exclude QATAR**
  - **Queen wants to invite QATAR or ROMANIA**
  - **Kings wants to snub ROMANIA or PERU**

  Who can we invite?

Diplomatic problem

- Embassy ball.
  - **King wants to invite PERU or exclude QATAR**
  - **Queen wants to invite QATAR or ROMANIA**
  - **Kings wants to snub ROMANIA or PERU**

  (P or -Q) & (Q or R) & (-R or -P)

Diplomatic problem

- Embassy ball.

  King wants to invite PERU or exclude QATAR

  Queen wants to invite QATAR or ROMANIA

  Kings wants to snub ROMANIA or PERU

  (P or -Q) & (Q or R) & (-R or -P) is satisfied by

  P=true, Q=true, R=false

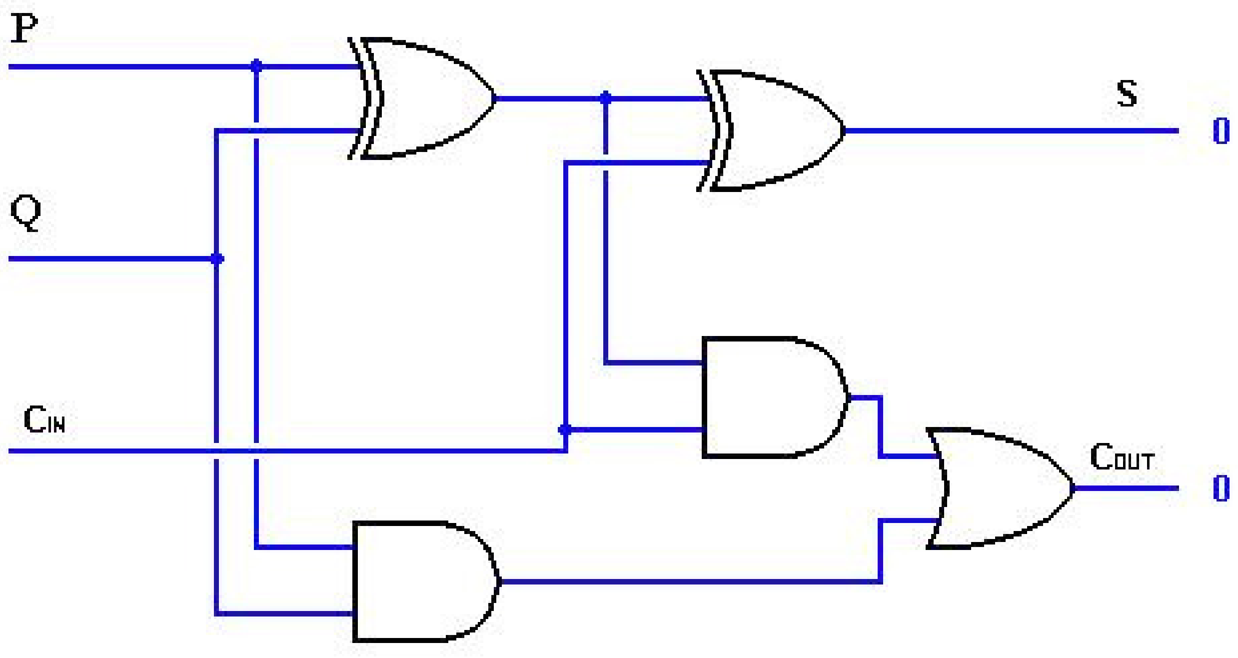
Diplomatic problem

- Embassy ball.
  - **King wants to invite PERU or exclude QATAR**
  - **Queen wants to invite QATAR or ROMANIA**
  - **Kings wants to snub ROMANIA or PERU**

  (P or -Q) & (Q or R) & (-R or -P) is satisfied by
  P=true, Q=true, R=false and by
  P=false, Q=false, R=true

