Artificial Intelligence

Lecture 5

Ph. D. Lect. Horia Popa Andreescu

2017-2018   3rd year, semester 5

- The slides for this lecture are based (partially) on chapter 5 of the Stuart Russel Lecture Notes [R, ch5], and on the same chapter from Russel & Norvig's book [RN, ch. 5]

Constrain satisfaction problem (CSP)

- CSP are solved with general-purpose heuristics rather than problem-specific ones.
- Formally, a CSP is defined by a set of constrains C1, C2, …, Cm; a set of variables X1, X2, …, Xn having as possible values in the domains D1, D2, …,Dn.
- Each constraint Ci involves some subset of the variables and specifies an allowable combination of the values for that subset.
- A state of the problem is defined by an assignment of values to some or all the variables. (Xi=vi, Xj=vj, …)
- An assignment that doesn't violate any constraint is called consistent or legal assignment.
- A solution to the problem is a complete assignment (all variables have legal values), that satisfies all the constraints.

- Some CSP also require a solution that maximizes some objective function.
- Example:
  - Graph coloring problem – we have a map and we want to color the countries such as no two neighboring countries are colored with the same color.
  - Crypto arithmetic puzzle problem: the letters in the problem are in fact digits which are to be determined, so that the addition remains correct

```
  TWO+
  TWO=
 FOUR
```

Varieties of CSP [R, 5/7]

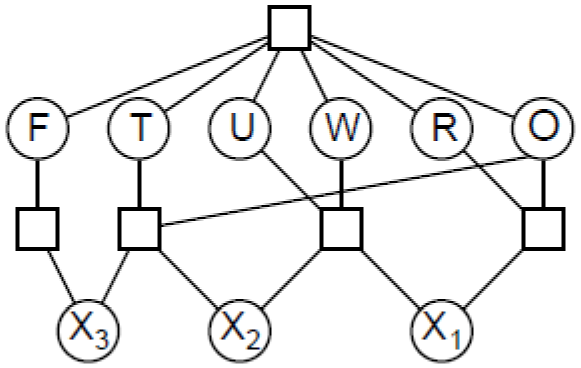
- Discrete variables
  - On finite domains – with complete assignments
    - E.g. boolean CSP, including boolean satisfiability
  - On infinite domains (integers, strings, etc.)
    - E.g. job scheduling, variables are start / stop day for each job
    - Need a constraint language, e.g. StartJob1+5<=StartJob3
    - Linear constraints are solvable, nonlinear are undecidable
- Continuous variables
  - E.g. start/end times for Hubble Telescope observations
  - Linear constraints are solvable in polynomial time by LP methods
- Simple CSP
- Distributed CSP

Varieties of constraints [R, 5/8]

- Unary constraints involving a single variable
  - e.g. SA is not green
- Binary constraints involving pairs of variables
  - e.g. SA different than WA
- Higher order constraints involving 3 or more variables
  - e.g. Cryptarithmetic column constraints
- Preferences (soft constraints) e.g red is better than green
  - They are representable by a cost for each variable assignment → constrained optimization problems

Example: cryptarithmetic [R, 5/9]

