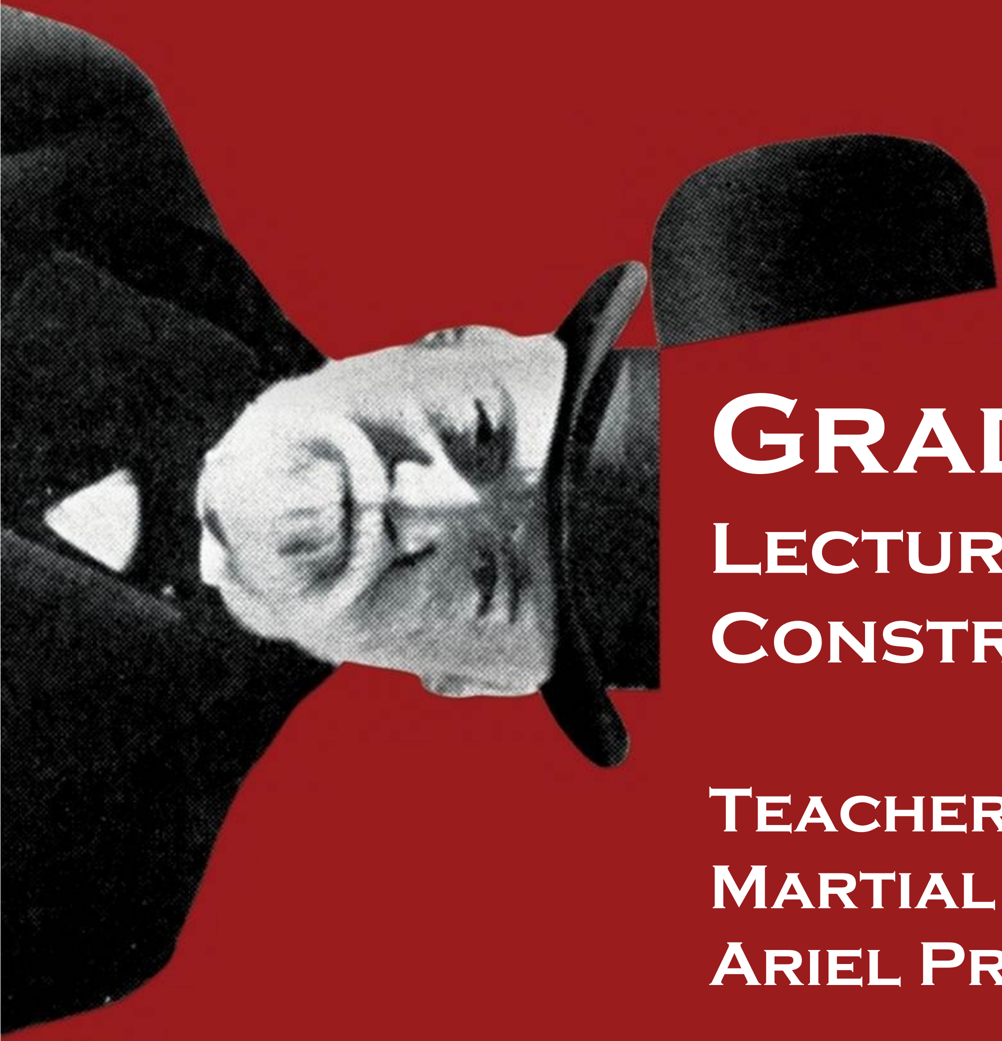


Graduate AI

Lecture 14: Constraint Satisfaction 2

Teachers:
Martial Hebert
Ariel Procaccia (this time)

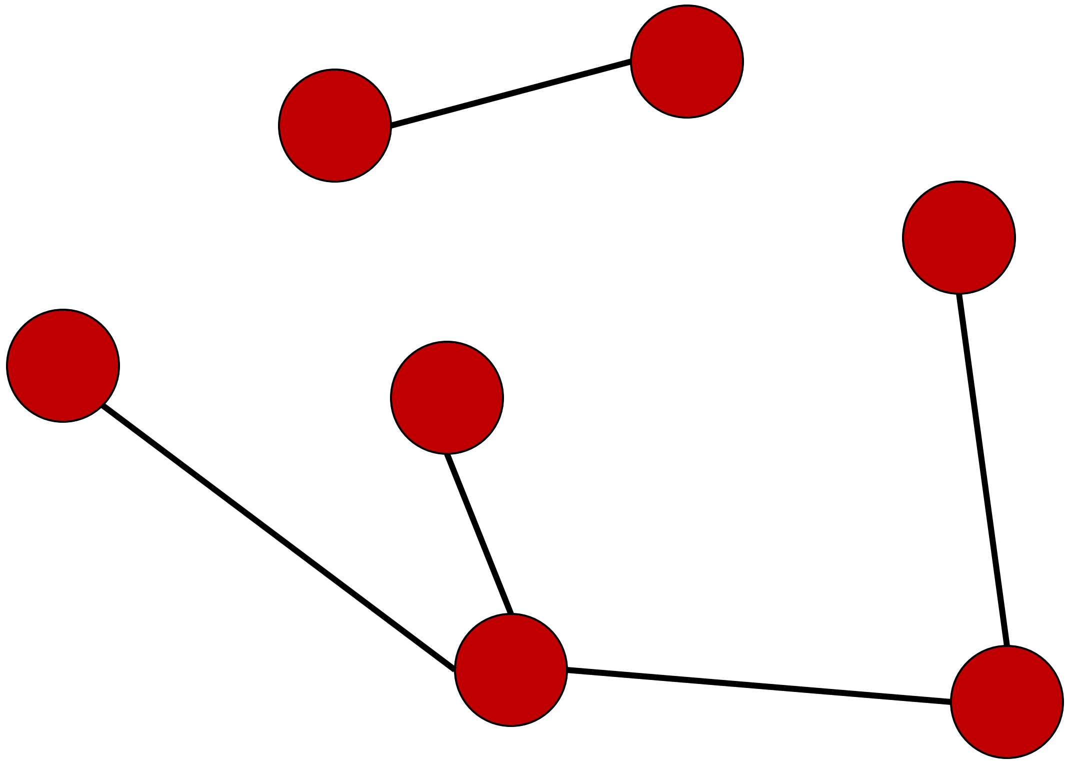
Reminder

- CSPs consist of:
  - Variables
  - Domains
  - Constraints: legal tuples of values for subsets of variables
- Goal: complete and consistent assignment
- Example: graph coloring

