Constraint Satisfaction Problems

吉建民

USTC
jianmin@ustc.edu.cn

2025 年 3 月 11 日

Used Materials

Disclaimer: 本课件采用了 S. Russell and P. Norvig's Artificial Intelligence –A modern approach slides, 徐林莉老师课件和其他网络课程课件，也采用了 GitHub 中开源代码，以及部分网络博客内容

课程回顾

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

课程回顾

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

Table of Contents

CSP examples

Backtracking search for CSPs

Problem structure and problem decomposition

Local search for CSPs

Constraint satisfaction problems (CSPs)

- 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 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

Constraint satisfaction problems (CSPs)

A constraint satisfaction problem (CSP) consists of three components, $\mathcal{X}$, $\mathcal{D}$, and $\mathcal{C}$:

- $\mathcal{X}$ is a set of variables, $\{X_1, \ldots, X_n\}$
- $\mathcal{D}$ is a set of domains, $\{D_1, \ldots, D_n\}$, one for each variable
  - Each domain D_i consists of a set of allowable values, $\{v_1, \ldots, v_k\}$ for variable X_i.
- $\mathcal{C}$ is a set of constraints that specify allowable combinations of values
  - Each constraint C_i consists of a pair $\langle scope, rel \rangle$, where *scope* is a tuple of variables that participate in the constraint and *rel* is a relation that defines the values that those variables can take on
  - A relation can be represented as an explicit list of all tuples of values that satisfy the constraint, or as an abstract relation that supports two operations: testing if a tuple is a member of the relation and enumerating the members of the relation

Constraint satisfaction problems (CSPs)

To solve a CSP, we need to define a state space and the notion of a solution

- Each state in a CSP is defined by an assignment of values to some or all of the variables, $\{X_i = v_i, X_j = v_j, \ldots\}$
- An assignment that does not violate any constraints is called a consistent or legal assignment
- A complete assignment is one in which every variable is assigned
- A partial assignment is one that assigns values to only some of the variables
- A solution to a CSP is a consistent, complete assignment

Example: Map-Coloring

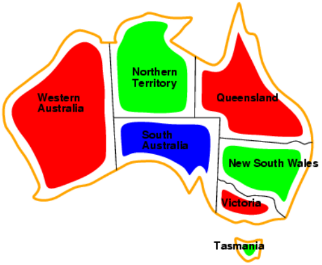
Variables $\mathcal{X} = \{WA, NT, Q, NSW, V, SA, T\}$
Domains $D_i = \{red, green, blue\}$
Constraints: adjacent regions must have different colors

$$\mathcal{C} = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, \\ WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}$$

where $SA \neq WA$ is a shortcut for $\langle (SA, WA), SA \neq WA \rangle$ and $SA \neq WA$ can be fully enumerated in turn as

$$\{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)\}$$

Example: Map-Coloring



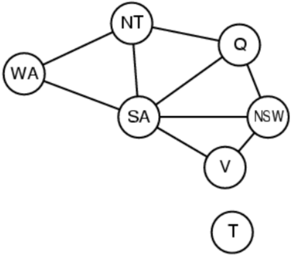
Solutions are assignments satisfying all constraints, e.g.,

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

Constraint graph（约束图）

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

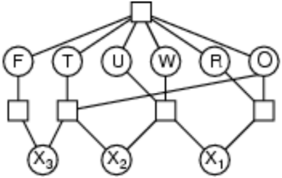
- Discrete variables
  - finite domains 有限区域:
    - n variables, domain size $d \to O(d^n)$ complete assignments
    - e.g., Boolean CSPs, incl. Boolean satisfiability (NP-complete)
  - infinite domains 无限值域 (integers, strings, etc.)
    - e.g., job scheduling, variables are start/end days for each job
    - need a constraint language（约束语言）
    - 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

- Unary（一元） constraints involve a single variable,
  e.g., $SA \neq green$
- Binary（二元） constraints involve pairs of variables,
  e.g., $SA \neq WA$
- Higher-order constraints involve 3 or more variables,
  e.g., cryptarithmetic（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）
  $\rightarrow$ constrained optimization problems

Example: Cryptarithmetic

```
   T W O
 + T W O
 -------
 F O U R
```



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:

alldiff (F,T,U,W,R,O)

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

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

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

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

where X_1, X_2, and X_3 are auxiliary variables representing the digit carried over into the tens, hundreds, or thousands column.