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



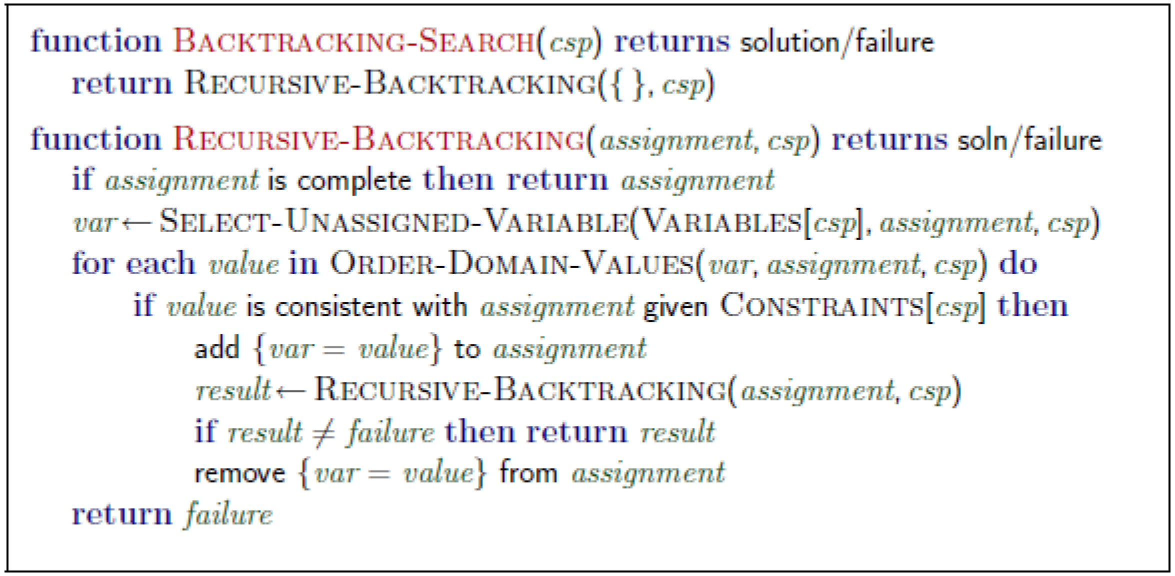
Variables: $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$
Domains: $\{0,1,2,3,4,5,6,7,8,9\}$
Constraints
 alldiff(F,T,U,W,R,O)
 $O+O=R+10\cdot X_1$, etc.

STANDARD SEARCH FORMULATION [R, 5/11]

- Let's start with the straightforward, dumb 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 (not doable!)
