Review I

CMPSCI 383
December 6, 2011
General Information about the Final

• Closed book closed notes
• Includes midterm material too
• But expect more emphasis on later material
What you should know
Chapter 6: Constraint Satisfaction Problems

• Representations: atomic, factored, structured

• Definition of a constraint satisfaction problem:
  • In CSPs, states are defined by assignments of values to a set of *variables* $X_1...X_n$. Each variable $X_i$ has a *domain* $D_i$ of possible values.
  • States are evaluated based on their consistency with a set of *constraints* $C_1...C_m$ over the values of the variables.
  • A goal state is a complete assignment to all variables that satisfies all the constraints.
Example: Map coloring

- Variables — $WA, NT, Q, NSW, V, SA, T$
- Domains — $D_i = \{\text{red, green, blue}\}$
- Constraints — adjacent regions must have different colors.
  - E.g. $WA \neq NT$ (if the language allows this)
  - E.g. $((WA, NT), [(\text{red, green}), (\text{red, blue}), (\text{green, red}), \ldots])$
Example: Map coloring

- Solutions are **complete** and **consistent** assignments: every variable assigned, all assignments legal, e.g.:
  \[\{WA=\text{red}, NT=\text{green}, Q=\text{red}, NSW=\text{green}, V=\text{red}, SA=\text{blue}, T=\text{green}\}\]
Types of Constraints

- **Unary constraint**: concerns only a single value; e.g., $SA \neq \text{green}$
- **Binary constraint**: concerns the relative values of two variables
- **Global constraint**: concerns an arbitrary number of variables, e.g., $\text{Alldiff}$
Constraint graph
Local Consistency

• Node Consistency: $X_i$ is node-consistent if every value in the domain $D_i$ satisfies all of $X_i$’s unary constraints.
  • A network is node-consistent if every variable is node-consistent

• Arc Consistency: $X_i$ is arc-consistent with respect to $X_j$ if for every value in the domain $D_i$ there is some value in $D_j$ that satisfies the binary constraint on arc $(X_i,X_j)$
  • A network is arc-consistent if every variable is arc-consistent with every other variable
Arc Consistency (slightly different from the book)

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
    \((X_i, X_j) \leftarrow \text{REMOVE-FIRST}(queue)\)
    if \text{REMOVE-INCONSISTENT-VALUES}(X_i, X_j) then
        for each \(X_k\) in \text{NEIGHBORS}[X_i] do
            add \((X_k, X_i)\) to queue

function \text{REMOVE-INCONSISTENT-VALUES}(X_i, X_j) returns true iff succeeds
removed \leftarrow \text{false}
for each \(x\) in \text{DOMAIN}[X_i] do
    if no value \(y\) in \text{DOMAIN}[X_j] allows \((x,y)\) to satisfy the constraint \(X_i \leftrightarrow X_j\)
    then delete \(x\) from \text{DOMAIN}[X_i]; \(removed \leftarrow \text{true}\)
return removed
Naive Search Formulation

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
4. \( b = (n - k)d \) at depth \( k \), hence \( n! \cdot d^n \) leaves (\( d \) is domain size)
Backtracking Search

- Variable assignments are commutative, i.e.,
  \[ WA = \text{red} \text { then } NT = \text{green} \] same as \[ NT = \text{green} \text { then } WA = \text{red} \]

- 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
Simple backtracking search

- Depth-first search
- Choose values for one variable at a time
- Backtrack when a variable has no legal values left to assign.
- If search is uninformed, then general performance is relatively poor
Improving backtracking efficiency

• Approaches
  • Minimum remaining values heuristic (MRV)
    • Select the *most constrained* variable
      (the variable with the smallest number of remaining values)
  • Degree heuristic
    • Select the variable that is involved in the largest number of constraints with other unassigned variables: The most constraining variable.
  • Least-constraining value heuristic
    • Given a variable, choose the *least constraining value* — the value that leaves the maximum flexibility for subsequent variable assignments.
Combining Search with Inference

• Forward checking
  • Precomputing information needed by MRV
  • Early stopping

• Constraint propagation
  • Arc consistency (2-consistency)
Forward checking

- Can we detect inevitable failure early?
  - *And avoid it later?*
- Yes — track remaining legal values for unassigned variables
- Terminate search when any variable has no legal values.
Forward checking

- Assign \(\{WA=\text{red}\}\)
- Effects on other variables connected by constraints with WA
  - \(NT\) can no longer be red
  - \(SA\) can no longer be red
Forward checking