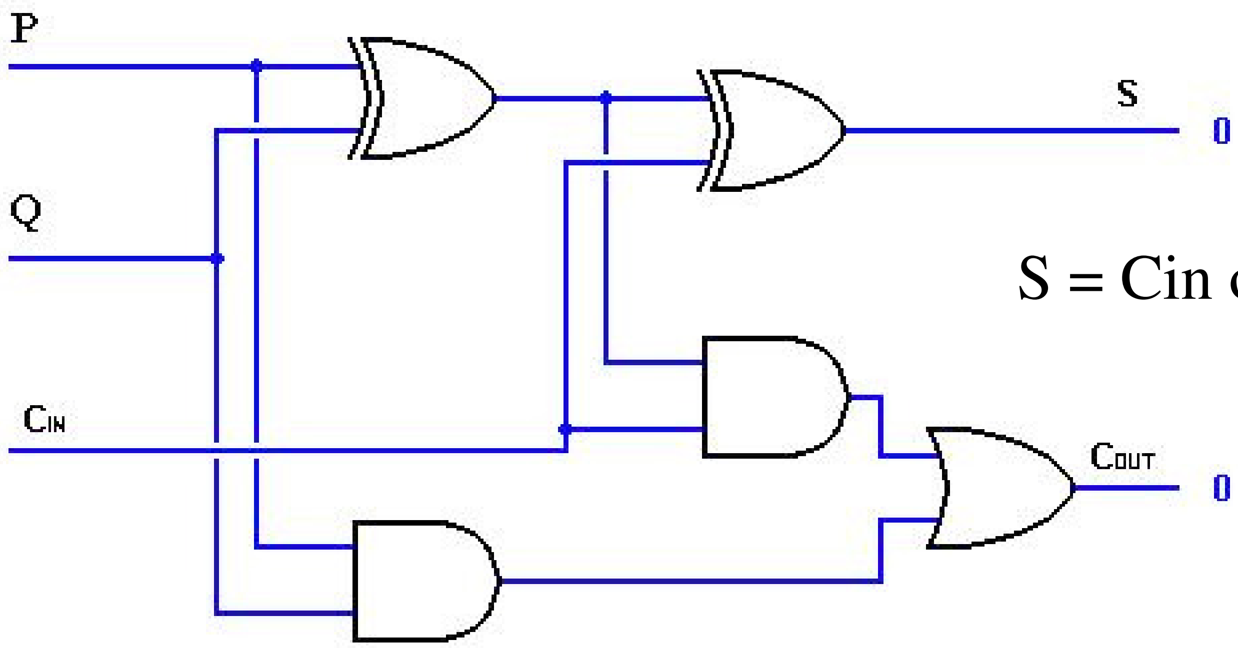
Other applications of SAT

- Hardware verification



Other applications of SAT

- Hardware verification



S = Cin or (P or Q), …

Other applications of SAT

- Scheduling jobs in a factory
  - **Set of jobs, resource constraints, start and due dates, …**
  - **Just need to sequence jobs**
    - Xij = true iff Job i occurs before Job j

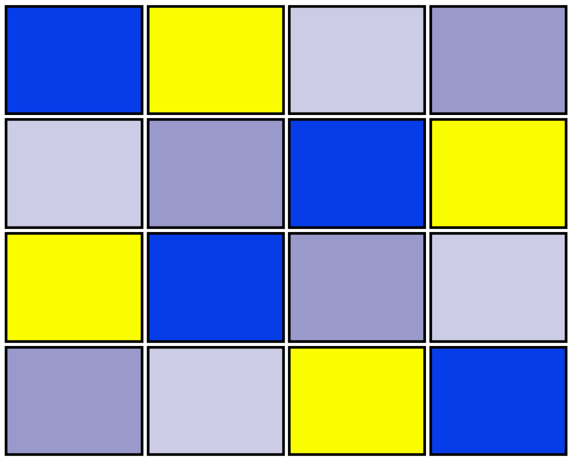
    …

Other applications of SAT

- Design theory
  - **Experimental design, Latin squares with special properties**
  - **Open problems solved by encoding problems into SAT**
    - [Fujita, Slaney, Bennet, 1993]
    - Best paper at the International Joint Conference on Artificial Intelligence, 1993

Latin square

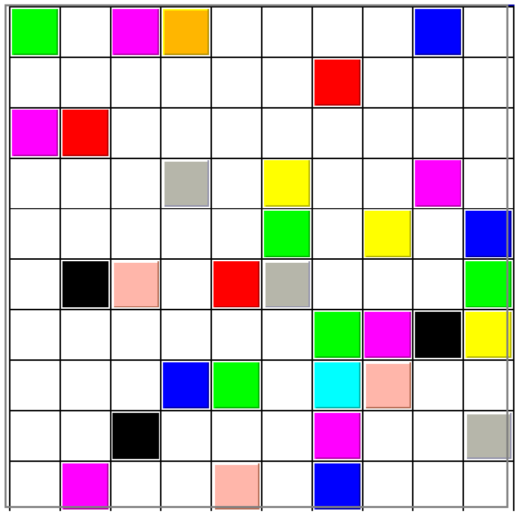
- Also called “quasigroup”
- Each colour occurs once in each row and column