  - 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
- 4) b=(n-l)d at depth l, hence n!d^n leaves!!!!

BACKTRACKING SEARCH [R, 5/13]

- Dept-first search for CSP with single-variable assignments is called **backtracking search (BS).**
- BS is the basic uninformed algorithm for CSPs
- It can solve n-queens problem for approx. n=25



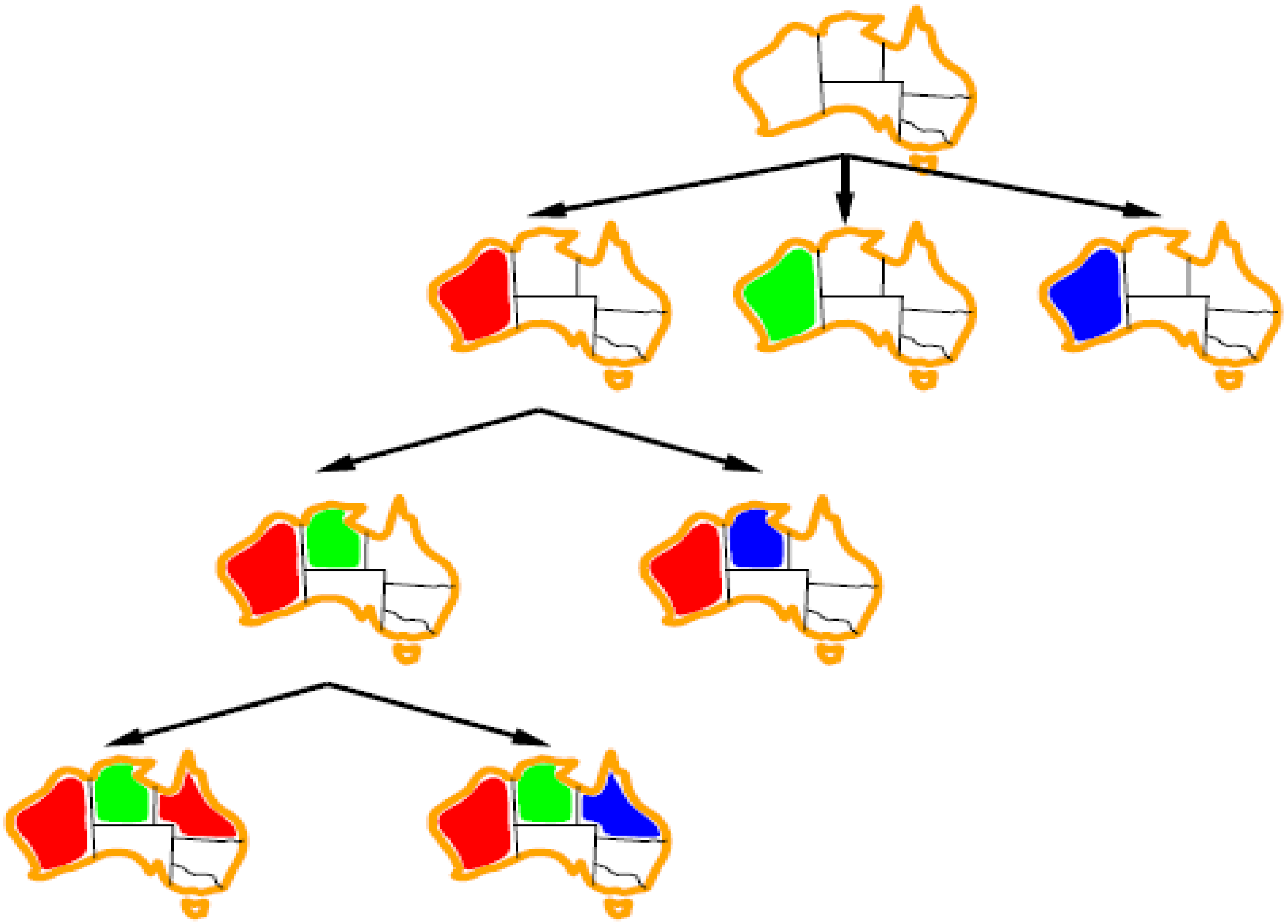
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

