

1

Constraint Satisfaction Problems

Chapter 5

Review: Last Chapter

2

Best-first search

Heuristic functions estimate costs of shortest paths

Good heuristics can dramatically reduce search cost

Greedy best-first search expands lowest h

— incomplete and not always optimal

A* search expands lowest $g + h$

— complete and optimal

— also optimally efficient (up to tie-breaks, for forward search)

Admissible heuristics can be derived from exact solution of relaxed problems

Review: Last Chapter

3

Local search algorithms

- ▣ the **path** to the goal is irrelevant; the goal state itself is the solution
- ▣ keep a single "current" state, try to improve it

Hill-climbing search

- depending on initial state, can get stuck in local maxima

Simulated annealing search

- escape local maxima by allowing some "bad" moves but **gradually decrease** their frequency

Local beam search

- Keep track of k states rather than just one

Genetic algorithms

Outline

4

- CSP examples
- Backtracking search for CSPs
- Problem structure and problem decomposition
- Local search for CSPs

Constraint satisfaction problems (CSPs)

5

Standard search problem:

state is a “black box” – any old data structure that supports goal test, eval, successor

任何可以由目标测试、评价函数、后继函数访问的数据结构

CSP:

state is defined by **variables** X_i with **values** from **domain** (值域) D_i

goal test is a set of **constraints** specifying allowable combinations of values for subsets of variables

每个约束包括一些变量的子集，并指定这些子集的值之间允许进行的合并

Simple example of a **formal representation language** (形式化表示方法)

Allows useful **general-purpose** (通用的，而不是问题特定的) algorithms with more power than standard search algorithms

Example: Map-Coloring

6



Variables WA, NT, Q, NSW, V, SA, T

Domains $D_i = \{red, green, blue\}$

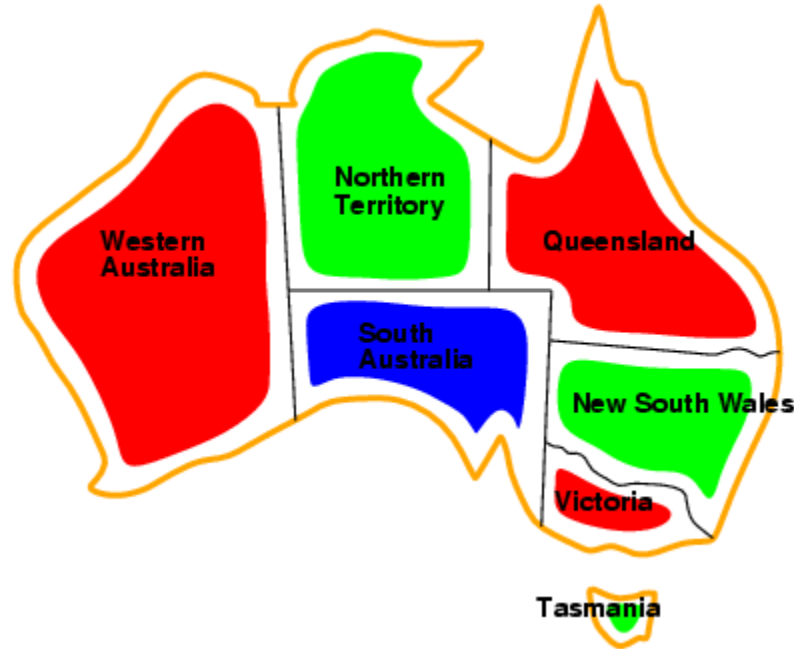
Constraints: adjacent regions must have different colors

e.g., $WA \neq NT$, or (if the language allows this), or

$(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), \dots\}$

Example: Map-Coloring

7



Solutions are assignments satisfying all constraints, e.g.,

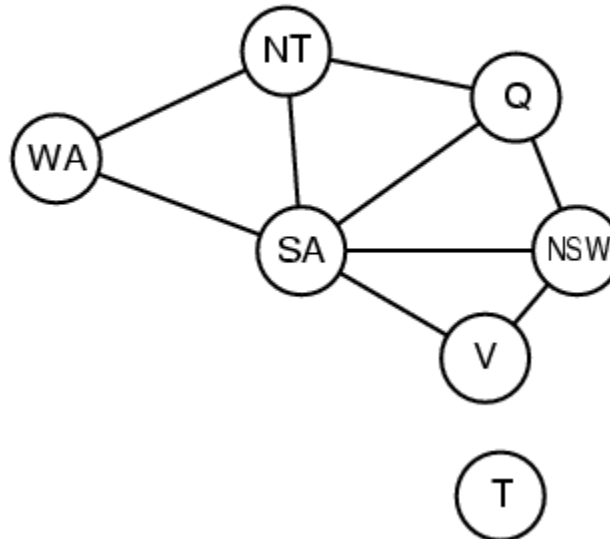
$\{WA=red, NT=green, Q=red, NSW=green, V=red, SA=blue, T=green\}$

Constraint graph (约束图)

8

Binary CSP: each constraint relates two variables

Constraint graph: nodes are variables, arcs are constraints



General-purpose CSP algorithms use the graph structure to speed up search.

E.g., Tasmania is an independent subproblem!

Varieties of CSPs

9

- Discrete variables
 - ▣ finite domains 有限区域:
 - n variables, domain size $d \rightarrow O(d^n)$ complete assignments
 - e.g., Boolean CSPs, incl. Boolean satisfiability (NP-complete)
 - ▣ infinite domains 无限值域 (integers, strings, etc.)
 - e.g., job scheduling, variables are start/end days for each job
 - need a **constraint language** (约束语言), e.g.,
 - **linear** constraints solvable, **nonlinear** undecidable
- Continuous variables
 - ▣ e.g., start/end times for Hubble Space Telescope observations
 - ▣ linear constraints solvable in polynomial time by linear programming (LP) methods

