Constraint Satisfaction Problems

- Objectives
  - Constraint satisfaction problems
  - Backtracking
  - Iterative improvement
  - Constraint propagation
- Reference
  - Russell & Norvig: Chapter 6.
  - R. Dechter, Constraint Processing, Morgan Kaufmann, 2003.

Constraints in Practice

- We encounter constraints regularly in daily life.
  - Ex Selecting courses to take
  - Ex Replacing a light bulb
- Problems involving constraints can be complex.
  - Ex University course timetabling
  - As problem complexity grows, intelligent agents offer a more productive alternative for finding solutions.
  - Generally, problems involving constraints are NP-hard.

University Course Timetabling

- Hundreds of courses are offered each semester.
- Room-slot: No two lectures can be offered in the same room at the same time slot.
- Instructor: Lectures by an instructor cannot be scheduled into the same time slot.
- Course: Lectures of a course cannot be offered at the same time slot.
- Course group: For courses to be co-taken, their lectures cannot be scheduled into the same time slots.

Constraint Network

- For agents to solve these problems, we need to
  - model their environments, and
  - develop solution algorithms working on the model.
- A constraint network (CN) has two components:
  1. a set V of variables with associated domains, and
  2. a set C of constraints.
- $V = \{x_1,\ldots,x_n\}$ is a finite set of variables.
  - Each variable x_i is associated with a domain of possible values.
  - We focus on discrete variables with finite domains.
    - Denote the domain of x_i by $D_i = \{x_{i_1},\ldots,x_{i_k}\}$.

Constraints

- An assignment over a subset $X \subseteq V$ is a combination of values for variables in X.
  - X is the scope of the assignment.
  - An assignment is complete, if its scope $X = V$.
- A constraint C_i is a set of acceptable assignments over a subset $X \subseteq V$ of variables.
  - Subset X is the scope of constraint C_i .
- Arity of a constraint is the cardinality of its scope.
  - Unary and binary constraints
- $C = \{C_1,\ldots,C_m\}$ is a set of constraints.
  - A binary CN has only unary and binary constraints.

Consistency of Assignments

- Consider assignments $\underline{x}$ over (W,N,Q,S) = (r,g,r,b) and $\underline{y}$ over (W,S) = (r,b).

  Then $\underline{y}$ is the projection of $\underline{x}$ onto {W,S} .
  - Is $\underline{z}$ over (W,N) = (r,b) the projection of $\underline{x}$ onto {W,N} ?
- Consider assignments $\underline{x}$ over (W,N,S) = (r,g,b) and $\underline{y}$ over (N,S,R) = (g,b,g).

  Then $\underline{x}$ and $\underline{y}$ are consistent, since their projections to scope intersection $\{W,N,S\} \cap \{N,S,R\}$ are equal.
  - Is $\underline{z}$ over (W,N,Q,R) = (r,b,r,b) consistent with $\underline{x}$?

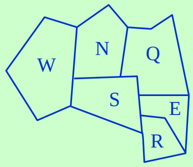
Constraint Satisfaction

- A constraint C_k of scope $Y \subset V$ is irrelevant to an assignment $\underline{x}$ over $X \subset V$ if $X \cap Y = \emptyset$.
- An assignment $\underline{x}$ over $X \subset V$ satisfies a relevant constraint C_k of scope $Y \subset V$ if there exists an assignment $\underline{y}$ in C_k s.t. $\underline{x}$ and $\underline{y}$ are consistent.
- An assignment that satisfies all relevant constraints is called a consistent or legal assignment.
- A solution of a CN is a complete, consistent assignment.
- A CN is inconsistent if it has no solution.