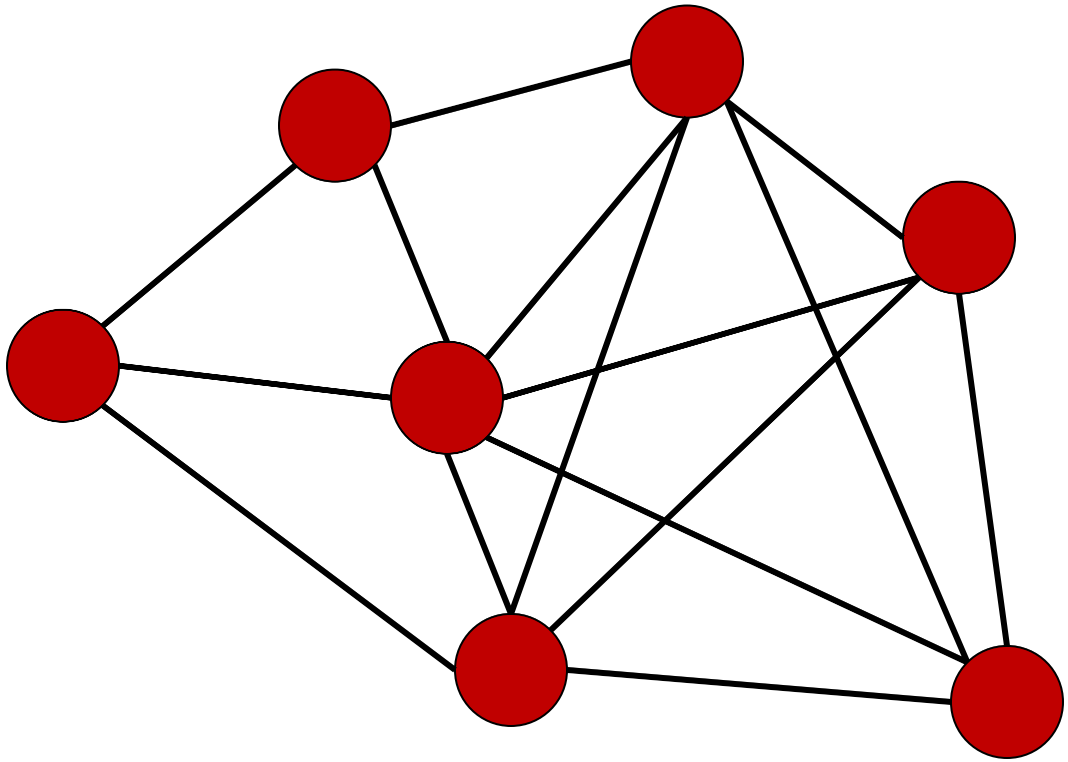
How hard are CSPs?

- In theory, solving a general CSP is NP-c
  - Obviously in NP
  - Captures graph coloring so NP-hard
- In practice, CSPs are often easy to solve
- Where are the hard problems?
- Identify **order parameter** to predict problem difficulty

Is this graph 4-colorable?



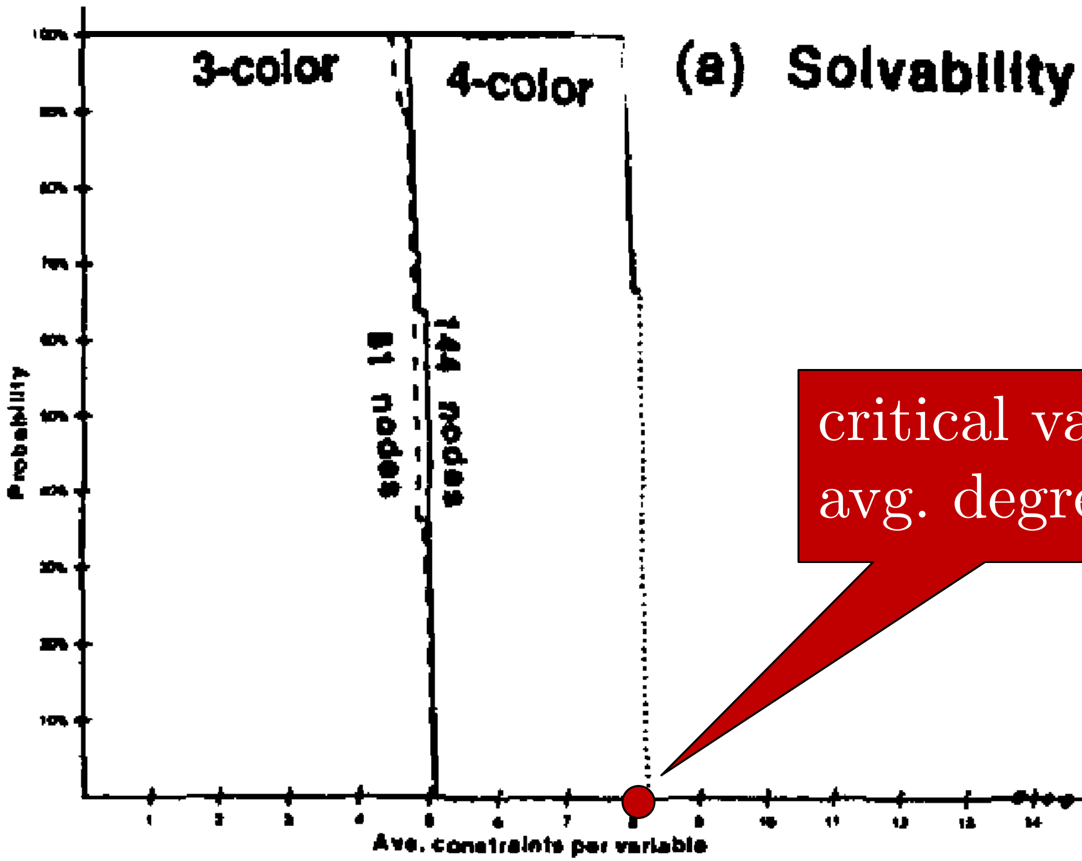
Is this graph 4-colorable?