Varieties of constraints

10

Unary (一元) constraints involve a single variable,

e.g., $SA \neq \text{green}$

Binary (二元) constraints involve pairs of variables,

e.g., $SA \neq WA$

Higher-order constraints involve 3 or more variables,

e.g., cryptarithmic (密码算数) column constraints

Preferences (soft constraints), e.g., red is better than green

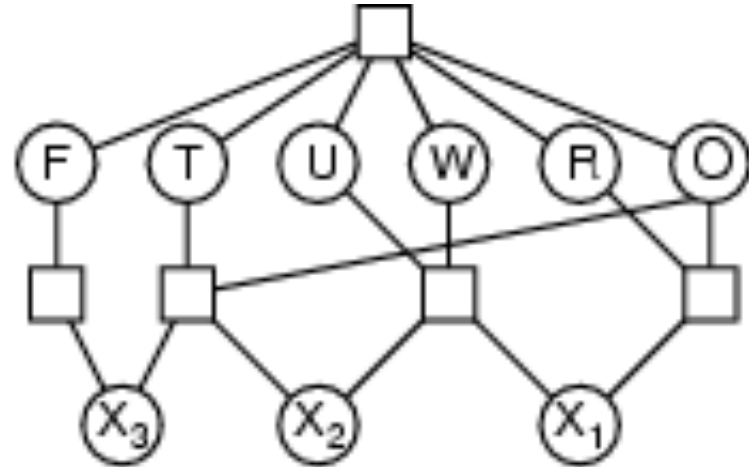
often representable by a cost for each variable assignment (个体变量赋值的耗散)

→ constrained optimization problems

Example: Cryptarithmic

11

$$\begin{array}{r} \text{T W O} \\ + \text{T W O} \\ \hline \text{F O U R} \end{array}$$



Variables: $F, T, U, W, R, O, X_1, X_2, X_3$

Domains: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Constraints:

$\text{alldiff}(F, T, U, W, R, O)$

$$O + O = R + 10 \cdot X_1$$

$$X_1 + W + W = U + 10 \cdot X_2$$

$$X_2 + T + T = O + 10 \cdot X_3$$

$$X_3 = F, T \neq 0, F \neq 0$$

Real-world CSPs

12

Assignment problems

e.g., who teaches what class
who reviews which papers

Timetabling problems

e.g., which class is offered when and where?

Hardware configuration

Transportation scheduling

Factory scheduling

Floorplanning (平面布置)

Notice that many real-world problems involve real-valued variables

Enumerate assignments

13

Dumb

Exponential time d^n

But complete

can we be clever about exponential time algorithms?

Standard search formulation (incremental增量形式化)

14

Let's start with the straightforward approach, then fix it

States are defined by the values assigned so far

- ▣ **Initial state:** the empty assignment, $\{\}$
- ▣ **Successor function:** assign a value to an unassigned variable that does not conflict with current assignment
→ fail if no legal assignments
- ▣ **Goal test:** the current assignment is complete

1. This is the same for all CSPs! 😊
2. Every solution appears at depth n with n variables
→ use depth-first search
3. Path is irrelevant, so can also use complete-state formulation (完全状态形式化)
4. $b = (n - l)d$ at depth l , hence $n! \cdot d^n$ leaves !!!! 😞

Backtracking search

15

Variable assignments are **commutative** (可交换性), i.e.,
[**WA = red** then **NT = green**] same as [**NT = green** then **WA = red**]

Only need to consider assignments to a single variable at each node
b = d and there are **dⁿ** leaves

Depth-first search for CSPs with single-variable assignments is called
backtracking search