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
```

EXAMPLES, OPTIMIZATION

- Slides 14-36 from [R, 5/14-36]

Backtracking example (R-5-17)

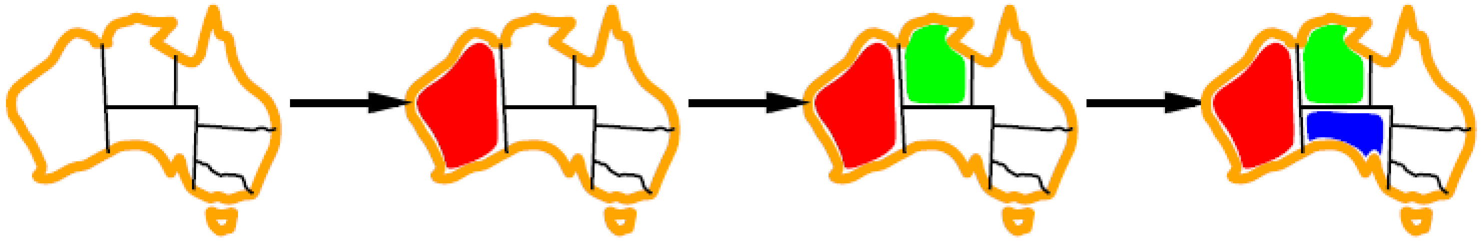


Improving backtracking efficiency (R-5-18)

- Based on the specific problem that we try to solve we can try improving the Backtracking (BT) algorithm by considering:
  - Which variable should be assigned next?
  - In what order should its values be tried?
  - Can we detect inevitable failure early?
  - Can we take advantage of problem structure?
- Improved BT:
  - Minimum remaining values (MRV)
  - Degree heuristic
  - Least constraining value
  - Forward checking
  - Constraint propagation