Average degree

- Order parameter for graph coloring: average degree $= 2|E|/|V|$
- For a random graph, what is the probability of being colorable, as a function of the average degree?
- Should be 1 at $x=0$ and go down to 0

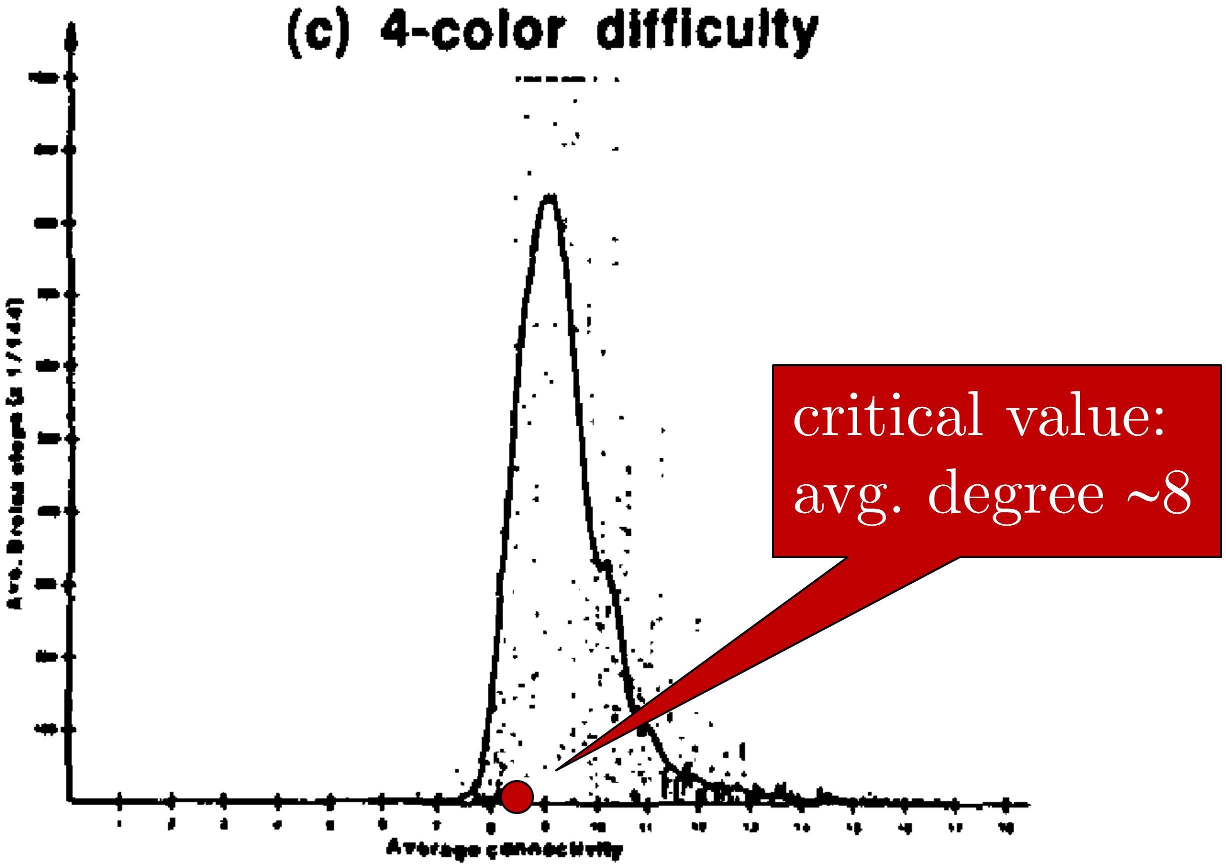
Phase transition



critical value:
avg. degree ~8

Cheeseman et al., IJCAI 1993

Peak in Difficulty



Cheersman et al., IJCAI 1993

Coincidence?