Backtracking search is the basic uninformed algorithm for CSPs

Can solve **n**-queens for **n ≈ 25**

Backtracking search

16

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
  return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment given CONSTRAINTS[csp] then
      add {var = value} to assignment
      result ← RECURSIVE-BACKTRACKING(assignment, csp)
      if result ≠ failure then return result
      remove {var = value} from assignment
  return failure
```

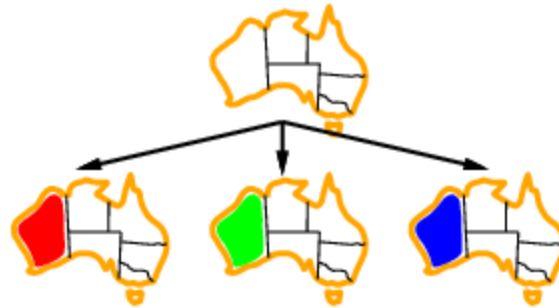

Backtracking example

17



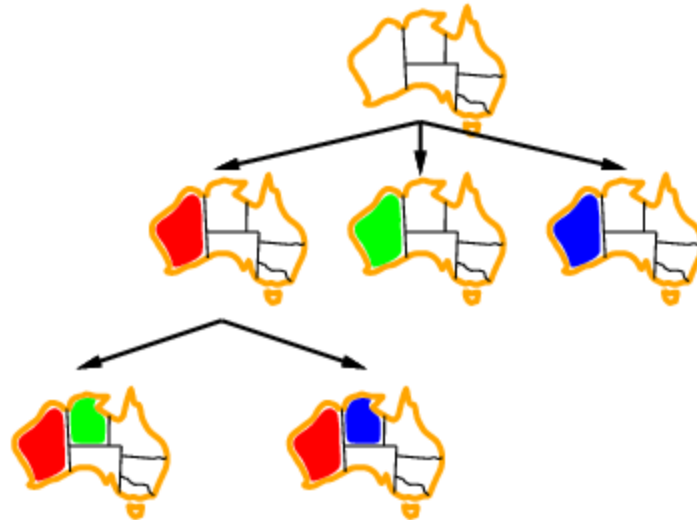
Backtracking example

18



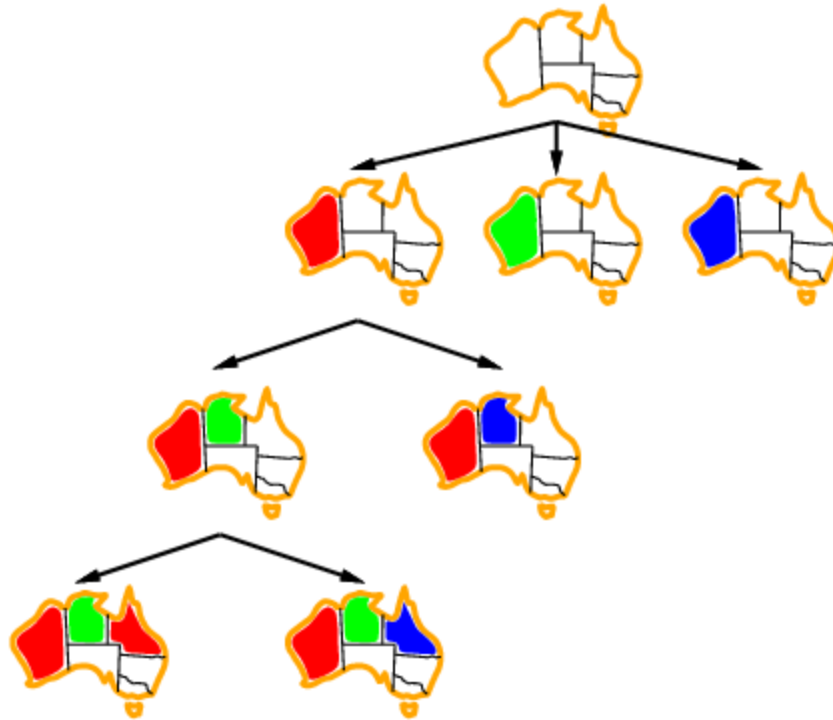
Backtracking example

19



Backtracking example

20



Another backtracking example

21

Is $\{\neg a \vee b, \neg b \vee c, \neg c, \neg a\}$ satisfiable?

Enumerate	a	b	c	
	1	1	1	×
	1	1	0	×
	1	0	1	×
	1	0	0	×
	0	1	1	×
	0	1	0	×
	0	0	1	×
	0	0	0	✓

Backtrack	a	b	c
	1	-	-
	0	1	1
	0	1	0
	0	0	1
	0	0	0

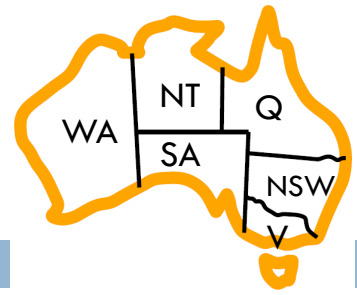
Improving backtracking efficiency

22

General-purpose methods can give huge gains in speed:

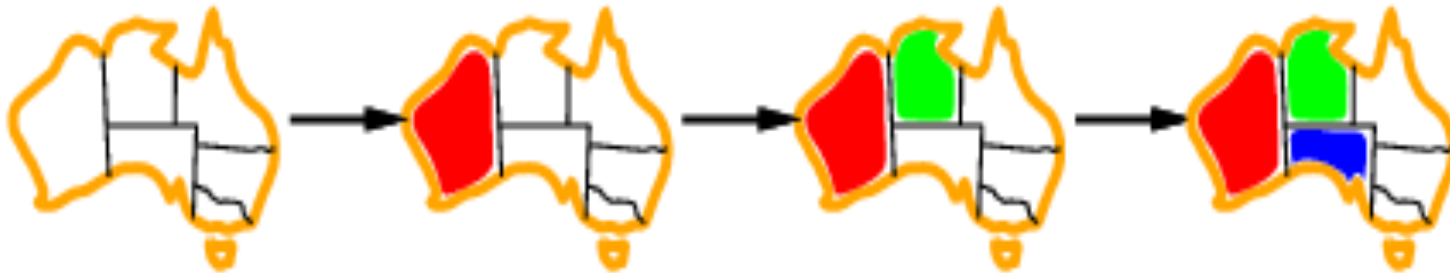
1. Which variable should be assigned next?
2. In what order should its values be tried?
3. Can we detect inevitable (不可避免的) failure early?
4. Can we take advantage of problem structure?

Minimum remaining values



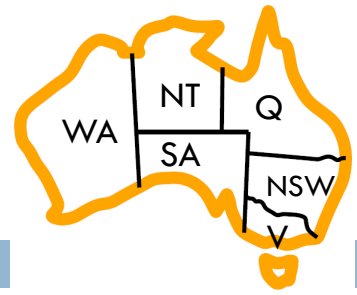
23

Minimum remaining values 最少剩余值(MRV):
choose the variable with the fewest legal values



- Why min rather than max?
- Called **most constrained variable**
- “Fail-fast” ordering

Degree heuristic (度启发式)

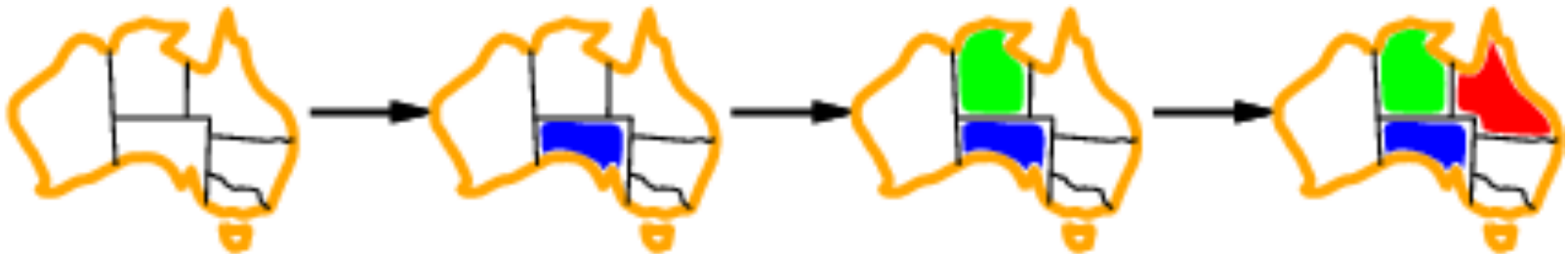


24

Tie-breaker among MRV variables

Degree heuristic:

choose the variable with the most constraints on remaining variables

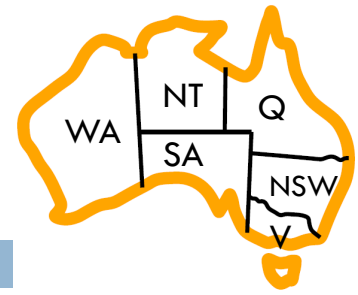


Improving backtracking efficiency

25

General-purpose methods can give huge gains in speed:

1. Which variable should be assigned next?
2. In what order should its values be tried?
3. Can we detect inevitable failure early?
4. Can we take advantage of problem structure?

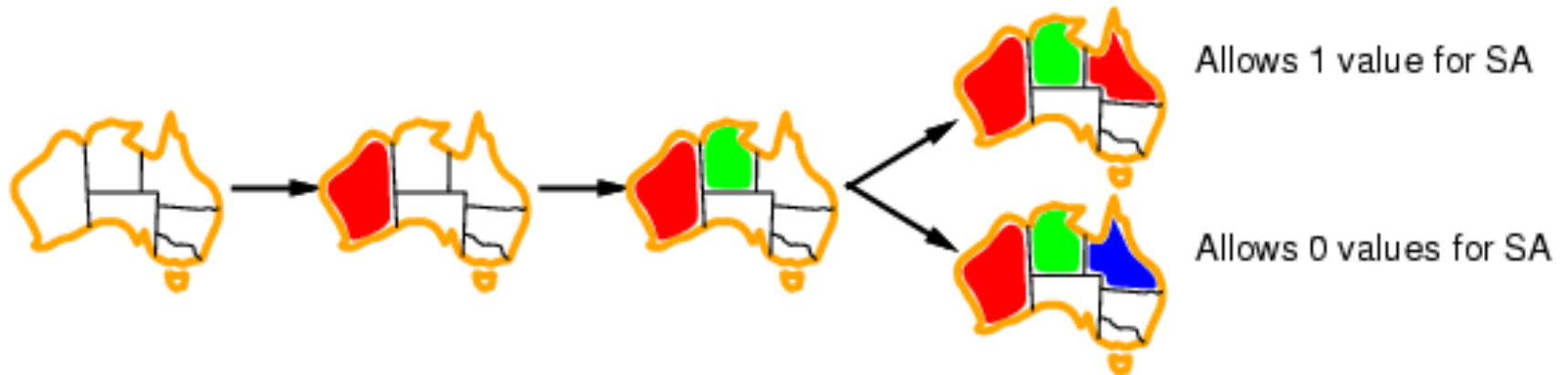


Least constraining value

26

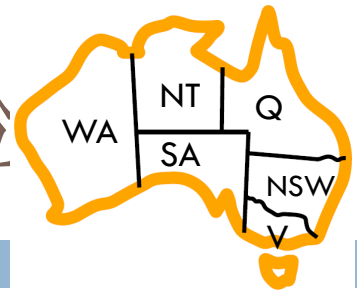
Given a variable, choose the least constraining value (最少约束值) :

- ▣ the one that rules out the fewest values in the remaining variables
- ▣ Note that it may take some computation to determine this!



Combining these heuristics makes 1000 queens feasible

Forward checking—前向检验



27

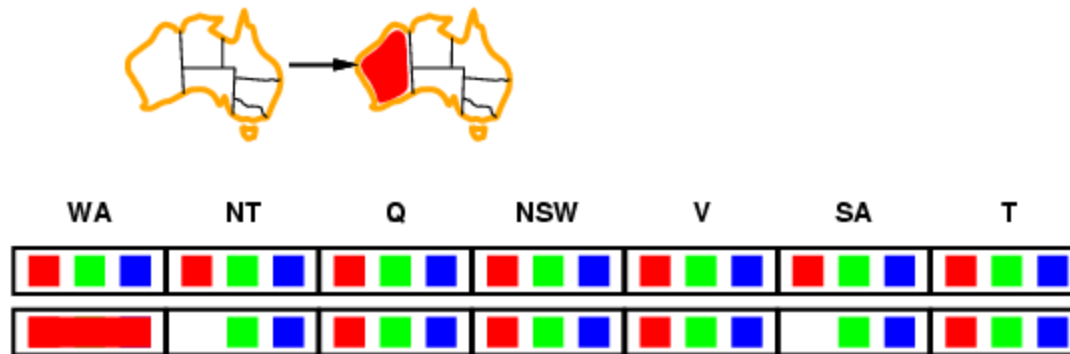
Idea: Keep track of remaining legal values for unassigned variables

Terminate search when any variable has no legal values



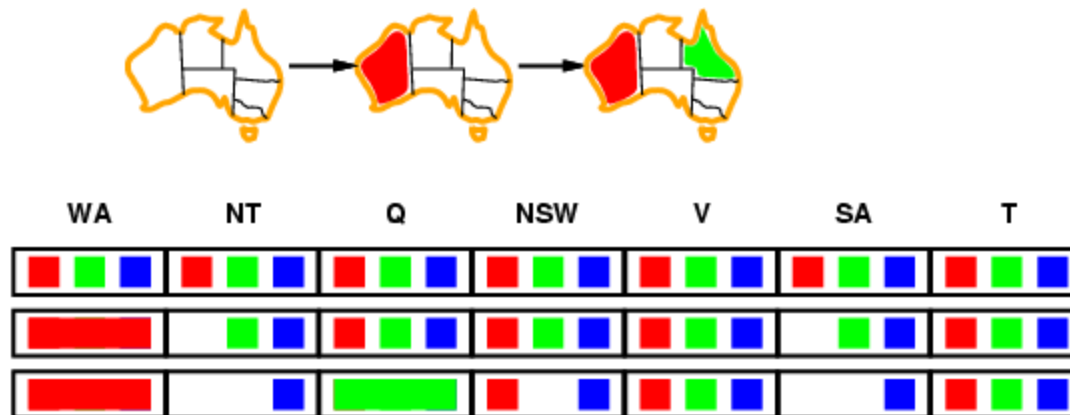
A hand-drawn map of Australia with a thick orange border. The map is divided into regions labeled with abbreviations: WA (Western Australia) on the left, NT (Northern Territory) in the top center, Q (Queensland) in the top right, SA (South Australia) in the center, NSW (New South Wales) in the bottom right, and V (Victoria) in the bottom right corner.

Idea: Keep track of remaining legal values for unassigned variables
 Terminate search when any variable has no legal values

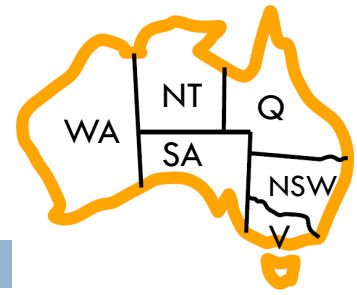


A hand-drawn map of Australia with a thick orange border. The map is divided into regions labeled with abbreviations: WA (Western Australia) on the left, NT (Northern Territory) in the top center, SA (South Australia) in the center, Q (Queensland) in the top right, NSW (New South Wales) in the bottom right, and V (Victoria) in the bottom right corner.

Terminate search when any variable has no legal values



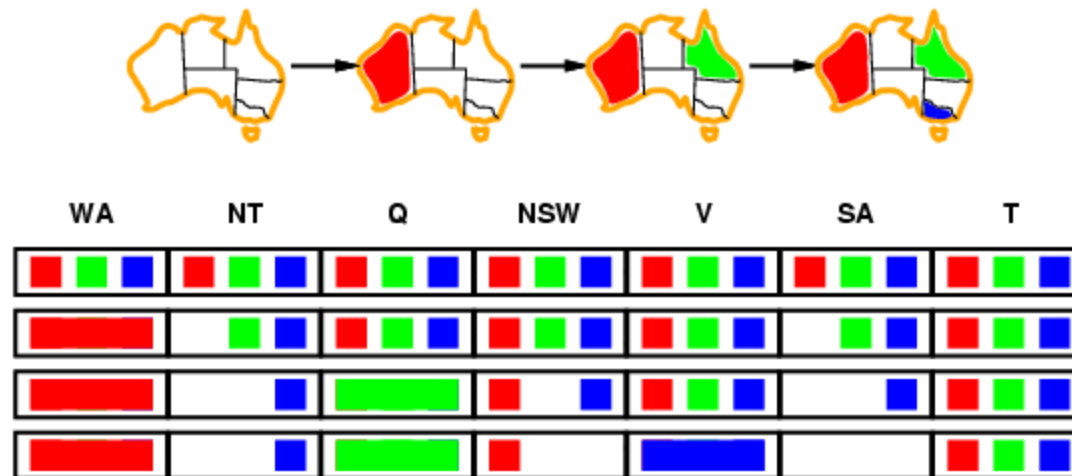
Forward checking



30

Idea: Keep track of remaining legal values for unassigned variables