  - Arc consistency

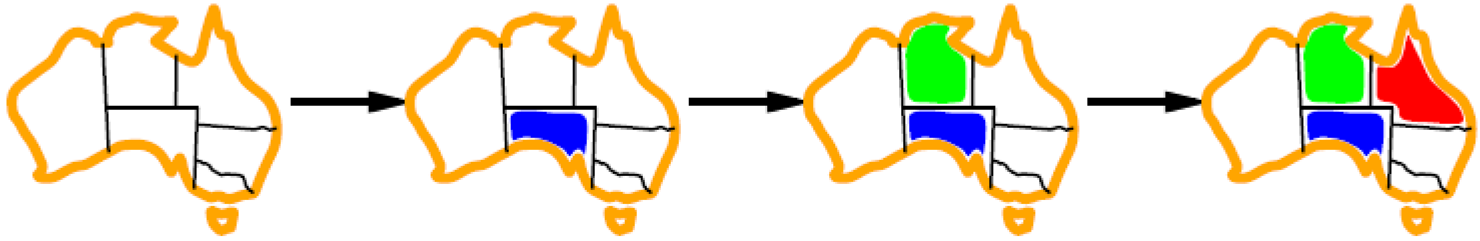
Minimum remaining values (R-5-19)

- Consists of: choosing the variable with the fewest legal values



Degree heuristic (R-5-20)

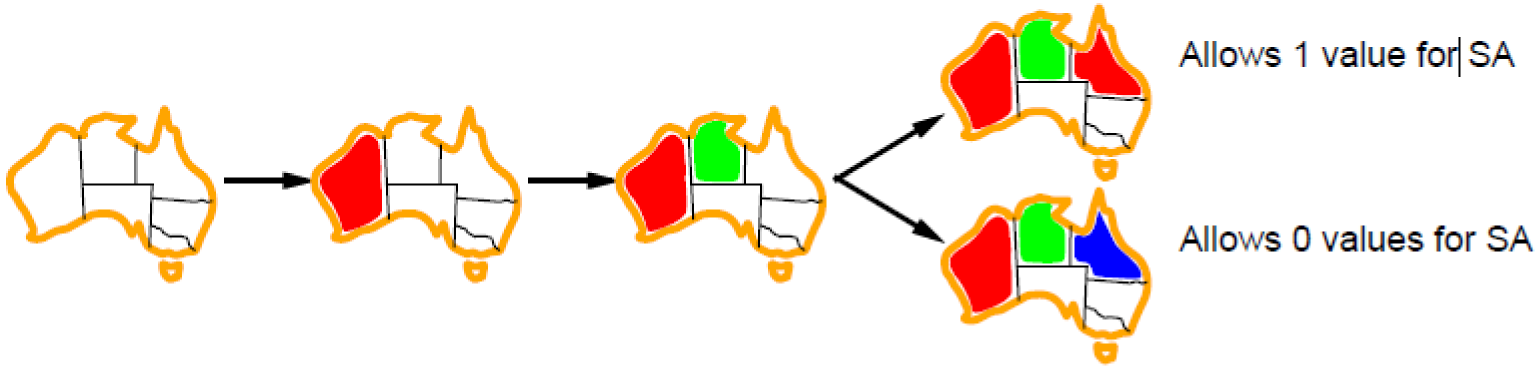
- Tie-breaker among MRV variables
- Degree heuristic:
  - choose the variable with the most constraints on remaining variables



Least constraining value (R-5-21)

- Given a variable, choose the least constraining value:

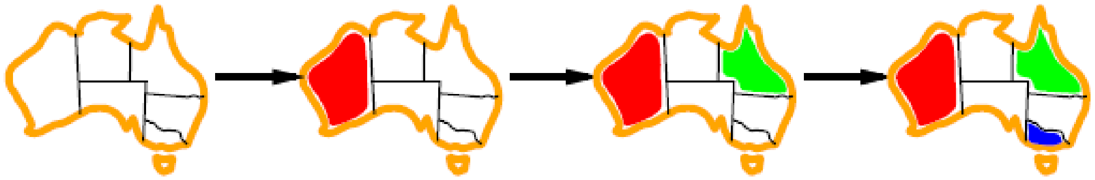
  the one that rules out the fewest values in the remaining variables



- Combining these heuristics makes 1000 queens feasible

Forward checking (R-5-25)

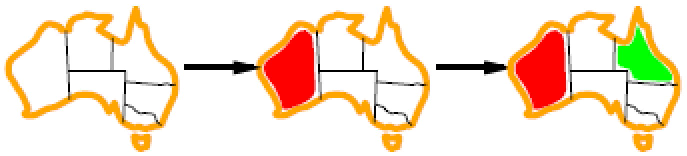