Real-world CSPs

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

Dumb

Exponential time d^n

But complete

can we be clever about exponential time algorithms?

Table of Contents

CSP examples

Backtracking search for CSPs

Problem structure and problem decomposition

Local search for CSPs

Standard search formulation (incremental 增量形式化)

- Let's start with the straightforward approach, then fix it
- States are defined by the values assigned so far
  - Initial state: the empty assignment, $\emptyset$
  - Successor function: assign a value to an unassigned variable that does not conflict with current assignment
    $\rightarrow$ 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
   $\rightarrow$ 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!
   d is the maximum size of the domain

Backtracking search

- 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 d^n 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 $n \approx 25$

Backtracking search

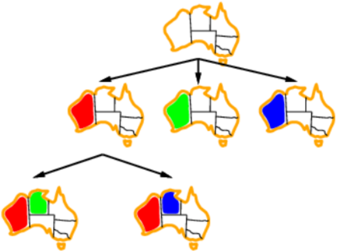
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

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
```

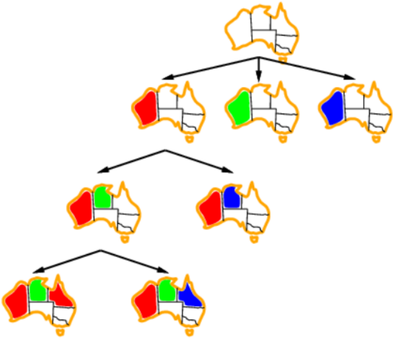
Backtracking example

Backtracking example

Backtracking example



Backtracking example



Backtracking example

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

Enumerate	a	b	c	
	1	1	1	$\times$
	1	1	0	$\times$
	1	0	1	$\times$
	1	0	0	$\times$
	0	1	1	$\times$
	0	1	0	$\times$
	0	0	1	$\times$
	0	0	0	$\surd$

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

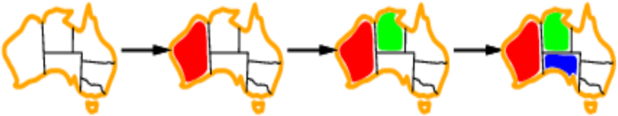
Improving backtracking efficiency

General-purpose methods can give huge gains in speed:

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

Minimum remaining values

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



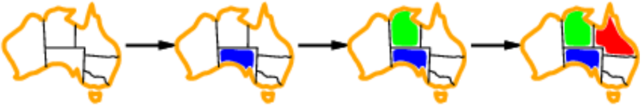
- Why min rather than max?
- Called most constrained variable
- "Fail-fast" ordering

Degree heuristic（度启发式）

Tie-breaker among MRV variables

Degree heuristic:

choose the variable with the most constraints on remaining variables



Improving backtracking efficiency

General-purpose methods can give huge gains in speed:

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

Least constraining value

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

Improving backtracking efficiency

General-purpose methods can give huge gains in speed:

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

Forward checking—前向检验

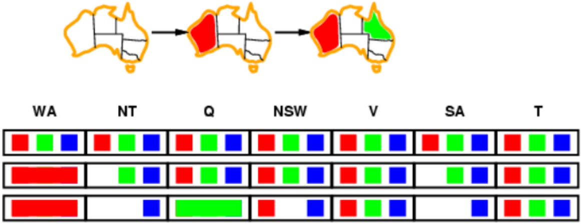
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values

Forward checking—前向检验