Constraint Satisfaction Problems

- Tasks to decide whether a solution exists for a CN and to find solutions if they exist are referred to as constraint satisfaction problems (CSPs).
- Focus
  - Whether a solution exists and finding one if so
- Ex Map coloring
- Ex The n-queens
- Other CSPs

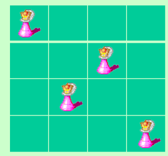
Ex Map Coloring CSP



- Color each region with one of {r,g,b} s.t. no adjacent regions have the same color.
- Map coloring CN
  - What is V and what are variable domains?
  - What is an assignment over X = {W,N,S}?
  - What is the constraint C_1 btw W and N?
  - What are the scopes for constraints in C?
  - Does assignment $\underline{x}$ = {W=r,N=r,S=b,E=g} satisfy C_1?
  - Is assignment $\underline{y}$ = {N=r,Q=g,S=b,E=g} legal?
  - What is a solution of the CN?

Ex The n-queen

- Place n queens on n×n chessboard s.t. no queen attacks another.
- Constraint network for 4-queen
  - What are the variables and their domains?
  - What is the constraint btw x_1 and x_3 ?
  - How many constraints are in C?
  - What is a solution?



Other CSPs

- Crossword puzzles
- 3SAT
- Job shop scheduling
- Radio link frequency assignment
- Space telescope scheduling
- 3D interpretation of 2D drawing
- Hospital nurse scheduling
- Airline flight scheduling
- Floor plan layout
- Automobile transmission design

Constraint Graphs

- Constraints in CNs can be depicted as constraint graphs, and they are useful in solving CSPs.
  - Common forms include primal graph and hypergraph.
- Primal graph: Each node represents a variable and each link connects two constrained variables.
- Hypergraph: Each node represents a variable. Each hyperlink represents a constraint and connects all variables in its scope. It is drawn by a link from a box to each variable in the scope.

Ex Crypt-Arithmetic Problem

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

- Substitute each letter by a distinct digit without leading zero s.t. the sum is arithmetically correct.
- Constraint network and constraint graphs
  - ❑ What are the variables?
  - ❑ How to represent requirement for distinct digit?
  - ❑ How to represent requirement for no leading zero?
  - ❑ How to represent arithmetic sum?
  - ❑ What is the primal graph?
  - ❑ What is the hypergraph?

Solving CSPs by Search

- Tree search such as BFS, DFS, etc can be applied to CSPs.
  - ❑ How?
  - ❑ How many leaf nodes are at level n+1?
  - ❑ How many complete assignments are there?
- Observation
  - ❑ Unlike problems such as 8-puzzle, neither order of variable assignment, nor search path matters for CSPs.
  - ❑ Hence, it suffices for search to focus on a single order.

Chronological Backtracking

- Idea: Depth-first search that assigns one variable each time (forwards) and backtracks when a variable has no legal values to assign.
- A recursive algorithm
  - ❑ Ex The 4-queen
  - ❑ Open: variable order, value order, backtracking depth
- Chronological backtracking
  - ❑ Always backtrack to the most recent value assignment
  - ❑ Termination property


```
backtracking(config[]) {    // assume access to CN
  if config is complete, return config;
  v = getVariable(config);
  vu[] = orderDomainValue(v, config);
  for each u in vu,
    if config ⋈ (v=u) is legal,
      config = config ⋈ (v=u);
      result = backtracking(config);
      if result != null, return result;
      remove (v=u) from config;
  return null;
}
```

Complexity of Backtracking

- Suppose |V| = n and $|D_i| \le k$.
- In the worst case, almost the full search tree is explored.
- What is the number of nodes in full search tree?
  - ❑ How many levels are in the full search tree?
  - ❑ What is the branching factor?
- Sum of geometric series
  - ❑ Sum $k^0 + k^1 + \ldots + k^n$ equals $(k^{n+1}-1)/(k-1)$.
- Time complexity of backtracking

Variable Ordering

- Does the order of variables matter to efficiency of chronological backtracking?
- Ex Map coloring with value order (r,g,b), except for variable Q, it is (b,g,r).
  - ▪ Order 1: (W,N,Q,E,R,S)
  - ▪ Order 2: (W,N,S,Q,E,R)
- Styles of variable ordering
  - ❑ Fixed
  - ❑ Dynamic

Minimum Remaining Values (MRV) Heuristic

- Choose next variable x with the fewest legal values.
  - ❑ Ex Map coloring
- Rational behind MRV
  - ❑ If x has no legal value, search tree is pruned immediately.
  - ❑ Otherwise, x is most likely to cause failure soon, thereby pruning search tree.
- Overhead
  - ❑ Maintain the number of legal values for each var.
  - ❑ After y is assigned, check each adjacent, remaining variable, and updates its number of legal values.
  - ❑ Upon backtracking, recover values of remaining variables.