- Idea: Keep track of remaining legal values for unassigned variables.
- Terminate search when any variable has no legal values



WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦
🟥	🟦	🟩	🟥	🟦		🟥🟩🟦

Constraint propagation (R-5-26)

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

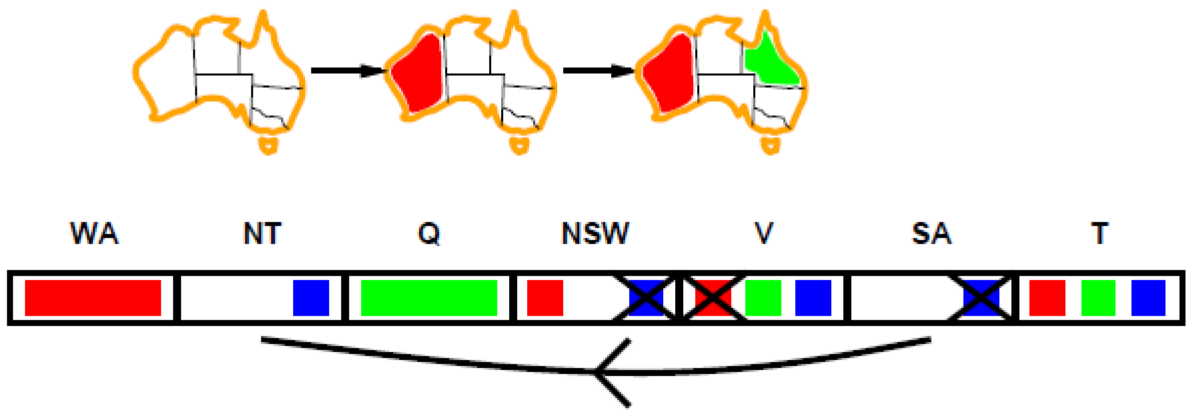


WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦

- NT and SA cannot both be blue!
- Constraint propagation repeatedly enforces constraints locally

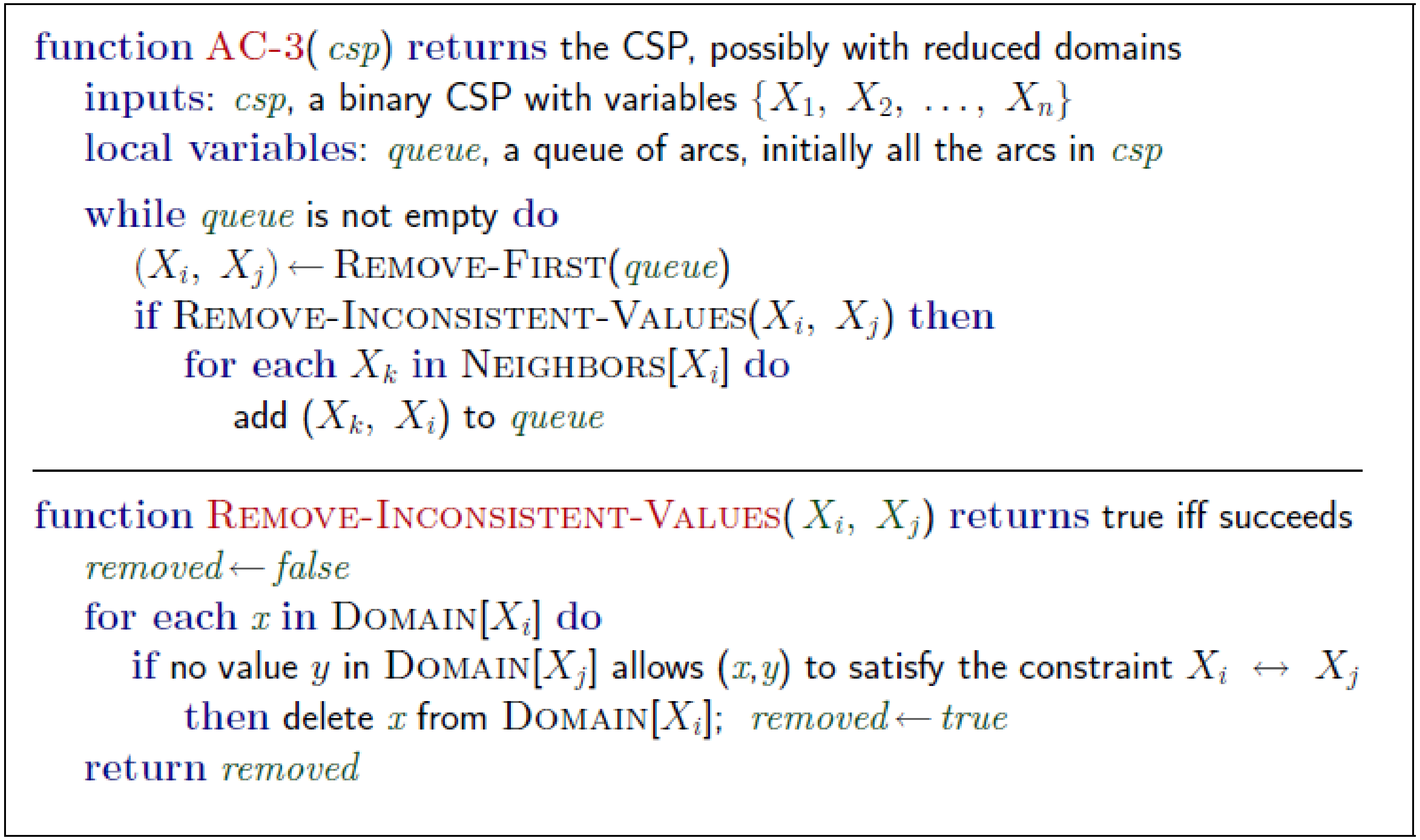
Arc consistency (R-5-30)

- Simplest form of propagation makes each arc consistent
- X → 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 (R-5-31)