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

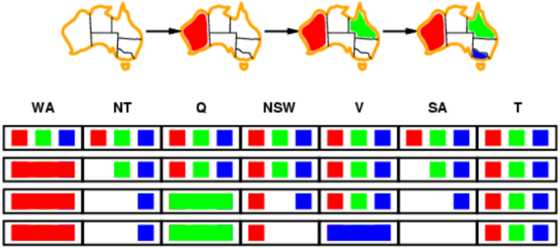
Forward checking—前向检验

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



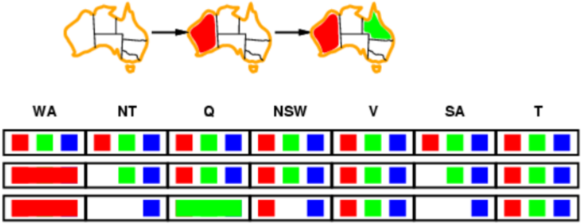
Forward checking—前向检验

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



Constraint propagation —约束传播

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

Constraint propagation: inference in CSPs

Constraint propagation: using the constraints to reduce the number of legal values for a variable, which in turn can reduce the legal values for another variable, and so on

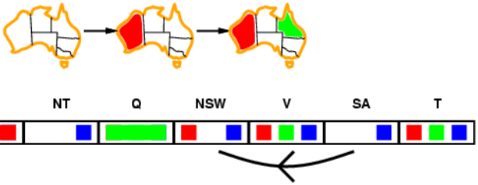
- A single variable is node-consistent, if all the values in the variable's domain satisfy the variable's unary constraints
- A variable in a CSP is arc-consistent with respect to another variable, if every value in its domain satisfies the variable's binary constraints for some value of the other variable
- A two-variable set $\{X_i, X_j\}$ is path-consistent with respect to a third variable X_m if, for every assignment $\{X_i = a, X_j = b\}$ consistent with the constraints on $\{X_i, X_j\}$, there is an assignment to X_m that satisfies the constraints on $\{X_i, X_m\}$ and $\{X_m, X_j\}$
- A CSP is k-consistent if, for any set of $k-1$ variables and for any consistent assignment to those variables, a consistent value can always be assigned to any kth variable
- A CSP is strongly k-consistent if it is k-consistent and is also $(k-1)$-consistent, $(k-2)$-consistent, ..., all the way down to 1-consistent.
- Global constraint is one involving an arbitrary number of variables (but not necessarily all variables)
  For examples, the *Alldiff* constraint says that all the variables involved must have distinct values

Arc consistency —弧相容

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

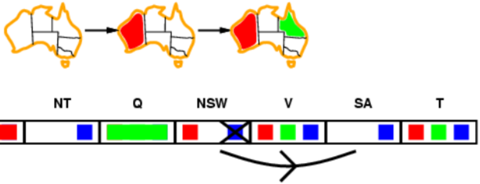


Arc consistency —弧相容

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

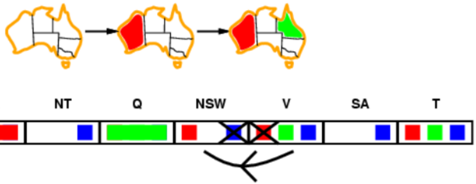


Arc consistency —弧相容

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



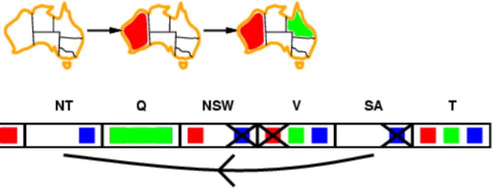
If X loses a value, neighbors of X need to be rechecked

Arc consistency —弧相容

Simplest form of propagation makes each arc consistent

$X \to Y$ 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

function AC-3(*csp*) **returns** the CSP, possibly with reduced domains
  inputs: *csp*, a binary CSP with variables $\{X_1, X_2, \ldots, X_n\}$
  local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

  while *queue* is not empty **do**
    $(X_i, X_j) \leftarrow$ REMOVE-FIRST(*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*