Maximum Degree (MDeg) Heuristic

- Choose next variable x that involves the largest number of constraints with unassigned variables.
  - ❑ Ex Map coloring
- Rational: Reduce branching factor on future choices.
- Overhead: After assigning y, update degree counts for remaining variables constrained by y.
  - ❑ Upon backtracking, recover counts of remaining variables.
- MRV and MDeg can be combined in chronological backtracking, with MRV as the primary heuristic and MDeg as the tie-breaker.

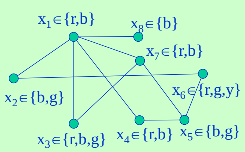
Value Selection

- Given a variable order, does the order in which values are assigned matter to search efficiency?
- Ex Map coloring with variable ordering (W,N,Q,E,R,S) and value order (r,g,b)
- Styles of value ordering
  - ❑ Fixed
  - ❑ Dynamic

Forward Checking

- Idea: Rather than assigning blindly, check forward a little before assigning.
- To assign variable x with x=u, for each unassigned variable y connected to x by a constraint C_i, delete from D_y each value inconsistent with x=u by C_i .
  - ❑ If D_y becomes empty, consider another value of x.
  - ❑ If D_y is non-empty for each y, assign x=u.
- Ex Map coloring with order (W=r,Q=g,R=b,…)
- Benefit: Avoid a value that will cause failure before it is assigned, and hence save computation.

Look Back in Search



- When a branch of search fails, chronological backtracking backs to the preceding variable.
  - ❑ Ex Inefficiency of chronological backtracking
- Idea of improvement: jump back further
  - ❑ Ex Jump back to x_3 , x_2, or x_1
- Challenge: What is the adequate jump back point?
  - ❑ Intuition: Back all the way to source of failure

Dead-end and Conflict Set

- Let $\underline{u}_i$ =(u_1,…,u_i) be a consistent assignment for x_1,…,x_i and x be an unassigned variable. If there is no value u in D_x s.t. ($\underline{u}_i$, x=u) is consistent, then
  - ❑ $\underline{u}_i$ is a dead-end state relative to x,
  - ❑ x is a dead-end variable relative to $\underline{u}_i$, and
  - ❑ $\underline{u}_i$ is a conflict set of x.
- If $\underline{u}_i$ does not contain a subtuple that is in conflict with x, then $\underline{u}_i$ is a minimal conflict set of x.

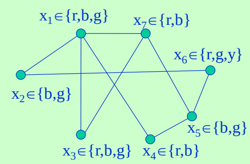
Culprit Variable

- Let $\underline{u}_i = (u_1,\ldots,u_i)$ be an assignment. A prefix assignment of $\underline{u}_i$ is $\underline{u}_k = (u_1,\ldots,u_k)$, where $k \le i$.
- Let $\underline{u}_i = (u_1,\ldots,u_i)$ for $x_1,\ldots,x_i$ be a dead-end state relative to x. The culprit variable of $\underline{u}_i$ is x_k where

  $k = \min \{ j \mid j \le i$ and $\underline{u}_j$ is a conflict set of $x \}$.
  - ❑ Ex What is the culprit variable of assignment (x_1=r, x_2=b, x_3=b, x_4=b, x_5=g, x_6=r) relative to x_7?
- Culprit variable is the adequate jump back point.
  - ❑ Why?