Terminate search when any variable has no legal values



Improving backtracking efficiency

31

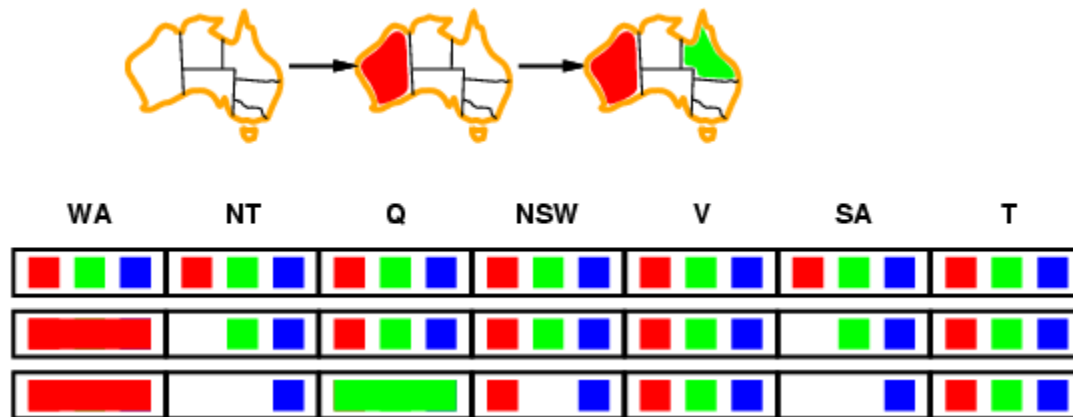
General-purpose methods can give huge gains in speed:

1. Which variable should be assigned next?
2. In what order should its values be tried?
3. **Can we detect inevitable failure early?**
4. Can we take advantage of problem structure?

Constraint propagation — 约束传播

32

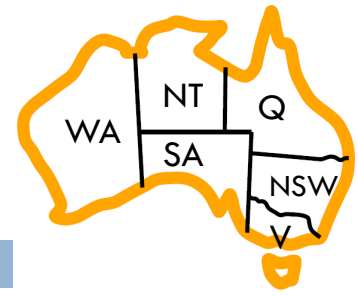
Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:



NT and *SA* cannot both be blue!

Constraint propagation repeatedly enforces constraints locally

Arc consistency — 弧相容

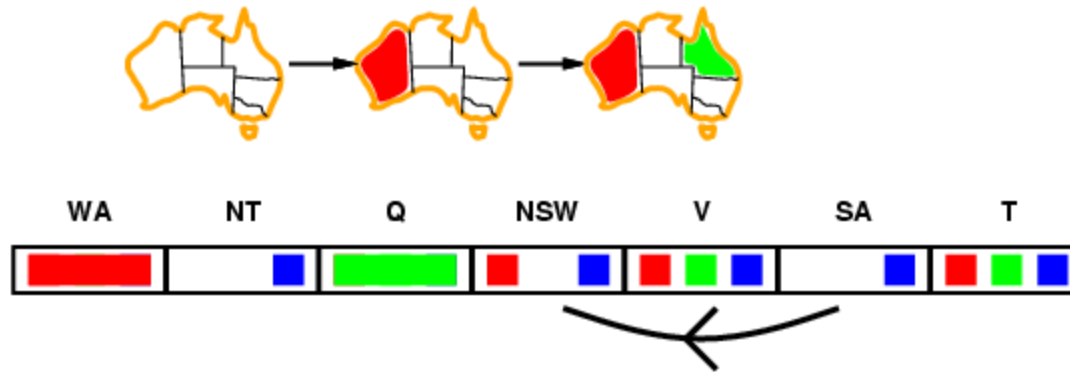


33

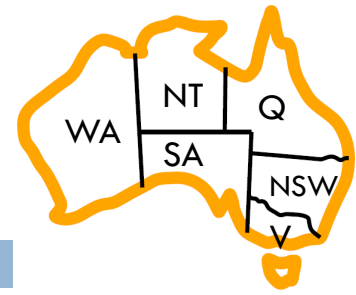
Simplest form of propagation makes each arc **consistent**

$X \rightarrow Y$ is consistent iff

for **every** value x of X there is **some** allowed y



Arc consistency

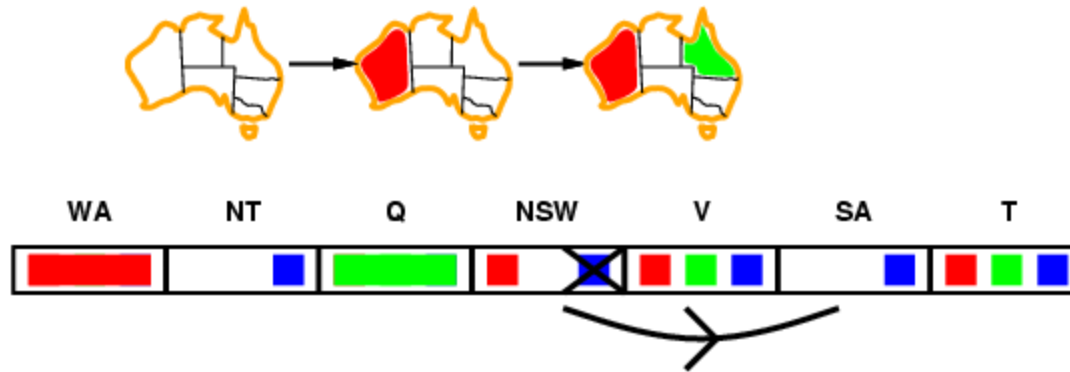


34

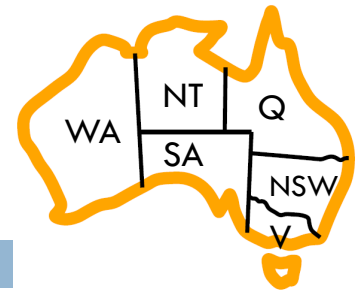
Simplest form of propagation makes each arc **consistent**

$X \rightarrow Y$ is consistent iff

for **every** value x of X there is **some** allowed y



Arc consistency

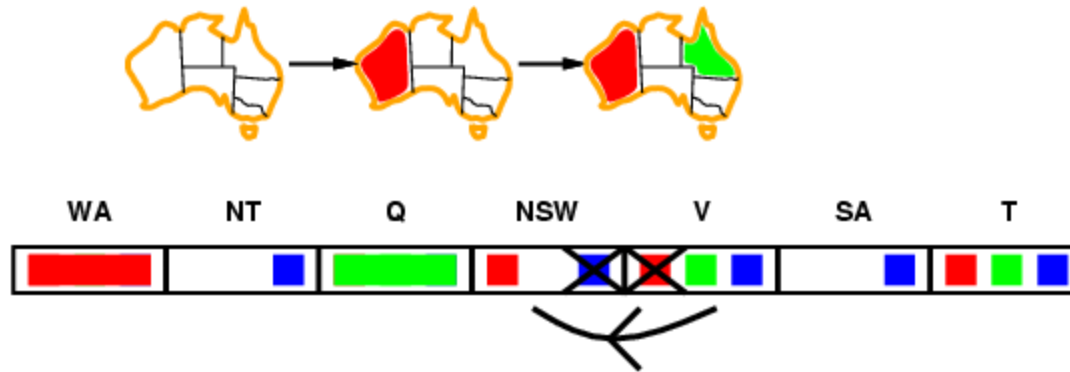


35

Simplest form of propagation makes each arc **consistent**

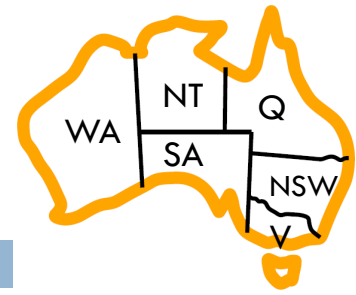
$X \rightarrow Y$ is consistent iff

for **every** value x of X there is **some** allowed y



If X loses a value, neighbors of X need to be rechecked

Arc consistency

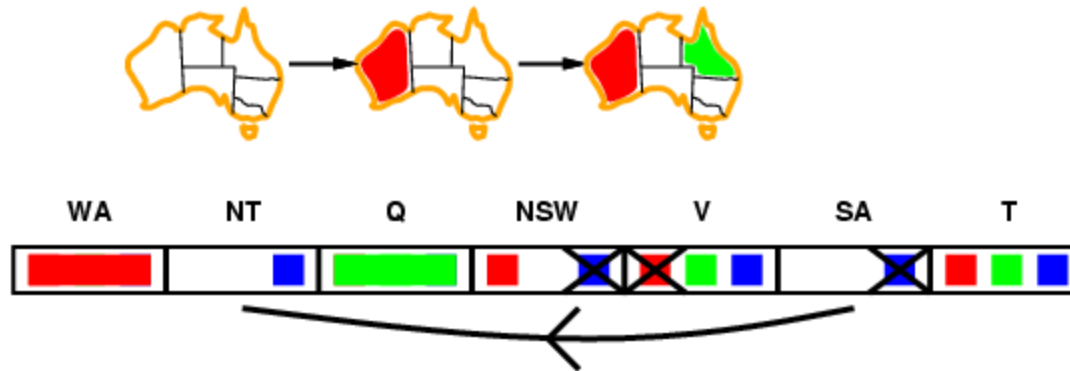


36

Simplest form of propagation makes each arc **consistent**

$X \rightarrow Y$ is consistent iff

for **every** value x of X there is **some** allowed y



If X loses a value, neighbors of X need to be rechecked

Arc consistency detects failure earlier than forward checking

Can be run as a preprocessor or after each assignment

Arc consistency algorithm AC-3