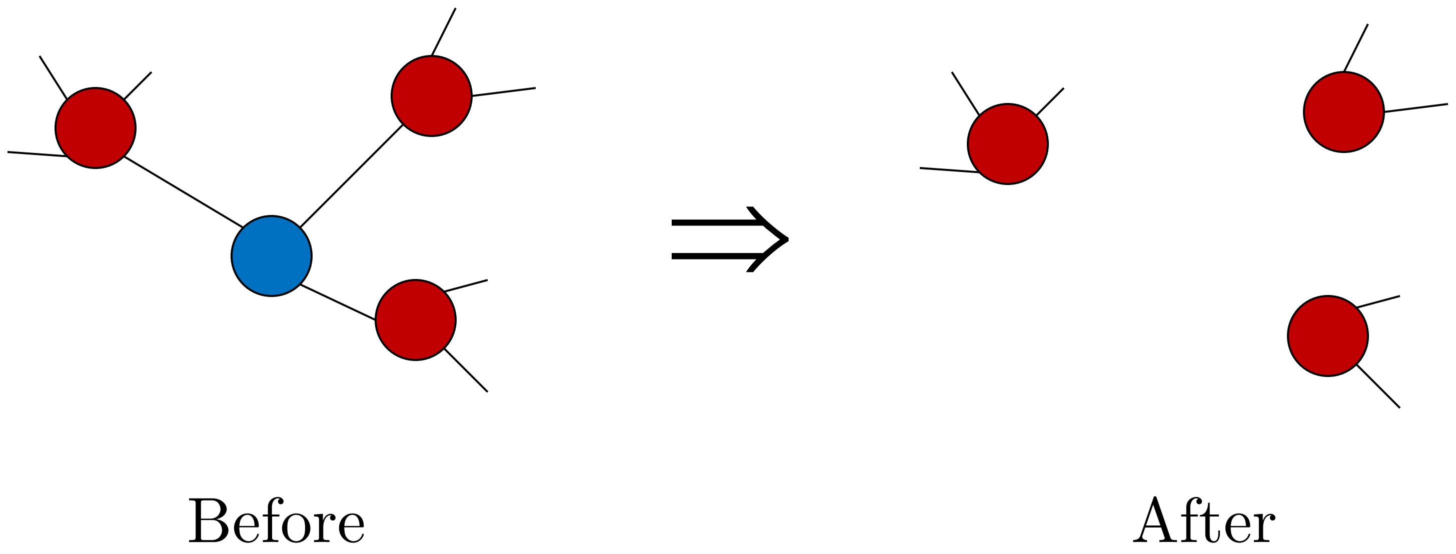
- Algorithm used: backtracking search with the heuristics we discussed
- Graph coloring is most difficult around the **critical value** of the order parameter
- In that region problems are neither underconstrained nor overconstrained

Generating hard graphs

- We want to test our CSP solvers with hard problems!
- Example: graph coloring
- First, reduce the graph using operators shown on next slide
- Second, concentrate on graphs with avg. degree around the critical value