Backjumping Algorithm

- The backjumping algorithm
  - ❑ Behaviour when there is no backing
  - ❑ Behaviour when there is backjumping
    - Ex A coloring problem
- Does backjump() jump back to culprit variable?




```
backjump() {                              // assume access to CN
   i =1; Di' = Di ; latest = 0;
   while 1 ≤ i ≤ n,
         xi = selectValue(i);          // will update latest
         if xi == null, then { i = latest; latest--; }
         else { i++; latest = 0; if i ≤ n, Di' = Di ; }
   end while;
   if i == 0, return "no solution";
   else return assignment over {x1,…,xn};
}
```



```
selectValue(i) {    // assume access to Di' and latest
   while Di' ≠ {},
         select u ∈ Di' and remove u from Di' ;
         consistent = true; k = 1;
         while k<i and consistent,
            if k > latest, then latest=k;
            if (uk , xi=u) is inconsistent, consistent = false;
            else k++;
         if consistent, return u;
   return null;
```

Local Search

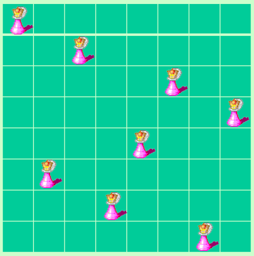
- Backtracking builds up a consistent, partial solution by assigning one variable at a time until completion.
- Local search iteratively improves a complete but inconsistent assignment, one variable at a time.
- Technical issues
  - Which variable should be improved next?
  - Which value of the variable should be selected?
- Abbreviation
  - NCVs: Number of Constraint Violations

Time Bounded Min-Conflicts Algorithm

```
minConflictTB(c[], maxSteps) { // c: constraints in CN
  current = a complete assignment;
  for i=1 to maxSteps,
      if current is legal, return current;
      v = randomly selected variable violating c;
      u = value of v minimizing NCVs with ties
          broken randomly;
      set v = u in current;
  return failure;
}
```


Example and Properties

- Ex Solving 8-queens
- Can agent avoid repeating the same assignments?
- An algorithm for CSPs is complete if it always finds a solution when one exists, and always terminates when no solution exists.
- Is minConflictTB complete?



Nogood

- An intelligent CSP agent should learn to avoid regenerating a conflict set.
- Given a CN, any partial assignment that does not appear in any solution is a nogood.
  - Is every conflict set a nogood?
  - Is every nogood a conflict set?
- Assignment x over X satisfies a nogood y over Y, if $X \supseteq Y$ and x and y are consistent.
  - Ex Does x over (u,v) = (1,2) satisfy constraint C_i over (u,v,w) = {(1,2,3), (2,4,6)}?
  - Ex Does x satisfy nogood y over (u,v,w) = (1,2,3)?

Min-Conflicts Algorithm With Nogood Learning

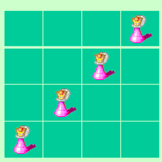
```
minConflictNL(c[]) {      // c[]: constraints from CN
   parSol[] = ∅;
   varLeft[] = a complete assignment;
   return minConflictNL(parSol, varLeft, c);
}
```

- Each element of parSol and varLeft is a (variable, value) pair.
- parSol and varLeft have no variable in common.
- parSol ⋈ varLeft is a complete assignment.
- parSol is a consistent partial assignment.


```
minConflictNL(parSol[], varLeft[], c[]) {
   nogood[] = ∅;
   loop
     if a[] = varLeft ⋈ parSol is legal, return a[];
     (v,u) = varLeft[k] violating c;
     vu[] = values of v consistent with parSol wrt c & nogood;
     if vu is emtpy,
        if parSol is empty, return no-solution;
        add parSol to nogood;
        move most recently added element of parSol to varLeft;
     else
        x = vu[i] minimizing NCVs with varLeft wrt c;
        remove (v,u) from varLeft; add (v,x) to parSol;
   end loop
}
```


Example and Properties

- Ex The 4-queens



- Is minConflictNL complete?
  - ❑ Does it always find a solution when one exists?
  - ❑ Does it terminate when there is no solution?

Relative Arc Consistency

- Forward checking propagates implications of a constraint from variable x to y.
- Constraint propagation is a general approach to propagate implications of constraints among variables, and arc consistency is an effective method of constraint propagation.
- Let C_{xy} be a constraint of scope {x,y}. Variable x is arc-consistent relative to y iff, for every value $u \in D_x$, there exists a value $w \in D_y$ s.t. $(u,w) \in C_{xy}$.
- Ex Constraint C_{xy}: $x < y$, with $D_x = D_y = \{1,2,3\}$
  - ❑ Is x arc-consistent relative to y?