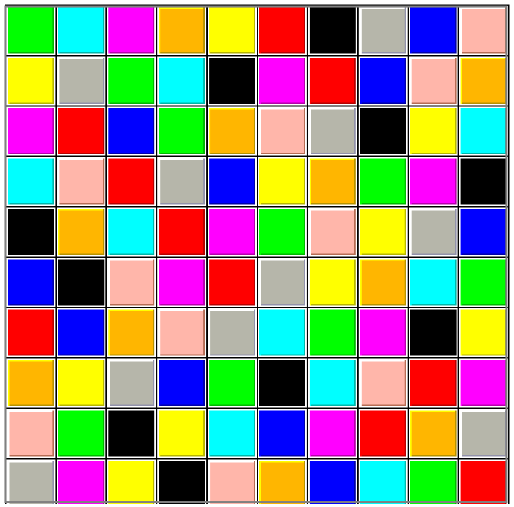
Quasigroup or Latin Square
(Order 4)

Latin squares

- Can we complete a partially coloured Latin square?
  - **NP-complete problem**
  - **Applications in telecommunications**



32% preassignment



SAT model for Latin square

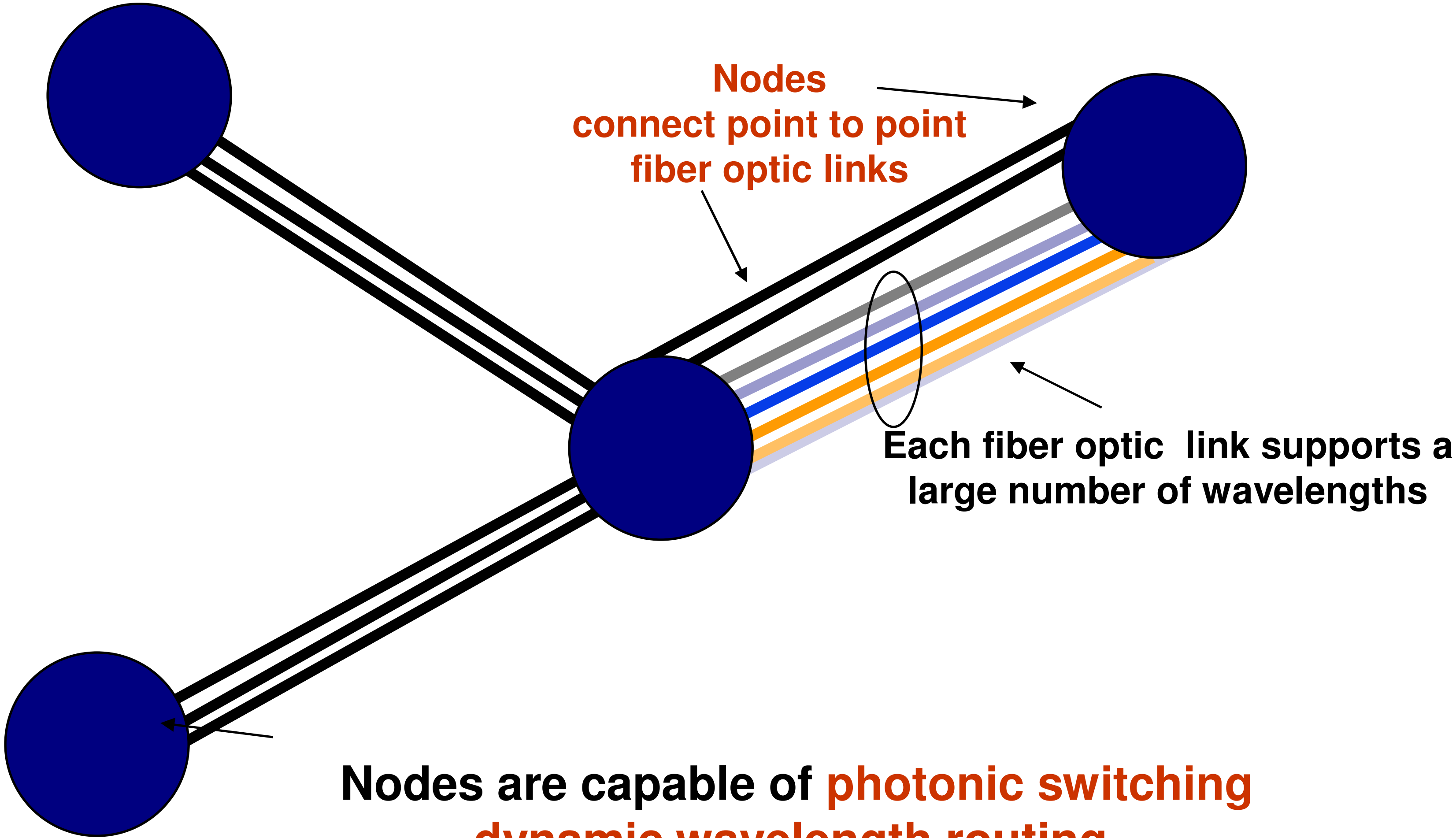
- What are the variables?
  - **Xijk is true if square (i,j) has the colour k**