Reduction operators

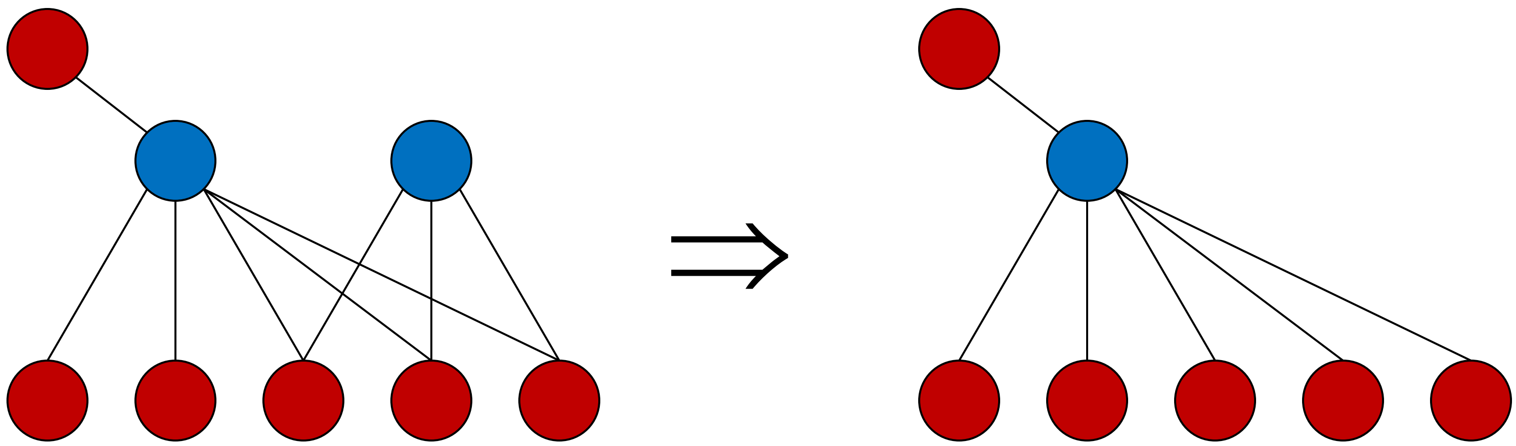
Underconstrained



Examples shown for 4-coloring

Reduction operators

Subsumed

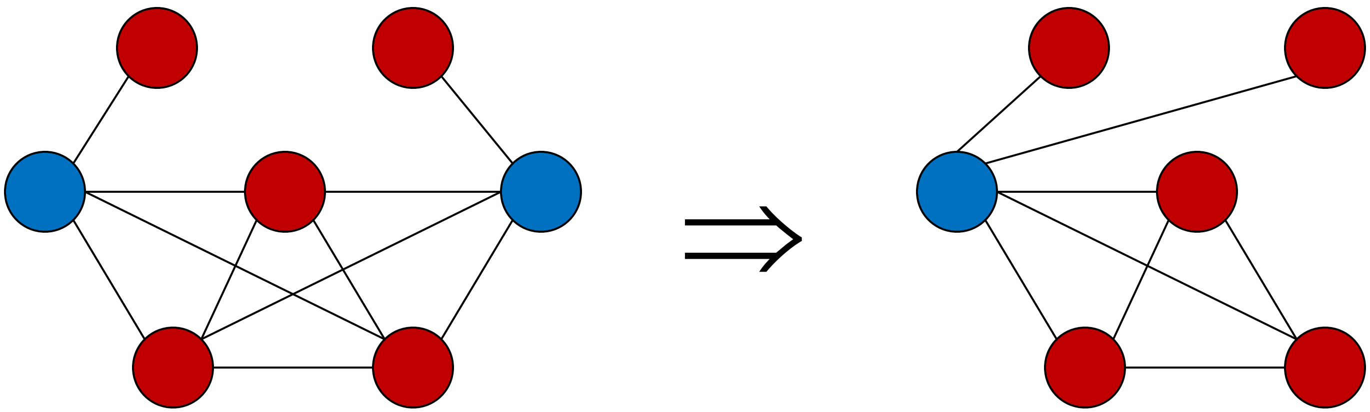


Before After

Examples shown for 4-coloring

Reduction operators

Connected to (k-1)-clique



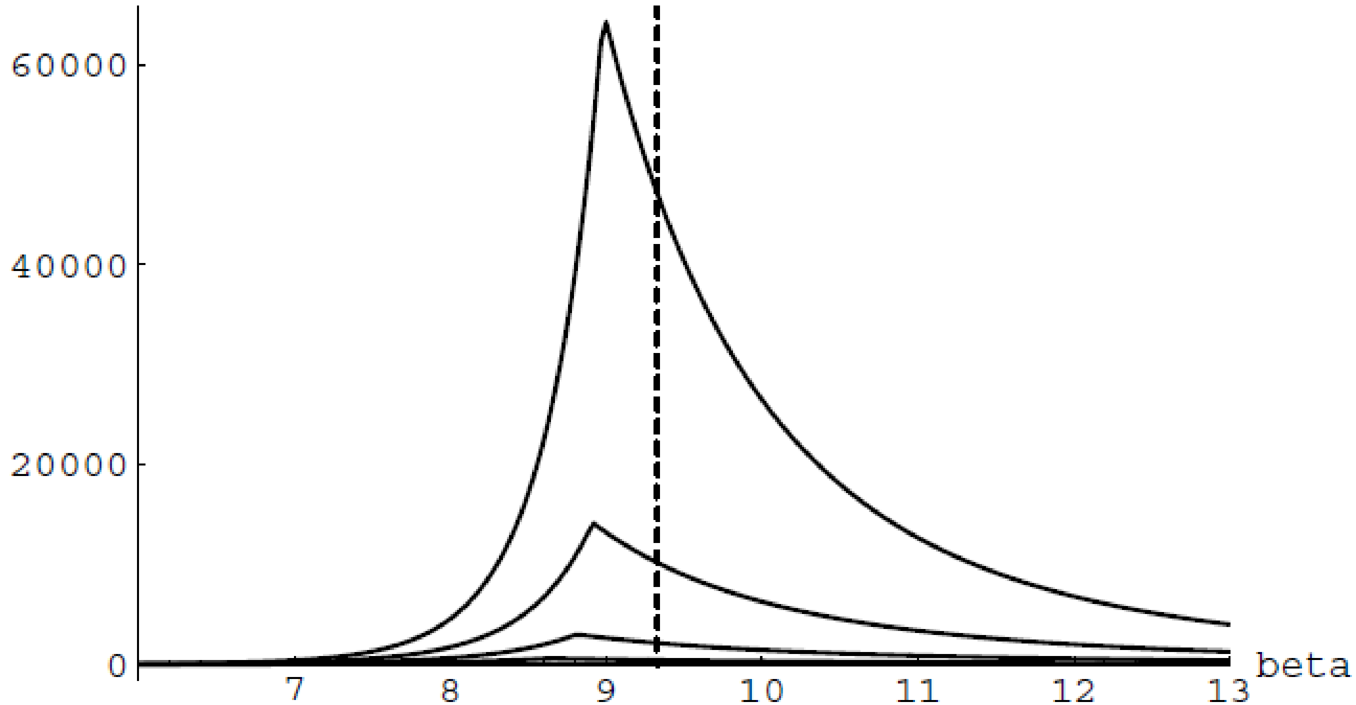
Before $\qquad$ After

Examples shown for 4-coloring

GENERAL FRAMEWORK

- Nogoods = illegal tuples of values for variables
- Sperner system = family of sets s.t. no set is contained in another set
- Construct Sperner system of nogoods by considering only minimized (inclusion-minimal) nogoods
- Order parameter:
  β = #minimized nogoods / #variables
- Q: How many minimized nogoods in k-graph coloring?
- A: #minimized nogoods = $|E|\cdot k$
- #minimized nogoods / #variables $\propto$ avg. degree

Theoretical prediction



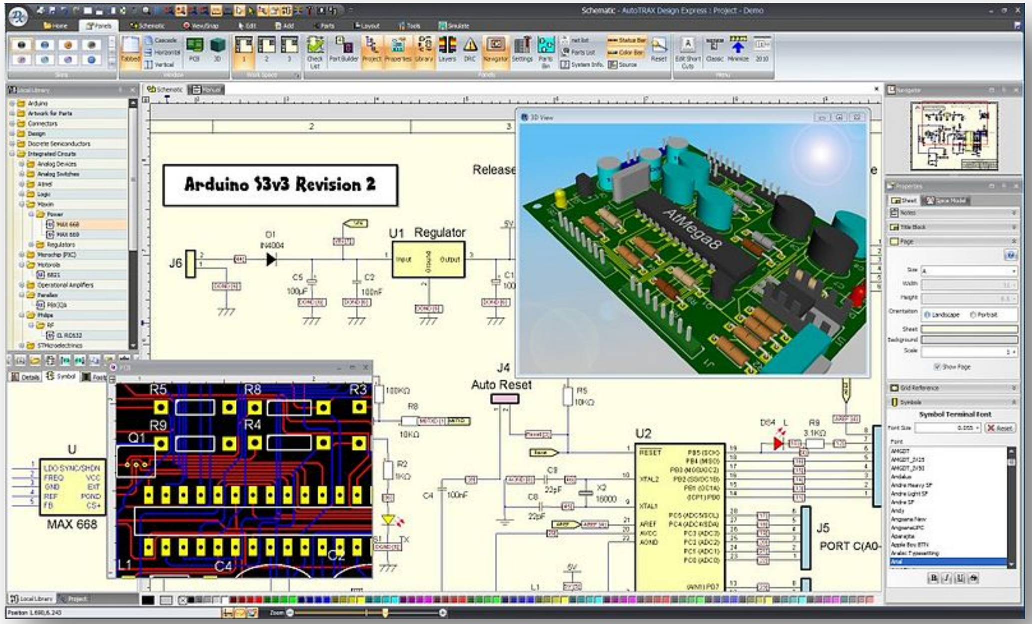
Williams and Hogg, AIJ 1994

CSP example: SAT

- Given a formula in propositional logic, find a satisfying assignment (or prove that none exists)
- Example: $(a \vee b) \wedge \overbrace{(\neg a \vee \overset{\text{literal}}{\neg b} \vee c)}^{\text{clause}}$
- Conjunctive normal form = conjunction of disjunctive clauses
- First established NP-complete problem
  - S. A. Cook. The complexity of theorem proving procedures. STOC 1971