```
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
```

$O(n^2d^3)$, can be reduced to $O(n^2d^2)$ (but detecting **all** is NP-hard)

Exploit the problem structure (R-5-33)

- Suppose each subproblem has c variables out of n total
- Worst-case solution cost is n=c dc, linear in n
- E.g., n=80, d=2, c=20
  - 2^{80} = 4 billion years at 10 million nodes/sec
  - 4 $*2^{20}$ = 0.4 seconds at 10 million nodes/sec
- The second variant is after decomposing the original problem into subproblems.

Problem decomposition into subproblems (RN-154)

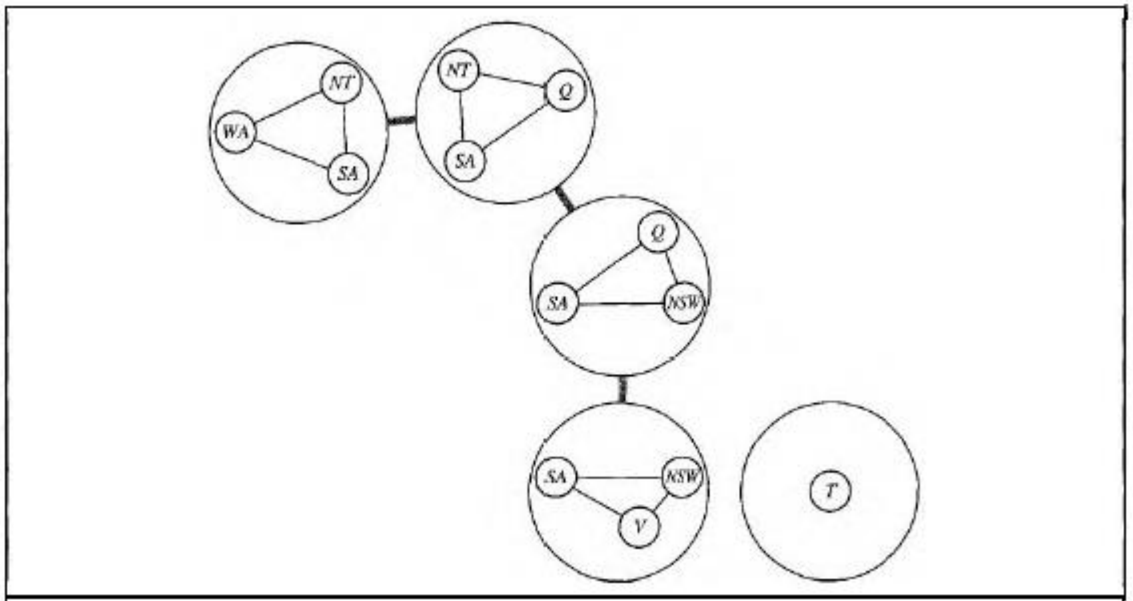


Figure 5.12 A tree decomposition of the constraint graph in Figure 5.11(a).

- We view each subproblem as a mega-variable whose domain is the set of all solutions for the subproblem. E.g. for the leftmost problem, one solution is { WA = *red, SA = blue, NT = green*}.
- Then, we solve the constraints connecting the subproblems using the efficient algorithm for trees.
- The constraints between subproblems simply insist that the **subproblem solutions agree on their shared variables**.

ITERATIVE ALGORITHMS FOR CSPS [R-5-37]

- Hill-climbing, simulated annealing typically work with “complete" states, 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 heuristic:
    - choose value that violates the fewest constraints
    - i.e., hillclimb with h(n) = total number of violated constraints

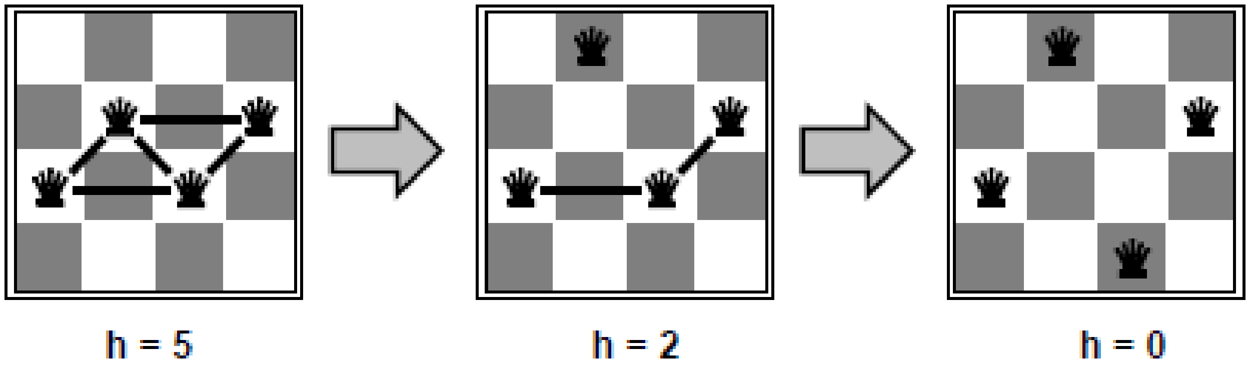
EXAMPLE: 4-QUEENS [R-5-38]

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

Operators: move queen in column

Goal test: no attacks

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

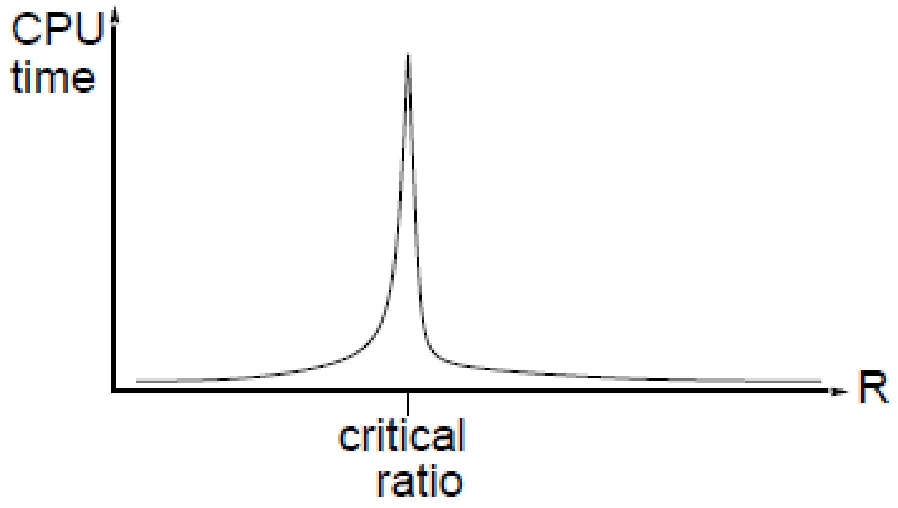


PERFORMANCE OF MIN-CONFLICTS [R-5-39]

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

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



Distributed CSP (DCSP)

- A DCSP problem is a CSP problem, where the variables are shared among agents, e.g. each agent has one variable assigned.
- A solution is a complete assignment for the variables, that satisfies the constraints
- The solution is attained by means of agent communication, each agent communicating the assignments it has made along with the known list of neighbors assignments
- The agents are connected if there is a constrain between them.
- Connections between agents may appear later, depending on the algorithm used

DSCP ALGORITHMS

- There are a series of algorithms in the literature, most known are:
  - ABT (Asynchronous Backtracking) [Yok98]
  - ABT kernel [Bes05]
  - Distributed Dynamic Backtracking (DisDB) [Bes05]
- ABT algorithm:
- The Asynchronous Backtracking algorithm uses 3 types of messages [Yok98]:
  - the OK message, which contains an assignment variable–value, is sent by an agent to the constraint-evaluating-agent in order to see if the value is good.