- Assign \(Q=\text{green}\)
- Effects on other variables connected by constraints with WA
  - \(NT\) can no longer be green
  - \(NSW\) can no longer be green
  - \(SA\) can no longer be green
Forward checking

- If $V$ is assigned *blue*
- Effects on other variables connected to WA
  - *SA is empty*
  - *NSW can no longer be blue*
- FC has detected a partial assignment that is *inconsistent* with the constraints.
Forward checking

- Solving CSPs with combination of heuristics plus forward checking is more efficient than either approach alone.
- FC checking propagates information from assigned to unassigned variables but does not provide detection for all failures.
  - NT and SA cannot be blue!
- Makes each current variable assignment arc consistent, but does not look far enough ahead to detect all inconsistencies (as AC-3 would)
Arc consistency

- $X \rightarrow Y$ is consistent iff
  for every value $x$ of $X$ there is some allowed $y$
- $SA \rightarrow NSW$ is consistent iff
  $SA=blue$ and $NSW=red$
Arc consistency

- $X \rightarrow Y$ is consistent iff for every value $x$ of $X$ there is some allowed $y$
- $NSW \rightarrow SA$ is consistent iff $NSW=\text{red}$ and $SA=\text{blue}$
  $NSW=\text{blue}$ and $SA=???$
  Arc can be made consistent by removing $\text{blue}$ from $NSW$
Arc consistency

- Arc can be made consistent by removing blue from NSW
- Recheck *neighbours*
  - Remove red from V
Arc consistency

- Arc can be made consistent by removing blue from NSW
- Recheck neighbours
  - Remove red from V
- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment.
  - Repeated until no inconsistency remains
Local search for csp

• Do we need the path to the solution or only the solution itself?
• Can we apply local search methods?
  • Hillclimbing
  • Simulated annealing
  • Genetic algorithms
• What’s a state?
Min-conflicts heuristic for local search

• To enable local search
  • 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., hill-climb 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; \(h(n) = 0\)
- **Evaluation**: \(h(n) = \text{number of attacks}\)

- Given random initial state, can solve \(n\)-queens in almost constant time for arbitrary \(n\) with high probability (e.g., \(n = 10,000,000\)). Average of 50 steps for \(n = 1\text{M}\). 
Exploiting the structure of CSPs

- Decompose into independent problems
- Tree-structured CSPs can be solved in linear time
- Reduce problems to tree-structured CSPs
  - Cycle cutset conditioning — Remove nodes to create trees
  - Tree decomposition — Decompose problem into a tree-structured set of subproblems
Cycle cutset conditioning

- Want to create a tree
  - What is a tree?
  - Why do we want to create one?
  - *Tree-structured CSPs solvable in linear time*
- Create a tree by deleting nodes
  - How can you delete nodes in CSPs?
  - *Set value and restrict domains*
- Does this always work well?
  - No, what can we do about that?
  - *Step through possible settings*
- What’s the payoff?
  - *Big efficiency gains*
Tree decomposition

- Again, want to create a tree
  - What’s another way of creating a tree?
  - Merging nodes

- Don’t need to memorize the following:
  - Rules for doing this:
    - Every variable in ≥1 subproblems
    - All connected variable pairs, and assoc. constraints, in ≥1 subproblems
    - If a variable appears in 2 subproblems, it must appear in all subproblems on the path connecting the two subproblems

- Now, how can we solve this new problem?
Things you should know about…

- Basic form of a CSP
- Be able to formulate a problem as a CSP
- Types of constraints
- Consistent assignment
- Complete assignment
- Constraint graph
- Constraint propagation
- Backtracking search for CSPs
- Heuristics to improve backtracking search
  - MRV
  - Degree heuristic
  - Least-constraining value
- Interleaving search and inference
  - Forward checking
  - Arc consistency
- Local search
  - Complete state formulation
  - Min-conflicts heuristic
- Using problem structure
  - Decomposing into independent subproblems
  - Turn into a tree structured problem
    - Basic knowledge of:
      - Cutset conditioning
      - Tree decomposition
What you don’t need to know for the exam

- Continuous domains
- Bounds propagation
- Bounds consistent
- MAC
- Details of Intelligent backtracking
- Constraint weighting
- Directed arc consistency

- Value symmetry
- Symmetry-breaking constraint
- Details of cycle cutset conditioning
- Details of tree decomposition
Chapter 13: Quantifying Uncertainty

- Sources of uncertainty
- MEU principle
- Basics of probability theory
  - Sample point/atomic event/possible world
  - Random variables
  - Joint and conditional distributions
  - Independence: absolute and conditional
  - Bayes Rule
Conditional probability