SAT applications

- Electronic design automation, e.g., testing and verification
- AI: automated theorem proving, knowledge base deduction
- Software (from Athanasios): checking if program crashes

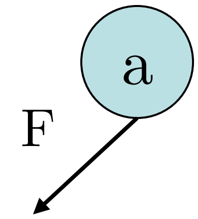


DPLL algorithm (1962)

$\neg a \vee b \vee c$
$a \vee c \vee d$
$a \vee c \vee \neg d$
$a \vee \neg c \vee d$
$a \vee \neg c \vee \neg d$
$\neg b \vee \neg c \vee d$
$\neg a \vee b \vee \neg c$
$\neg a \vee \neg b \vee c$

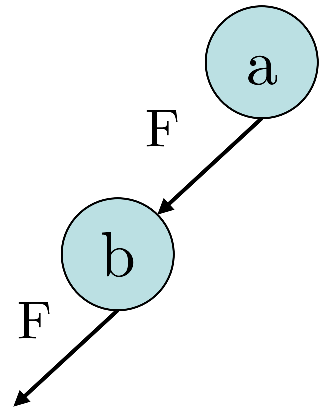
DPLL algorithm (1962)

$\neg a \lor b \lor c$
$c \lor d$
$c \lor \neg d$
$\neg c \lor d$
$\neg c \lor \neg d$
$\neg b \lor \neg c \lor d$
$\neg a \lor b \lor \neg c$
$\neg a \lor \neg b \lor c$



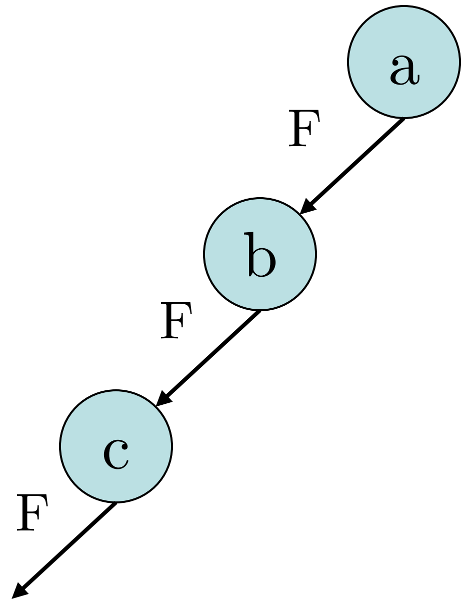
DPLL algorithm (1962)

$\neg a \vee c$
$c \vee d$
$c \vee \neg d$
$\neg c \vee d$
$\neg c \vee \neg d$
$\neg b \vee \neg c \vee d$
$\neg a \vee \neg c$
$\neg a \vee \neg b \vee c$



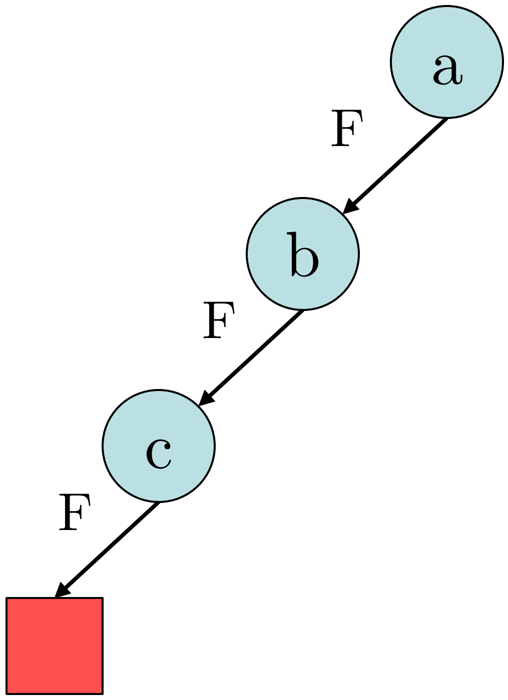
DPLL algorithm (1962)

$\neg a$
d
$\neg d$
$\neg c \vee d$
$\neg c \vee \neg d$
$\neg b \vee \neg c \vee d$
$\neg a \vee \neg c$
$\neg a \vee \neg b$



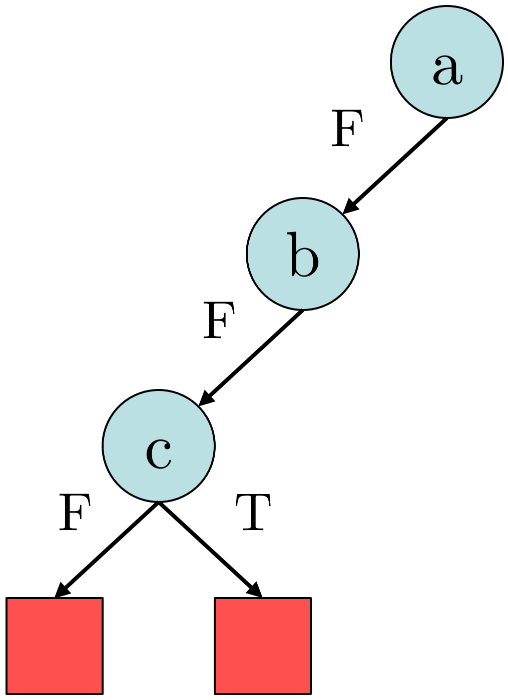
DPLL algorithm (1962)

$\neg a$
d
$\neg d$
$\neg c \lor d$
$\neg c \lor \neg d$
$\neg b \lor \neg c \lor d$
$\neg a \lor \neg c$
$\neg a \lor \neg b$