Enforcing Relative Arc-Consistency

- If x is not arc-consistent relative to y, consistency can be enforced by executing algorithm revise():

revise(x, y, C_{xy}) {
  for each u $\in D_x$,
    if there is no w$\in D_y$ s.t. (u,w)$\in C_{xy}$, $D_x = D_x \setminus \{u\}$;
}

- Ex Given C_{xy}: x < y with $D_x = D_y$ ={1,2,3}, make x arc-consistent relative to y.
  - Is y arc-consistent relative to x?
  - How to make an arc consistent in both ways?
- What is the complexity of revise()?

Arc Consistency

- Let C_{xy} be a constraint of scope {x,y}. Variables x and y are arc-consistent, iff x is arc-consistent relative to y and y is arc-consistent relative to x.
- A CN is arc-consistent iff every pair of constrained variables are arc-consistent.
- Arc consistency in a CN can be enforced by algorithm arcConsistency().
- Ex Forward checking in map coloring with order (W=r,Q=g,R=r,…)
  - After assigning W=r, is the CN arc-consistent?
  - After Q=g, what happens if arcConsistency() is run?

Enforce Arc-Consistency

arcConsistency() {
  agenda = {};
  for each pair {x, y} of variables adjacent in CN,
    agenda = agenda $\cup$ {(x,y),(y,x)};
  while agenda ≠ {},
    remove (x,y) from agenda and revise(x, y, C_{xy});
    if D_x is revised,
      for each z adjacent to x in CN and z ≠ y ,
        agenda = agenda $\cup$ {(z,x)};
}

- What is the complexity of arcConsistency()?

The k-consistency

- Can an arc-consistent CN contain inconsistency?
  - Could an arc-consistent CN have no solution?
  - Ex A triangle-structured coloring CSP
- A CN is k-consistent if for every set X of k-1 variables and for every consistent assignment over X, a consistent value can be assigned to any kth variable.
  - 1-consisency = node consisency
  - 2-consisency = arc consisency
  - 3-consisency = path consisency
  - Ex Is CN of 4-queens k-consistent for k=1,2,3?

Strong k-consistency

- A CN is strongly k-consistent if it is i-consistent for all $i \leq k$.
- If a CN with $|V|=n$ is strongly n-consistent, what would happen if backtracking() is applied to it?
- What is the complexity to make a general CN strongly n-consistent?
  - Solving CSPs are NP-hard in general.
- Between strong n-consistency and arc-consistency, a range of middle grounds exist to trade efficiency of search with efficiency to enforce consistency.

Solving Tree-Structured CSPs

- Info on the structure of constraint graph can guide limited consistency checking and effective search.
- Suppose the primal graph of CN is a tree T.
  - Ex A coloring problem where D = {r,g} for each variable
  - How does backtracking() behave with variable ordering (A,B,C,D,E,F)?
- Algorithm solveTreeCN()
  - After 1st loop, whether CN has solution is known.
  - If so, each parent is arc-consistent relative to each child.
  - The 2nd loop is backtracking free.
- Complexity of solveTreeCN()

solveTreeCN() {
  pick a node v and direct T with v as root;
  order nodes s.t., for each node x, $\pi(x)$ precedes x;
  denote the ordering by $r = (x_1, x_2, \ldots, x_n)$;
  for i=n to 2,
    revise($\pi(x_i)$, x_i);
    if domain of $\pi(x_i)$ is empty, return no-solution;
  for i=1 to n, assign x_i a value consistent with the value assigned to $\pi(x_i)$;
  return complete assignment;
}