- **Definition of conditional probability:**
  \[ P(a \mid b) = \frac{P(a \land b)}{P(b)} \text{ if } P(b) > 0 \]

- **Product rule** gives an alternative formulation:
  \[ P(a \land b) = P(a \mid b) P(b) = P(b \mid a) P(a) \]

- **Chain rule** is derived by successive application of product rule:
  \[
P(X_1, \ldots, X_n) = P(X_1, \ldots, X_{n-1}) \cdot P(X_n \mid X_1, \ldots, X_{n-1})
  = P(X_1, \ldots, X_{n-2}) \cdot P(X_{n-1} \mid X_1, \ldots, X_{n-2}) \cdot P(X_n \mid X_1, \ldots, X_{n-1})
  = \ldots
  = \prod_{i=1}^{n} P(X_i \mid X_1, \ldots, X_{i-1})
  \]
Independence

A and B independent (absolute, marginal) iff

\[ P(A \mid B) = P(A) \quad \text{or} \quad P(B \mid A) = P(B) \quad \text{or} \quad P(A, B) = P(A) \cdot P(B) \]

A and B conditionally independent given C iff

\[ P(A \mid B, C) = P(A \mid C) \quad \text{or} \quad P(B \mid A, C) = P(B \mid C) \]

or

\[ P(A, B \mid C) = P(A \mid C) \cdot P(B \mid C) \]
Bayes Rule

• Product rule $P(a \land b) = P(a \mid b) \cdot P(b) = P(b \mid a) \cdot P(a)$

\[ \Rightarrow \text{Bayes' rule: } P(a \mid b) = \frac{P(b \mid a) \cdot P(a)}{P(b)} \]

• or in distribution form
  \[ P(Y \mid X) = \frac{P(X \mid Y) \cdot P(Y)}{P(X)} = \alpha P(X \mid Y) \cdot P(Y) \]

• Useful for assessing diagnostic probability from causal probability:
  • $P(\text{Cause} \mid \text{Effect}) = P(\text{Effect} \mid \text{Cause}) \cdot P(\text{Cause}) / P(\text{Effect})$
  • E.g., let $M$ be meningitis, $S$ be stiff neck:
    \[ P(\text{mls}) = P(s \mid m) \cdot P(m) / P(s) = 0.8 \times 0.0001 / 0.1 = 0.0008 \]
  • Note: posterior probability of meningitis still very small!
Specifically (but not limited to) ----

- Be able to compute various probabilities, and probability distributions, from the full joint dist. (cf. problem 13.8 p. 507)
- Applying Bayes rule
  - E.g., diagnosis: \( p(\text{disease} \mid \text{symptoms}) \) cf. problem 13.15 p. 508
- Equivalent statements for conditional independence: cf. problem 13.17, p. 508
What you don’t need to know for the exam

• Wumpus world
Chapter 14: Probabilistic Reasoning

- Bayesian networks
- Exact Inference in Bayesian networks
- Approximate Inference in Bayesian networks
Bayesian networks

- A simple, graphical notation for conditional independence assertions and hence for compact specification of joint distributions
- Syntax:
  - a set of nodes, one per variable
  - a directed, acyclic graph (link \(\approx\) "directly influences")
  - a conditional distribution for each node given its parents: \(P(X_i | \text{Parents}(X_i))\)
- In the simplest case, conditional distribution represented as a \textbf{conditional probability table} (CPT) giving the distribution over \(X_i\) for each combination of parent values
Example

- Topology of network encodes conditional independence assertions:

  - *Weather* is independent of the other variables
  - *Toothache* and *Catch* are conditionally independent given *Cavity*
Example: Home security

- **Burglary**
  - $P(B) = 0.001$

- **Earthquake**
  - $P(E) = 0.002$

- **Alarm**
  - $P(A|B,E)$:
    - T T: 0.95
    - T F: 0.94
    - F T: 0.29
    - F F: 0.001

- **JohnCalls**
  - $P(J|A)$:
    - T: 0.90
    - F: 0.05

- **MaryCalls**
  - $P(M|A)$:
    - T: 0.70
    - F: 0.01
Benefits: Compactness

• A CPT for Boolean $X_i$ with $k$ Boolean parents has $2^k$ rows for the combinations of parent values.
• Each row requires one number $p$ for $X_i = true$ (the number for $X_i = false$ is just $1-p$).
• If each variable has no more than $k$ parents, the complete network requires $O(n \cdot 2^k)$ numbers.
• i.e., grows linearly with $n$, vs. $O(2^n)$ for the full joint distribution.
• For burglary net, $1 + 1 + 4 + 2 + 2 = 10$ numbers (vs. $2^5 - 1 = 31$).
Semantics