37

function AC-3(*csp*) **returns** the CSP, possibly with reduced domains

inputs: *csp*, a binary CSP with variables $\{X_1, X_2, \dots, X_n\}$

local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

while *queue* is not empty **do**

$(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\text{queue})$

if REMOVE-INCONSISTENT-VALUES(X_i, X_j) **then**

for each X_k **in** NEIGHBORS[X_i] **do**

 add (X_k, X_i) to *queue*

function REMOVE-INCONSISTENT-VALUES(X_i, X_j) **returns** true iff succeeds

removed \leftarrow false

for each x **in** DOMAIN[X_i] **do**

if no value y in DOMAIN[X_j] allows (x, y) to satisfy the constraint $X_i \leftrightarrow X_j$

then delete x from DOMAIN[X_i]; *removed* \leftarrow true

return *removed*

$O(n^2d^3)$ (but detecting **all** is NP-hard)

Improving backtracking efficiency

38

General-purpose methods can give huge gains in speed:

1. Which variable should be assigned next?
2. In what order should its values be tried?
3. Can we detect inevitable failure early?
4. **Can we take advantage of problem structure?**

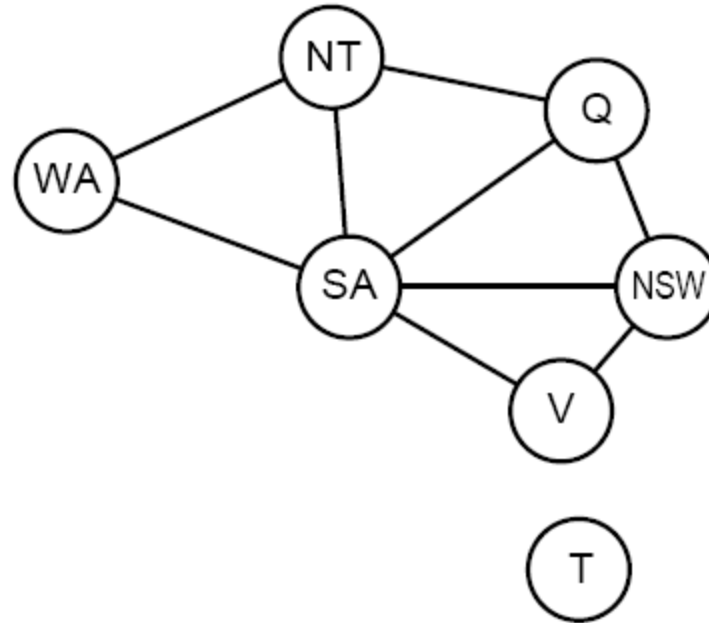
Outline

39

- CSP examples
- Backtracking search for CSPs
- Problem structure and problem decomposition
- Local search for CSPs

Problem structure

40



Tasmania and mainland are **independent subproblems**

Identifiable as **connected components** (连通域) of constraint graph

Problem structure contd.

41

Suppose each subproblem has c variables out of n total

Worst-case solution cost is $n/c \cdot d^c$, linear in n

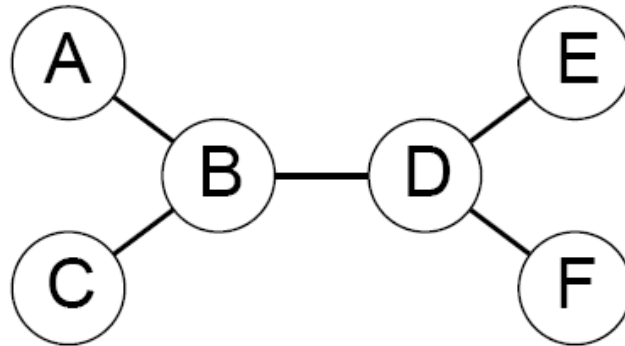
E.g., $n=80$, $d=2$, $c=20$

$2^{80} = 4$ billion years at 10 million nodes/sec

$4 \cdot 2^{20} = 0.4$ seconds at 10 million nodes/sec

Tree-structured CSPs

42



Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n \cdot d^2)$ time
任何一个树状结构的CSP问题可以在变量个数的线性时间内求解

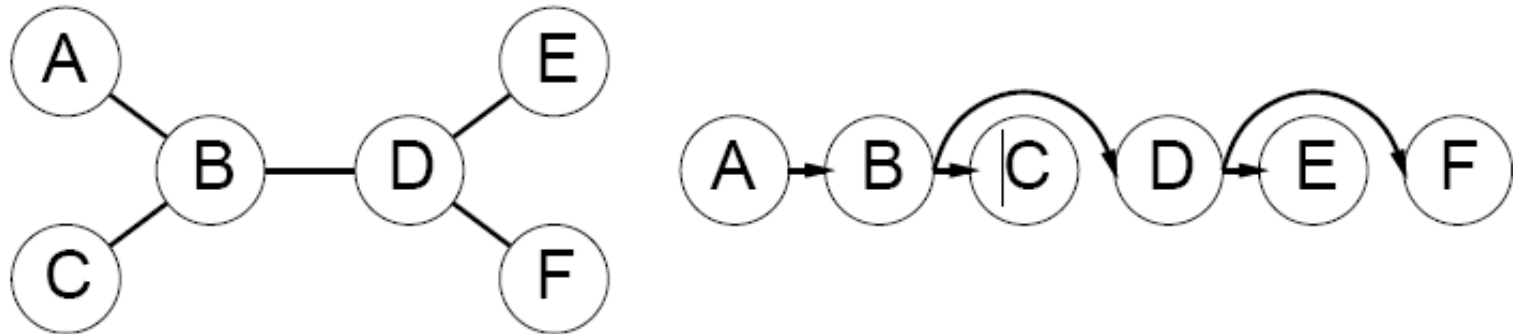