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

Improving backtracking efficiency

General-purpose methods can give huge gains in speed:

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

Table of Contents

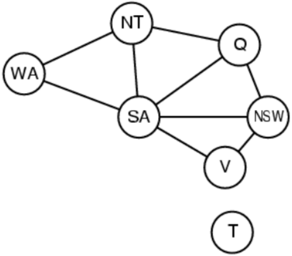
CSP examples

Backtracking search for CSPs

Problem structure and problem decomposition

Local search for CSPs

Problem structure

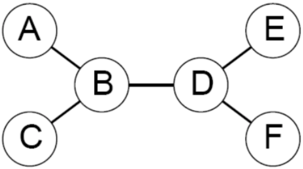


Tasmania and mainland are independent subproblems
Identifiable as connected components（连通域） of constraint graph
Can reduce the search space dramatically

Problem structure cont'd

- 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

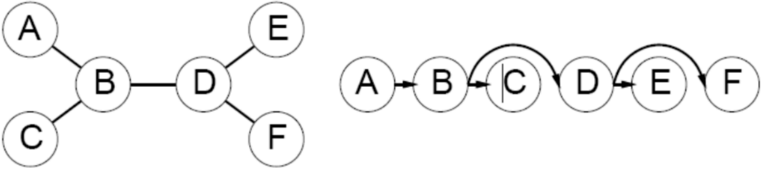


Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n \cdot d^2)$ time
任何一个树状结构的CSP问题可以在变量个数的线性时间内求解

- 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

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



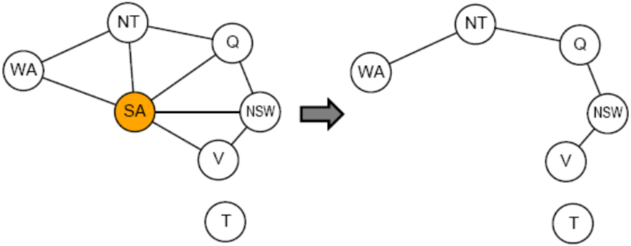
2. Apply arc-consistency to (X_k, X_i), when X_k is the parent of X_i
For i from n down to 2, apply $REMOVEINCONSISTENT(Parent(X_i), X_i)$

3. Now one can start at X_1 assigning values from the remaining domains without creating any conflict in one sweep through the tree!
For i from 1 to n, assign X_i consistently with $Parent(X_i)$

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

Nearly tree-structured CSPs

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 → 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

Tree Decomposition

- Decompose problem into a set of connected sub-problems, where two sub-problems are connected when they share a constraint
- Solve sub-problems independently and combine solutions

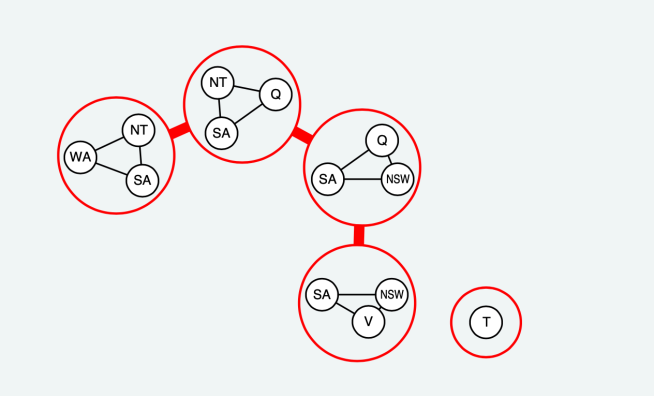


Table of Contents

CSP examples

Backtracking search for CSPs

Problem structure and problem decomposition

Local search for CSPs

Iterative algorithms for CSPs

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

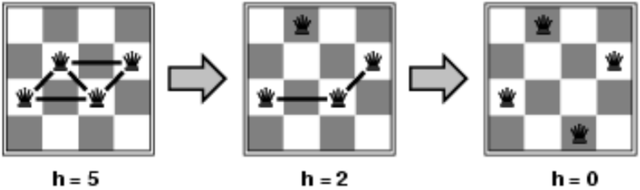
Example: 4-Queens

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

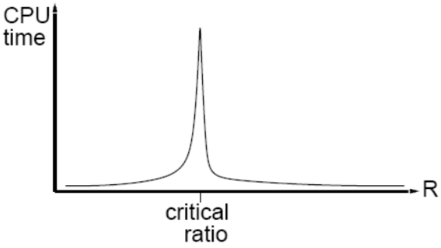


Performance of min-conflicts

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

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

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

Speedup 2: minmax optimization

Putweights 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

Speedup 2: minmax optimization

# 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

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

- 6.5 （第三版）
- 6.11, 6.12（第三版）