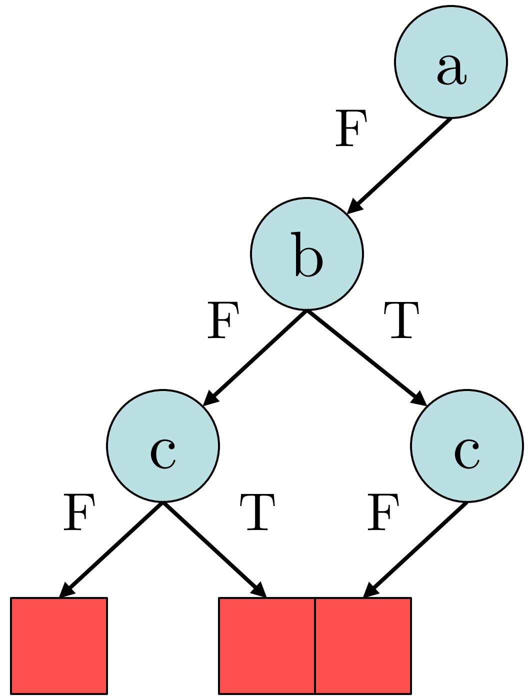
DPLL algorithm (1962)

$\neg a \vee c$
$c \vee d$
$c \vee \neg d$
d
$\neg d$
$\neg b \vee d$
$\neg a$
$\neg a \vee \neg b \vee c$



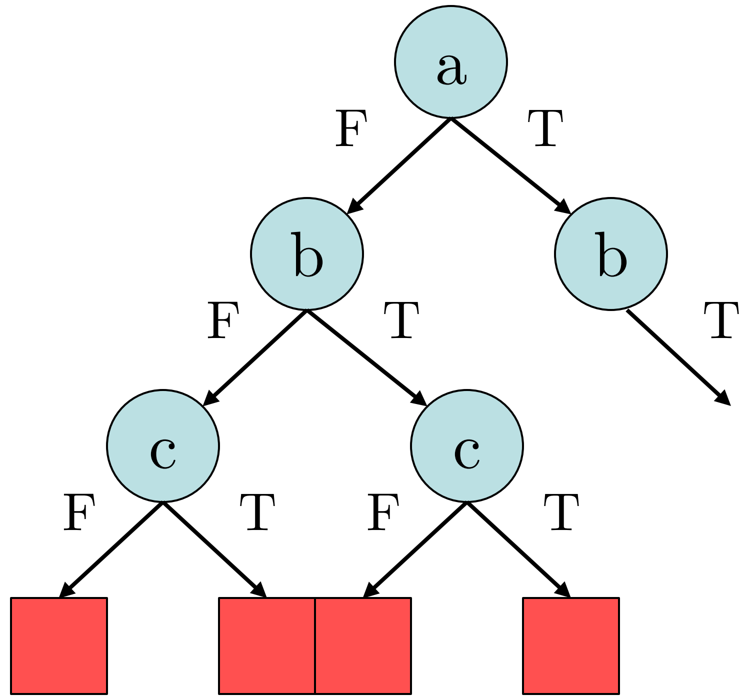
DPLL algorithm (1962)

$\neg a \vee b \vee c$
d
$\neg d$
$\neg c \vee d$
$\neg c \vee \neg d$
$\neg c \vee d$
$b \vee \neg c$
$\neg a \vee c$



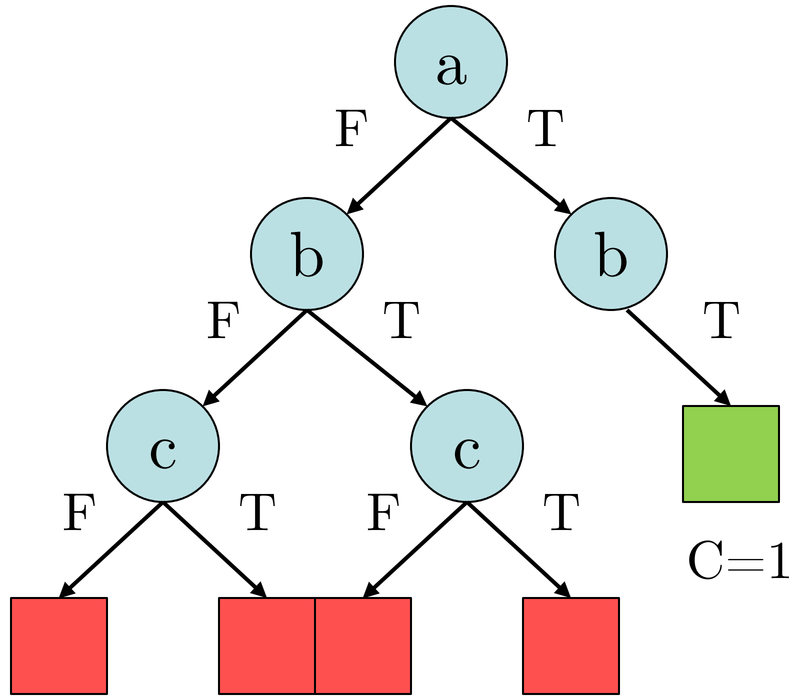
DPLL algorithm (1962)

$b \lor c$
$a \lor c \lor d$
$a \lor c \lor \neg d$
$a \lor \neg c \lor d$
$a \lor \neg c \lor \neg d$
$\neg c \lor d$
$b \lor \neg c$
c