• The full joint distribution is defined as the product of the local conditional distributions:

\[ P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | \text{Parents}(X_i)) \]

• Example

\[ P(j \land m \land a \land \neg b \land \neg e) = P(j \mid a) P(m \mid a) P(a \mid \neg b, \neg e) P(\neg b) P(\neg e) \]
Conditional Independence

Node X is conditionally independent of its non-descendants given its parents.
Conditional Independence

Node X is conditionally independent of all other nodes in the network given its “Markov blanket” (its parents, children, and their parents).
Conditional independence

- Are \textit{JohnCalls} and \textit{MaryCalls} independent?
  - No, they are not completely independent
- If the value of \textit{Alarm} is known, are \textit{JohnCalls} and \textit{MaryCalls} independent?
  - Yes, for each known value of A, J and M are independent
Conditional independence

• Are Burglary and Earthquake cond. independent?
  • Yes, nodes are conditionally independent of their non-descendents given their parents

• Are they completely independent?
  • No, one can ‘explain away’ the other if Alarm is known.
Constructing Bayesian networks

1. Choose an ordering of variables $X_1, \ldots, X_n$
2. For $i = 1$ to $n$
   - add $X_i$ to the network
   - select parents from $X_1, \ldots, X_{i-1}$ such that
     $$P(X_i \mid \text{Parents}(X_i)) = P(X_i \mid X_1, \ldots, X_{i-1})$$

This choice of parents guarantees:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{\rho} P(X_i \mid X_1, \ldots, X_{i-1}) \quad \text{(chain rule)}$$
$$= \prod_{i=1}^{\rho} P(X_i \mid \text{Parents}(X_i)) \quad \text{(by constr.)}$$
Example

- Suppose we choose the ordering M, J, A, B, E
Summary

- **Bayesian network:**
  - Directed acyclic graph whose nodes correspond to r.v.s; each note has a conditional distribution for its values given its parents.
  - Provides a concise way to represent conditional independence relations
  - Specifies the full joint distribution
  - Often exponentially smaller than explicit representation of the joint distribution
What you **don’t** need to know for the exam

- Noisy-OR, noisy-MAX, leak node
- Bayes nets with continuous variables
Inference in Bayesian networks

• Exact
  • Inference with joint probability distributions
  • Exact inference in Bayesian networks
  • Inference by enumeration
  • Complexity of exact inference

• Approximate
  • Inference by stochastic simulation
  • Simple sampling
  • Rejection sampling
  • Markov chain Monte Carlo (MCMC)
Inference terminology

- **Conditional probability table:** data structure that lists probabilities of a variable given one or more other variables.

- **Joint distribution:**
  distribution that is specified by a Bayesian network

- **Inference:** produces the probability distribution of one or more variables given one or more other variables.
Types of nodes in inference

- Burglary
- Earthquake
- Alarm
- JohnCalls
- MaryCalls

Evidence (or “observed”) variables
Types of nodes in inference

Query variables
Types of nodes in inference

Hidden variables
Simple inferences

Burglary

Earthquake

Alarm

JohnCalls

MaryCalls
Simple inferences

- Burglary
- Earthquake
- Alarm
- JohnCalls
- MaryCalls
Simple inferences

- Burglary
- Earthquake
- JohnCalls
- MaryCalls
More difficult inferences

Burglary

Earthquake

Alarm

JohnCalls

MaryCalls
Inference by enumeration

\[ P(B|j,m) = <0.284, 0.716> \]
Why approximate inference?

- Inference in singly connected networks is linear!
- ...but many networks are not singly connected
- Inference in multiply connected networks is exponential, even when the number of parents/node is bounded
- May be willing to trade some small error for more tractable inference
Stochastic simulation

• Core idea
  • Draw samples from a sampling distribution defined by the network
  • Compute an approximate posterior probability in a way that converges to the true probability

• Methods
  • Simple sampling from an empty network
  • Rejection sampling — reject samples that don’t agree with the evidence
  • Likelihood weighting — weight samples based on evidence
  • Markov chain Monte Carlo — sample from a stochastic process whose stationary distribution is the true posterior
Simple sampling

• Given an empty network...
• And beginning with nodes without parents...
• We can sample from conditional distributions and instantiate all nodes.
• This will produce one element of the joint distribution.
• Doing this many times will produce an empirical distribution that approximates the full joint distribution.
Example
Benefits and problems of simple sampling