Cutset Conditioning

- What if a CN is not tree-structured?
  - Can solveTreeCN() be extended to non-tree CNs?
- A cycle cutset of a graph is a subset of nodes whose removal renders the graph a tree.
- Solve binary CNs with cutsetCondition()
  - Ex Map coloring
- Complexity of cutsetCondition()
  - Denote $|V|=n$, $|D_i| \leq d$, and cutset X with $|X|=c$.

```
cutsetCondition() {
   choose a cycle cutset X and denote Z=V\X;
   A(X) = set of consistent assignments of X;
   for each x∈A(X),
      for each y∈Z,
          remove each u∈D_y inconsistent with x ;
      if D_y={} for any y∈Z, continue;
      apply solveTreeCN() to CN over Z;
      if solution z is found, return x ⋈ z ;
    return no-solution;
}
```


Cluster Tree Decomposition

- Decompose CN into subCNs, solve each subCN, and extend subCN solutions to CN by tree solving.

```
solveCNByClusterTree() {
   decompose CN into a cluster tree T;
   for each cluster Q in T,
     find set A(Q) of consistent assignments of Q;
      if A(Q)={}, return no-solution;
   set T as a binary CN;
   apply solveTreeCN() to T;
}
```


Decompose CN Into Cluster Tree

- Decompose V into overlapping clusters s.t. each x∈V is contained in at least one cluster.
- For every constraint in C, its scope must be contained in at least one cluster.
- Organize clusters into a tree T s.t. every two adjacent clusters have common variables.
- Each variable contained in two clusters in T must be contained in all clusters along the path.
- Ex Map coloring

Set Cluster Tree As Binary CN

- For each cluster Q and its assignment set A(Q), create a mega-variable q of domain D_q=A(Q).
  - Ex Map coloring
- For adjacent clusters Q and Y, denote Z=Q∩Y and corresponding mega-variables by q and y.
- Constraint over {q,y} is that their assignments must agree for variables in Z.
- Resultant is a binary CN made of mega-variables and constraints between them.

How To Decompose CN Into Cluster Tree

- Conditions of the cluster tree have been presented.
- Algorithms to generate such cluster trees from CNs are still needed.
- The decomposition includes the following steps:
  1. Triangulate primal graph
  2. Identify clusters
  3. Organize clusters into tree

Triangulate Primal Graph

```
triangulatePG(G) {
  G' = G; fillin = {};
  for i=1 to n,
      select node x_i in G with adjacent node set adj(x_i);
      if nodes in adj(x_i) are not pairwise linked,
          add links in G to make adj(x_i) pairwise linked;
          record added links to fillin;
      remove x_i and its incident links from G;
  add links in fillin to G'; return G';
}
```


Identify Clusters

- Bookkeeping during triangulatePG(G)
  - ❑ Record $Q_i = adj(x_i) \cup \{x_i\}$ before removing x_i from G.
- After completion of triangulatePG(G)

```
identifyCluster({Q_1, Q_2 ,…, Q_n}) {
    Clus = {Q_1, Q_2 ,…, Q_n};
    for i=n to 1,
        if Q_i ⊂ Q_k(k<i), Clus = Clus \ {Q_i};
    return Clus;
}
```


Organize Clusters Into Tree

```
buildClusterTree(Clus) {
   init empty cluster tree T;
   remove Q from Clus and add Q to T;
   while Clus ≠ {},
      select Q∈Clus and Q' in T s.t. |Q∩Q'| is maximal;
      remove Q from Clus and add Q to T;
      connect Q and Q';
   return T;
}
```

Triangulation Revisited

- How should node x_i be selected in each iteration?
- Ex Map coloring problem
  - What are the clusters produced by triangulation in order (N,U,E,W,S,R)?
  - What is the consequence to solveCNByClusterTree()?
- Fewer fillins from triangulation are preferred.
- Finding triangulation with the minimum number of fillins is NP-hard.
- Heuristic: Select x_i of the minimum number of fillins

Remarks

- CSPs cover a broad class of practical problems.
- Can be solved by backtracking, iterative improvement, or constraint propagation.
- Solving CSPs is NP-hard in general.
- Many practical problems can be solved effectively with proper encodings, algorithms, and heuristics.
- More advanced topics
  - Encoding and algorithm selection for practical CSPs
  - Constraint optimization
  - Distributed CSPs