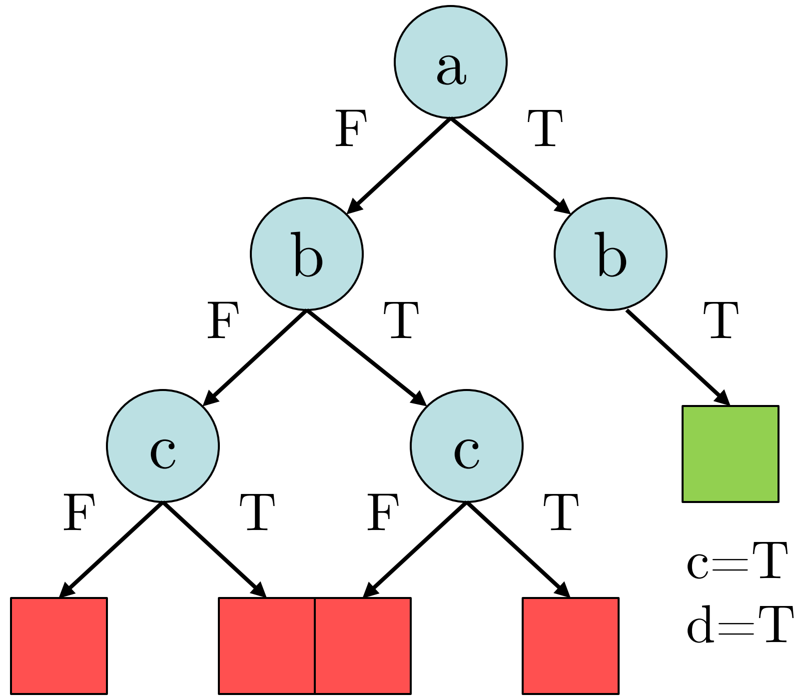
DPLL algorithm (1962)

$b \lor c$
$a \lor c \lor d$
$a \lor c \lor \neg d$
$a \lor \neg c \lor d$
$a \lor \neg c \lor \neg d$
$\neg c \lor d$
$b \lor \neg c$
c



DPLL ALGORITHM (1962)

$b \vee c$
$a \vee c \vee d$
$a \vee c \vee \neg d$
$a \vee \neg c \vee d$
$a \vee \neg c \vee \neg d$
d
$b \vee \neg c$
c



DPLL algorithm (1962)

- Assign next value
- Erase unsatisfied literals, backtrack when clause becomes empty
- **Unit propagation** = if clause has only one variable left, assign satisfying value
- **Boolean constraint propagation** = iteratively apply unit propagation until there are no unit clauses

Variable ordering for DPLL

- Three design principles for heuristics
- Constrainedness
  - Choose variables that are more constrained
  - Motivation: attack most difficult part of the problem first
  - Short clauses are most constraining: only take them into account
  - Several variants, e.g., most occurrences in short clauses

Variable ordering for DPLL

- Satisfaction
  - Try to find variables that come closest to satisfying the problem
  - Clause of length k rules out 2^{-k} of possible assignments; give weight 2^{-k} to each clause of length k
  - For each literal, calculate weighted sum of clauses that it appears in
  - Gives variable and value ordering

Variable ordering for DPLL

- Simplification
  - Want to simplify the problem as much as possible
  - For each assignment we get a cascade of unit propagations
  - Test all assignments and choose the one that caused the largest cascade
  - Successful variants only probe promising variables (based on other heuristics)

DPLL and Horn clauses

- $[(a \wedge b \wedge c) \Rightarrow d]$ is equivalent to $[\neg(a \wedge b \wedge c) \vee d]$ is equivalent to $[\neg a \vee \neg b \vee \neg c \vee d]$ which is a Horn clause
- Formal def: **Horn clause** = clause that has at most one non-negated variable

DPLL and Horn clauses

- **Theorem.** If BCP applied to a set of Horn clauses does not result in contradiction then the set is satisfiable
- **Proof**
  - Assume BCP finished
  - Remove satisfied clauses and assigned variables from unsatisfied clauses
  - Remaining clauses have at least two literals, therefore at least one negated variable
  - How do we satisfy the remaining clauses?
  - Satisfy remaining clauses by assigning false to all unassigned variables ■

DPLL and Horn clauses

- **Corollary.** Given only Horn clauses, DPLL runs in polynomial time
- Reason: we never take a wrong path in the tree because BCP immediately finds a conflict

Converting CSP to SAT

- SAT is obviously a CSP
- A CSP can also easily be encoded as SAT
  - Clearly a polytime encoding *exists* because SAT is NP-c
- For each variable X and every j∈Dom(X) we have a SAT variable $Z_{X=d}$
- For example, if Dom(X)={1,2,3,4} then we have $Z_{X=1}$, $Z_{X=2}$, $Z_{X=3}$, $Z_{X=4}$

Converting CSP to SAT

- "At least one value" clause:
  $Z_{X=1} \vee Z_{X=2} \vee Z_{X=3} \vee Z_{X=4}$
- At most one value" clauses:
  $(\neg Z_{X=1} \vee \neg Z_{X=2}) \wedge (\neg Z_{X=1} \vee \neg Z_{X=3}) \wedge$
  $(\neg Z_{X=1} \vee \neg Z_{X=4}) \wedge (\neg Z_{X=2} \vee \neg Z_{X=3}) \wedge$
  $(\neg Z_{X=2} \vee \neg Z_{X=4}) \wedge (\neg Z_{X=3} \vee \neg Z_{X=4})$

Converting CSP to SAT

- For every constraint and every tuple that falsifies the constraint, add clause
- For example if constraint is falsified by (X=1, Y=3) add constraint $(\neg Z_{X=1} \vee \neg Z_{Y=3})$

Linear encoding

- Impose an order on the domain of each variable
- Let X with Dom(X)={1,...,d}
- Add d-1 SAT variables $Z_{X\leq i}$ for all $i \in \{1,...,d-1\}$
- Add clauses $[\neg Z_{X\leq i} \vee Z_{X\leq i+1}]$ for all i
- Assign X=i by $Z_{X\leq i} = T$, $Z_{X\leq i-1}=F$
- Advantage: BCP automatically assigns $Z_{X\leq k} = T$ for every $k > i$, $Z_{X\leq k} = F$ for every $k<i-1$