Compare to general CSPs, where worst-case time is $O(d^n)$

This property also applies to logical and probabilistic reasoning:
an important example of the relation between syntactic restrictions (语法约束)
and the complexity of reasoning.

Algorithm for tree-structured CSPs

43

1. Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering



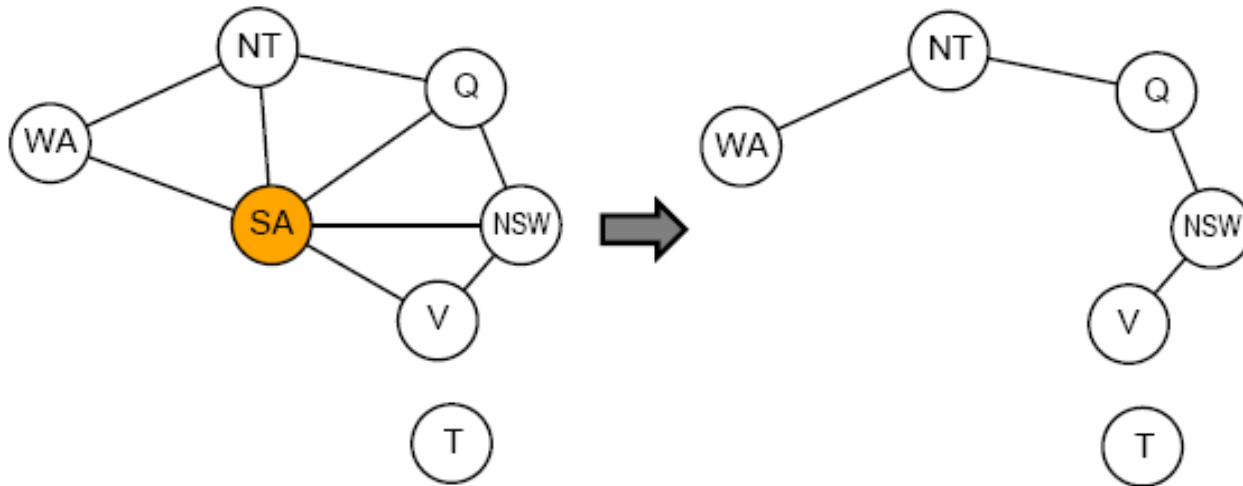
2. For j from n down to 2, apply REMOVEINCONSISTENT($Parent(X_j), X_j$)
3. For j from 1 to n , assign X_j consistently with $Parent(X_j)$

Complexity: $O(n \cdot d^2)$

Nearly tree-structured CSPs

44

Conditioning: instantiate a variable, prune its neighbors' domains



Cutset conditioning (割集调整) : instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree

Cutset size $c \rightarrow$ runtime $O(d^c (n - c)d^2)$, very fast for small c

Finding a smallest cutset is an NP problem, efficient approximate algorithms exist

Outline

45

- CSP examples
- Backtracking search for CSPs
- Problem structure and problem decomposition
- Local search for CSPs

Iterative algorithms for CSPs

46

Hill-climbing, simulated annealing typically work with
“complete” states (完全状态的形式化) , i.e., all variables assigned

To apply to CSPs:

- allow states with unsatisfied constraints

- operators **reassign** variable values

Variable selection: randomly select any conflicted variable

Value selection by **min-conflicts** (最小冲突) heuristic:

- choose value that violates the fewest constraints

 - 选择会造成与其它变量的冲突最小的值

- i.e., hillclimb with $h(n)$ = total number of violated constraints

Example: 4-Queens

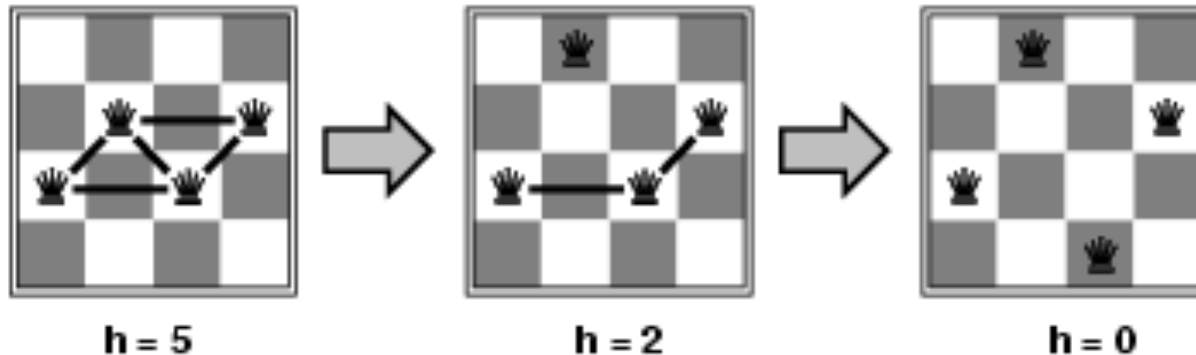
47

States: 4 queens in 4 columns ($4^4 = 256$ states)

Actions: move queen in column

Goal test: no attacks

Evaluation: $h(n)$ = number of attacks



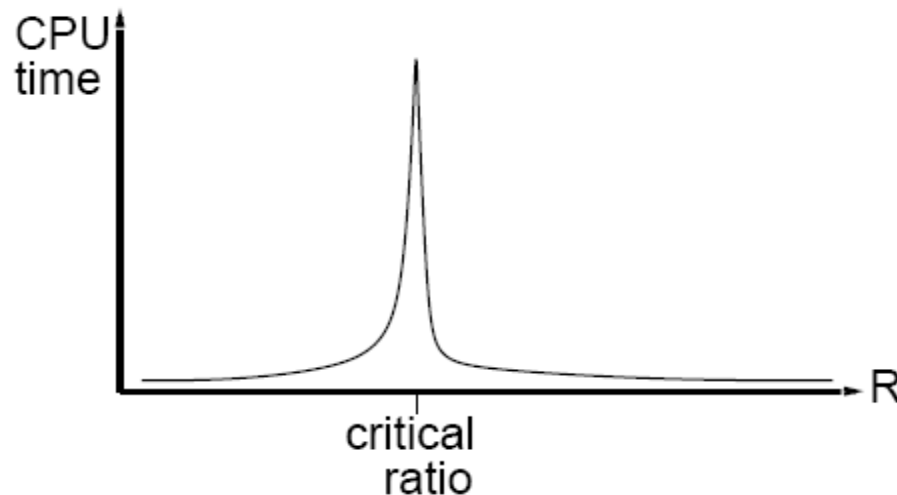
Performance of min-conflicts

48