- What are the constraints
  - **Row constraints**

    (-Xijk or -Xij+1k), ….
  - **Column constraints**

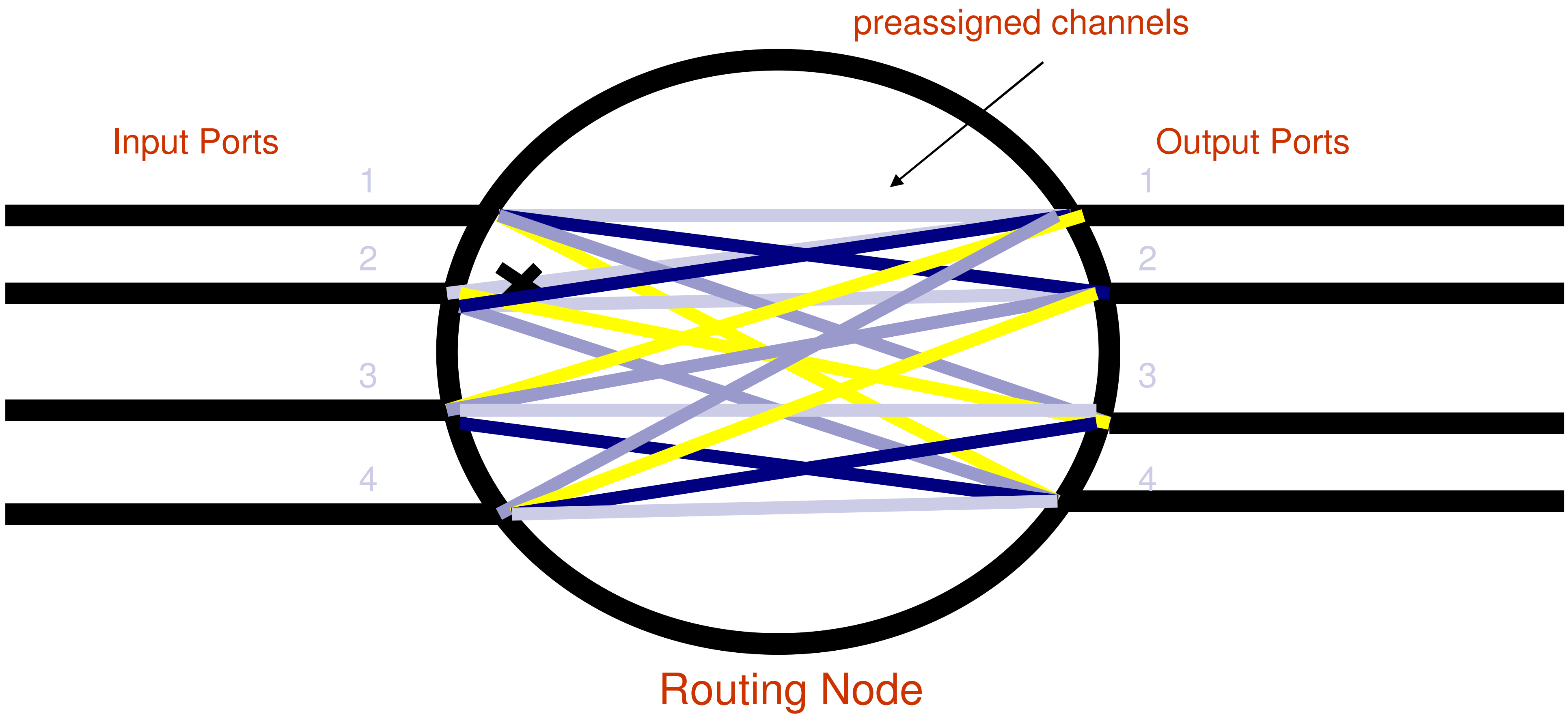
    (-Xijk or -Xi+1jk), …

Fibre-optic routing



Nodes are capable of photonic switching
--dynamic wavelength routing --
which involves the setting of the wavelengths.