  - the nogood message which contains a list (called nogood) with the assignments for which the inconsistency was found is being sent in case the constraint-evaluating-agent found an unfulfilled constraint.
  - the add-link message, sent to announce the necessity to create a new direct link, caused by a nogood appearance.

ABT FAMILY ALGORITHMS

- Starting from the algorithm of asynchronous backtracking (ABT), several derived techniques were suggested [Yok98], based on this one and known as the ABT family.
- They differ in the way that they store nogoods, but they all use additional communication links between unconnected agents to detect obsolete information.
- These techniques are based on a common core (called ABT kernel) from which some of the known techniques can be obtained, including the algorithm of asynchronous backtracking, by eliminating the old information among the agents.
- The ABTkernel algorithm requires, like ABT, that constraints are directed- from the value-sending agent to the constraint-evaluating agent-forming a directed acyclic graph. Agents are ordered statically in agreement with constraint orientation.
- The ABT kernel algorithm, is a new ABT-based algorithm that does not require to add communication links between initially unconnected agents. The ABT kernel algorithm is sound but may not terminate (the ABT kernel may store obsolete information).

ABT FAMILY ALGORITHMS (II)

- A first way to remove obsolete information is to add new communication links to allow a nogood owner to determine whether this nogood is obsolete or not. These added links were proposed in the original ABT algorithm[Yok98][Bes05].
- A second way to remove obsolete information is to detect when a nogood could become obsolete. This solution leads to the DisDB algorithm. No new links are added among the agents. To achieve completeness, this algorithm has to remove obsolete information in finite time. To do so, when an agent backtracks, it forgets all nogoods that hypothetically could become obsolete[Bes05].

- A classification of the main DCSP algorithms can be found at [3], with the observation that those algorithms had been first introduced between 1992 and 2006. In the meantime variants, hybrid versions, optimizations of those algorithms were studied.

Bibliography

- [RN] Russel S., Norvig P. – Artificial Intelligence – A Modern Approach, 2nd ed. Prentice Hall, 2003 (1112 pages)
- [R] Stuart Russel – Course slides (visited oct. 2012 at http://aima.cs.berkeley.edu/instructors.html#homework)
- [Yok8]. Yokoo, M., Durfee, E. H., Ishida, T., Kuwabara, K. The distributed constraint satisfaction problem: formalization and algorithms. IEEE Transactions on Knowledge and Data Engineering 10(5) (1998) 673--685
- [Bes05] Bessiere, C., Brito, I., Maestre, A., Meseguer, P. Asynchronous Backtracking without Adding Links:A New Member in the ABT Family. A.I., 161 (2005) 7--24.
- [3] http://en.wikipedia.org/wiki/DisCSP