• Works well for an empty network
  • Simple
  • In the limit (many samples), the estimated distribution approaches the true posterior
• But in nearly all cases, we have evidence, rather than an empty network
• What can we do?
• Throw out cases that don’t match the evidence
Rejection sampling

- Sample the network as before...
- But discard samples that don’t correspond with the evidence.
- Similar to real-world estimation procedures, but the network is the stand-in for the world (much cheaper and easier).
- However, hopelessly expensive for large networks where $P(e)$ is small.
Likelihood weighting

• Do simple sampling as before...
• But weight the likelihood of each sample based on the evidence
• Don’t need to know details…
MCMC: Markov Chain Monte Carlo

• The “state” of the system is the current assignment of all variables

• Algorithm
  • Initialize all variables randomly
  • Generate next state by sampling one variable given its Markov blanket
  • Sample each variable in turn, keeping other evidence fixed.

• Variable selection can be sequential or random
MCMC Problems

• Difficult to tell if it has converged
• Multiple parameters (e.g., burn-in period)
• Can be wasteful if the Markov blanket is large because probabilities don’t change much
Specifically (but not limited to)…

• Be able to write down probabilistic statements asserted by a Bayes net.
• Be able to draw a Bayes net from a verbal description: cf. problem 14.1, p. 558
• Be able to compute prob of query given evidence and a Bayes net
What you don’t need to know for the exam

• Variable elimination algorithm (14.4.2)
• Clustering algorithms (14.4.4)
• Likelihood weighting
• Why Gibbs sampling works
• Secs. 14.6, 14.7
Problem 14.14

Which of the following are asserted by the network?

\[ P(B,I,M) = P(B)P(I)P(M) \]

\[ \rightarrow P(J|G) = P(J|G,I) \]

\[ \rightarrow P(M|G,B,I) = P(M|G,B,I,J) \]

Figure 14.23  FILES: figures/politics.eps (Tue Nov 3 16:23:20 2009). A simple Bayes net with Boolean variables \( B = \text{BrokeElectionLaw}, I = \text{Indicted}, M = \text{PoliticallyMotivatedProsecutor}, G = \text{FoundGuilty}, J = \text{Jailed}. \)

Which of the following are asserted by the network?

\[ \times P(B,I,M) = P(B)P(I)P(M) \]

\[ \rightarrow P(J|G) = P(J|G,I) \]

\[ \rightarrow P(M|G,B,I) = P(M|G,B,I,J) \]
Problem 14.14 contd.

Figure 14.23  FILES: figures/politics.eps (Tue Nov 3 16:23:20 2009). A simple Bayes net with Boolean variables \( B = \text{Broke Election Law}, \ I = \text{Indicted}, \ M = \text{Politically Motivated Prosecutor}, \ G = \text{Found Guilty}, \ J = \text{Jailed}. \)

Calculate \( P(b, i, \neg m, g, j) \)

\[
P(b, i, \neg m, g, j) = P(b)P(\neg m)P(i|b, \neg m)P(g|b, i, \neg m)P(j|g) \\
= .9 \times .9 \times .5 \times .8 \times .9 = .2916
\]
Problem 14.14 contd.

What is the prob that someone goes to jail given that they broke the law, have been indicted, and face a politically motivated prosecutor?
Since $B, I, M$ are fixed true in the evidence, we can treat $G$ as having a prior of 0.9 and just look at the submodel with $G, J$:

$$P(J|b, i, m) = \alpha \sum_g P(J, g) = \alpha[P(J, g) + P(J, \neg g)]$$
$$= \alpha[\langle P(j, g), P(\neg j, g) \rangle + \langle P(j, \neg g), P(\neg j, \neg g) \rangle]$$
$$= \alpha[.81, .09] + \langle 0, 0, 1 \rangle = \langle .81, .19 \rangle$$

That is, the probability of going to jail is 0.81.
Problem 14.4 contd.

The long way:

\[ P(J = t | B = t, I = t, M = t) = P(j | b, i, m) = P(j, b, i, m) / P(b, i, m) = \]
\[ \alpha \sum_g P(j, g, b, i, m) = \alpha \sum_g P(b)P(m)P(i | b, m)P(g | b, i, m)P(j | g) = \]
\[ \alpha \sum_g 1 \times 1 \times 1 \times P(g | b, i, m)P(j | g) = \]
\[ \alpha \left( P(g | b, i, m)P(j | g) + P(\neg g | b, i, m)P(j | \neg g) \right) = \]
\[ \alpha (0.9 \times 0.9 + 0.1 \times 0) = \alpha \times 0.81 \]