Routing in Fiber Optic Networks

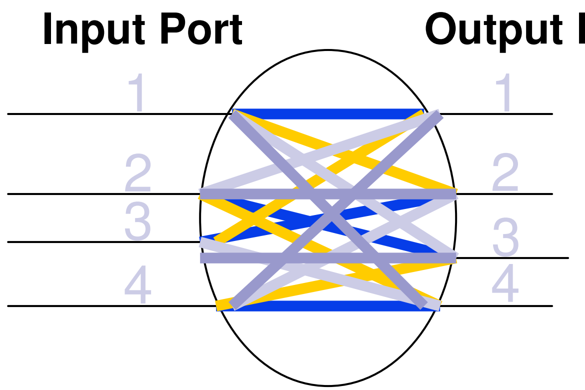


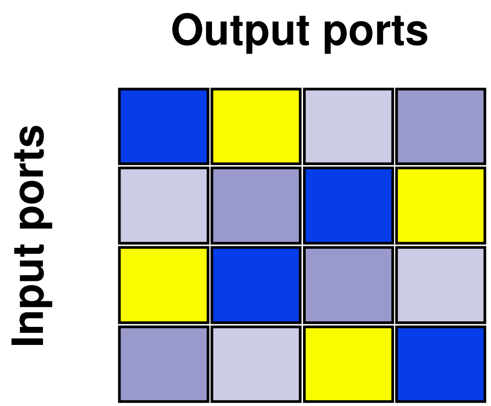
How can we achieve conflict-free routing in each node of the network?

Dynamic wavelength routing is a NP-hard problem.

Routing = Latin square problem

- Each channel cannot be repeated on input ports
- Each channell cannot be repeated on output ports





Orthogonal Latin squares

- Find a pair of Latin squares
  - **Every cell has a different pair of elements**
- Generalized form:
  - **Find a set of m Latin squares**
  - **Each possible pair is orthogonal**

Orthogonal Latin squares

1	2	3	4		1	2	3	4
2	1	4	3		3	4	1	2
3	4	1	2		4	3	2	1
4	3	2	1		2	1	4	3

11	22	33	44
23	14	41	32
34	43	12	21
42	31	24	13

- Two 4 by 4 Latin squares
- No pair is repeated

History

- Orthogial Latin squares were introduced by Euler in 1783
  - **Also called Graeco-Latin or Euler squares**

- No orthogonal Latin square of order 2
  - **There are only 2 (non)-isomorphic Latin squares of order 2 and they are not orthogonal**

History

- Euler conjectured in 1783 that there are no orthogonal Latin squares of order 4n+2
  - **Constructions exist for 4n and for 2n+1**
  - **Took till 1900 to show conjecture for n=1**
  - **Took till 1960 to show false for all n>1**

History

- 6 by 6 problem known as the 36 officer problem

“… Can a delegation of six regiments, each of which sends a colonel, a lieutenant-colonel, a major, a captain, a lieutenant, and a sub-lieutenant be arranged in a regular 6 by 6 array such that no row or column duplicates a rank or a regiment?”

More background

- Lam's problem
  - Existence of finite projective plane of order 10
  - Equivalent to set of 9 mutually orthogonal Latin squares of order 10
  - In 1989, this was shown not to be possible after 2000 hours on a Cray (and some major maths)
- Applications in experimental design
  - To ensure no dependency between independent variables

A simple SAT model

- What are the variables?
  - Xijkl = true if pair (i,j) is in row k column l, false otherwise

- What are the constraints?
  - Lots of them!
  - Latin square constraints
  - Orthogonality constraints

A simple SAT model

- Orthogonal Latin square has lots of symmetry
  - **Permute the rows**
  - **Permute the cols**
  - **Permute the numbers 1 to n in each square**
- How can we eliminate such symmetry?

Symmetry removal

- Fix first row
  11 22 33 …

- Fix first column
  11
  23
  32
  ..

- Eliminates all this symmetry?

See Ian Gent or Steve Linton if this interests you!