Given random initial state, can solve n -queens in almost constant time for arbitrary n with high probability (e.g., $n = 10,000,000$)

The same appears to be true for any randomly-generated CSP **except** in a narrow range of the ratio

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$



Example: 3-SAT problems

49

- Each constraint involves 3 variables

# vars	Backtrack+tricks	Min-conflicts
50	1.5s	0.5s
100	3m	10s
150	10h	25s
200		2m
250		3m
300		13m
350		20m

Speedup 1: simulated annealing

50

Idea: escape local maxima by allowing some "bad" moves

but **gradually decrease** their frequency

If 新状态比现有状态好，移动到新状态

Else 以某个小于1的概率接受该移动

▣ 此概率随温度“T”降低而下降

Speedup2: minmax optimization

51

Put weights on constraints

repeat

Primal search: update assignment to minimize weighted violation,
 until stuck

Dual step: update weights to increase weighted violation,
 until unstuck

until solution found, or bored

Speedup2: minmax optimization

52

# vars	Backtrack+tricks	Min-conflicts	Minmax
50	1.5s	0.5s	0.001s
100	3m	10s	0.01s
150	10h	25s	0.1s
200		2m	0.25s
250		3m	0.4s
300		13m	1s
350		20m	2.5s

Summary

53

CSPs are a special kind of problem:

states defined by values of a fixed set of variables

goal test defined by **constraints** on variable values

Backtracking = depth-first search with one variable assigned per node

- Variable ordering and value selection heuristics help significantly
- Forward checking prevents assignments that guarantee later failure
- Constraint propagation (e.g., arc consistency) does additional work to constrain values and detect inconsistencies

- The CSP representation allows analysis of problem structure
- Tree-structured CSPs can be solved in linear time
- Iterative min-conflicts is usually effective in practice

作业

54

- 5.6 (第二版) \approx 6.5 (第三版)
 - ▣ 分别用回溯算法、前向检验算法、**MRV**和最少约束值启发式算法手工求解图5.2 (6.2)中的密码算数问题。
- 5.8, 5.9 (第二版) = 6.11, 6.12 (第三版)