SAT algorithms

- How do we test if a problem is SAT or not?
  - **Complete methods**
    - Return “Yes” if SATisfiable
    - Return “No” if UNSATisfiable
  - **Incomplete methods**
    - If return “Yes”, problem is SATisfiable
    - Otherwise timeout/run forever, problem can be SAT or UNSAT

Complete SAT algorithms

A	B	C	D	$A \cdot B$	$C \cdot D$	$\overline{A} \cdot C \cdot \overline{D}$	Q
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	1
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	0	0	0	1	1
0	1	1	1	0	1	0	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0
1	0	1	0	0	0	0	0
1	0	1	1	0	1	0	1
1	1	0	0	1	0	0	1
1	1	0	1	1	0	0	1
1	1	1	0	1	0	0	1
1	1	1	1	1	1	0	1

- Truth tables

Complete SAT algorithms

A	B	C	D	$A \cdot B$	$C \cdot D$	$\overline{A} \cdot C \cdot \overline{D}$	Q
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	1
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	0	0	0	1	1
0	1	1	1	0	1	0	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0
1	0	1	0	0	0	0	0
1	0	1	1	0	1	0	1
1	1	0	0	1	0	0	1
1	1	0	1	1	0	0	1
1	1	1	0	1	0	0	1
1	1	1	1	1	1	0	1

- Truth tables
  - **Exponential time to construct**

Complete SAT algorithms

A	B	C	D	$A \cdot B$	$C \cdot D$	$\overline{A} \cdot C \cdot \overline{D}$	Q
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	1
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	0	0	0	1	1
0	1	1	1	0	1	0	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0
1	0	1	0	0	0	0	0
1	0	1	1	0	1	0	1
1	1	0	0	1	0	0	1
1	1	0	1	1	0	0	1
1	1	1	0	1	0	0	1
1	1	1	1	1	1	0	1

- Truth tables
  - **Exponential time to construct**
  - **Exponential space**

Complete SAT algorithms

- Davis Putnam algorithm
  - **Due to Davis, Logemann and Loveland from 1962**
  - **Original algorithm by Davis & Putnam in 1960 could use exponential space**
  - **DLL traded time for space to give linear space algorithm**

Complete SAT algorithms

- Davis Putnam algorithm
  - **Consider**

    (X or Y) & (-X or Z) & (-Y or Z) & …
  - **Basic idea**
    - Try X=true

Complete SAT algorithms

- Davis Putnam algorithm
  - **Consider**

    (X or Y) & (-X or Z) & (-Y or Z) & …
  - **Basic idea**
    - Try X=true
    - Remove clauses which must be satisfied

Complete SAT algorithms

- Davis Putnam algorithm
  - Consider

    (-X or Z) & (-Y or Z) & …
  - Basic idea
    - Try X=true
    - Remove clauses which must be satisfied
    - Simplify clauses containing -X

Complete SAT algorithms

- Davis Putnam algorithm
  - **Consider**

    (Z) & (-Y or Z) & …

  - **Basic idea**
    - Try X=true
    - Remove clauses which must be satisfied
    - Simplify clauses containing -X
    - Can now deduce that Z must be true

Complete SAT algorithms

- Davis Putnam algorithm
  - **Consider**

    (Z) & (-Y or Z) & …
  - **Basic idea**
    - Try X=true
    - Remove clauses which must be satisfied
    - Simplify clauses containing -X
    - Can now deduce that Z must be true
    - At any point, may have to backtrack and try X=false instead

Complete SAT algorithms

- Davis Putnam algorithm
  - **Branching heuristics**
    - Which variable to assign next?
    - Which value (true/false) to try first?
  - **Other refinements**
    - Clause learning
    - Intelligence backtracking
    - Implementation tricks
  - **Worst-case complexity**
    - O(1.696^n)

Incomplete SAT algorithms

- Most fit a very simple schema
  - **Randomly guess truth assignment**
  - **Repeat**
    - Pick a variable
    - Flip its truth value

Incomplete SAT algorithms

- Most fit a very simple schema
  - **Randomly guess truth assignment**
  - **Repeat**
    - Pick a variable
    - Flip its truth value
- Heuristics to pick variable
    - Any var in UNSAT clause
    - Var that maximizes number of SAT clauses if flipped
    - Var in UNSAT clause flipped longest time ago …

Other SAT algorithms

- All kinds of exotic approaches have also been tried
  - **Genetic algorithms**
  - **Ant algorithms**
  - **DNA algorithms**
  - **Quantum algorithms**
  - **…**

k-SAT

- Subclass of SAT problem
  - **Exactly k variables in each clause**
    (P or -Q) & (Q or R) & (-R or -P) is in 2-SAT

k-SAT

- Subclass of SAT problem
  - **k variables in each clause**
    (P or -Q) & (Q or R) & (-R or -P) is in 2-SAT
- 3-SAT is NP-complete
  - **SAT can be reduced to 3-SAT**

k-SAT

- Subclass of SAT problem
  - **k variables in each clause**
    (P or -Q) & (Q or R) & (-R or -P) is in 2-SAT
- 3-SAT is NP-complete
  - **SAT can be reduced to 3-SAT**
    (P or Q or R or S) = (P or Q or T) & T iff (R or S)
    = (P or Q or T) & (-T or R or S)
    & (T or -R) & (T or -S)

k-SAT

- Subclass of SAT problem
  - **k variables in each clause**

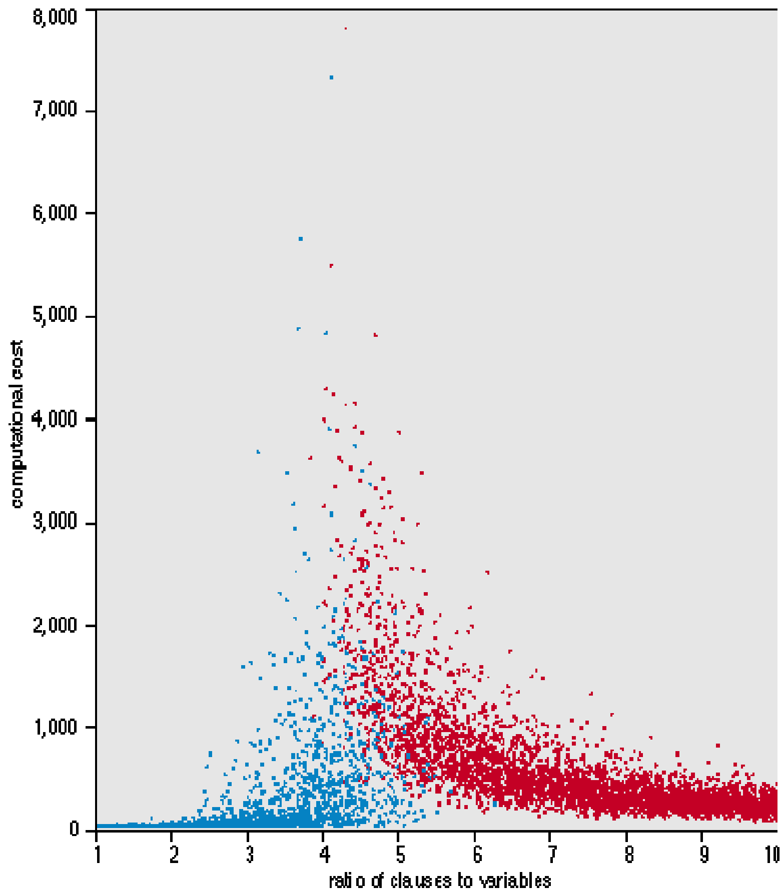
    (P or -Q) & (Q or R) & (-R or -P) is in 2-SAT
- 3-SAT is NP-complete
  - **SAT can be reduced to 3-SAT**
- 2-SAT is in P
  - **Exists linear time algorithm!**

Adding extra var to each clause makes problems hard … more on this later in the lectures!

3-SAT

- **Where are the hard 3-SAT problems?**
- **Sample randomly generated 3-SAT**
  - Fix number of clauses, l
  - Number of variables, n
  - By definition, each clause has 3 variables
  - Generate all possible clauses with uniform probability

Random 3-SAT



- Which are the hard instances?
  - **around *l/n* = 4.3**

What happens with larger problems?

Why are some dots red and others blue?

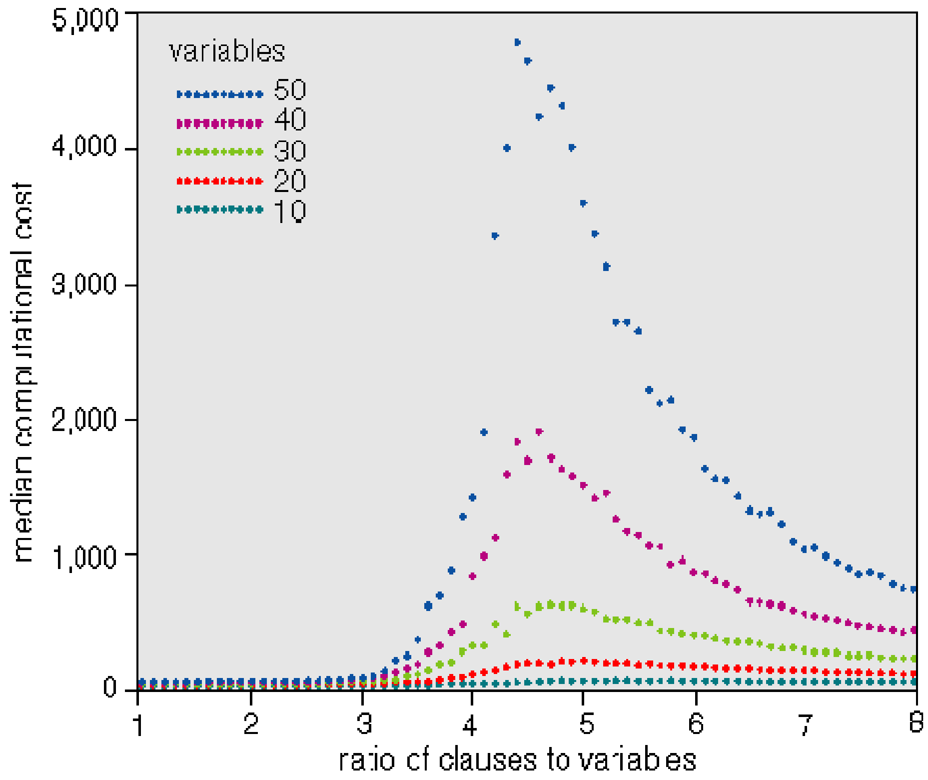
This is a so-called "phase transition"

Random 3-SAT

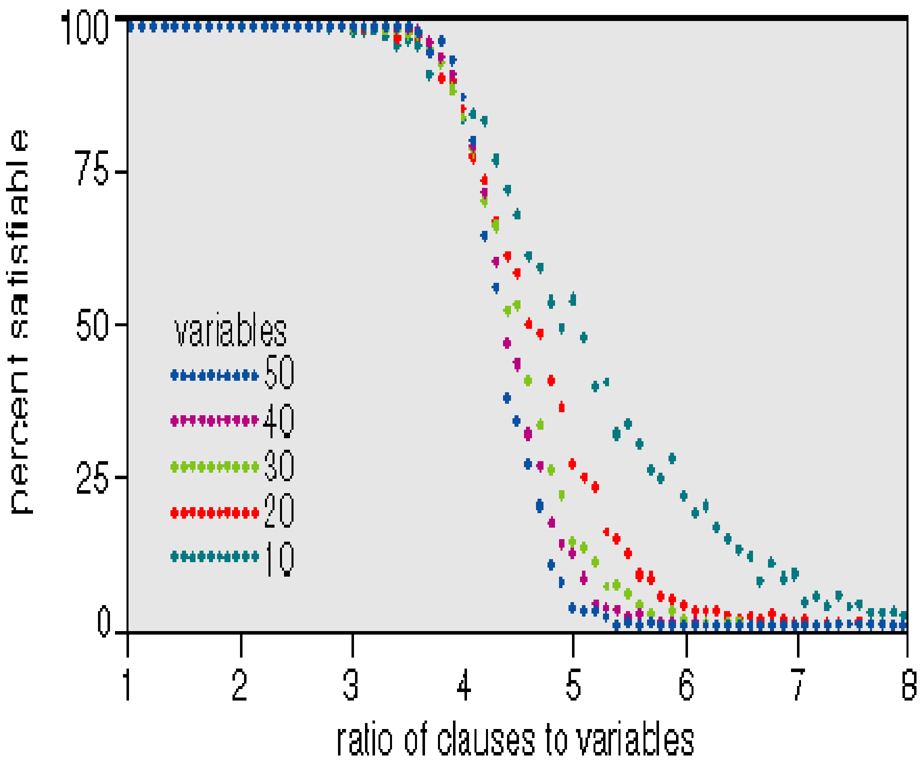
- Varying problem size, *n*

- Complexity peak appears to be largely invariant of algorithm
  - **complete algorithms like Davis-Putnam**
  - **Incomplete methods like local search**

What's so special about 4.3?



Random 3-SAT



- Complexity peak coincides with satisfiability transition
  - **l/n < 4.3 problems under-constrained and SAT**
  - **l/n > 4.3 problems over-constrained and UNSAT**
  - **l/n=4.3, problems on "knife-edge" between SAT and UNSAT**

Conclusions

- SAT is a useful problem class
  - **Theoretical importance**
  - **Practical value**

- Hard problems often turn up at a phase transition
  - **Next lecture: a closer look at this phenomenon!**