Material


• Homework: Due Monday, May 2\textsuperscript{nd}.
  – Do: problems 5.1, 5.3, 5.5, 5.8, 5.9, 5.11

• Reminder: Midterm next Monday, May 2\textsuperscript{nd} (will cover up to and including Chapter 5, and Bacchus tutorial)
Outline For Today

• Bacchus tutorial
  – note: some of this material slightly overlaps with chapter 5, but using more precise notation.
CSP

- Set of variables: \( \{ V_1, \ldots, V_n \} \)
- Set of domains for each \( V_i \), \( \text{Dom}[V_i] \)
- Set of constraints \( \{ C_1, \ldots, C_m \} \)
- Each constraint over a set of variables gives set of satisfying values of that set of vars.
  - E.g.: \( \text{VarsOf}[C_i] = \{ V_2, V_4, V_7 \} \)
  - Constraint might be a table of possible values:
CSP

• Goal is to:
  – 1) find one set of variable values that satisfy all constraints
  – 2) find all variable values that satisfy all constraints.
• This can be NP-hard in general.
  – Ex: 3-SAT (or 3-CNF)
    • (x_1 \lor \neg x_1 \lor \neg x_2) \land (x_3 \lor x_2 \lor x_4) \land (\neg x_1 \lor \neg x_3 \lor \neg x_4)
    • There is no polynomial-time algorithm to find a satisfying assignment to all variables (in a general 3-CNF formula) unless P=NP
CSP

• We’ve talked about consistency
  – node-consistency
  – arc-consistency
    • AC-3, and AC-4
  – Path consistency
  – (i,j) consistency
    • A CSP is (i,j)-consistent if whenever we choose consistent values for any set of i variables, we can find values for any set of j variables such that the values of all i+j variables are consistent.
  – Strongly k-consistent
    • if it is (j,1) consistent, for j=1 to k.
  – (1,j)-consistent, called i inverse consistency.
    • obtained by pruning domain values from variables.
Backtracking

• Generic Backtracking:
  – do a tree search, as soon as a variable becomes inconsistent, then back out to previous variable going to its next value.
  – Quite suboptimal

• Constraint propagation
  – enforce consistency in sub-CSP below a given tree-node

• Intelligent Backtracking
  – keep track of why things fail, and jump back to those reasons (rather than blindly to previous variable)
  – Dynamic variable order heuristics can improve search considerably.
No-Goods

• A “no-good”: set of assignments guaranteed not to be contained in any solution.

• All of backtracking and other forms of search can be seen as “no-good processing”, essentially discovering no-goods, and unioning them together to get larger no-goods.
  – larger no-goods are better since they cover more
  – We re-use the no-goods at other sections of the search
  – We can’t store all no-goods since there are an exponential number of them.
  – All of these are heuristic algorithms (none solve the problem in polynomial time of course).
Propositional Encoding

• Propositional Logic.
  – a proposition “p” is stated, meaning it is asserted to be true. We can encode CSP using propositional logic.
  – Primitive constraints:
    • $C$ is a constraint, if $a \in C$, then $a$ is an assignment such as:

      $$a = \{ V_1 \leftarrow x_1^0, \ldots, V_k \leftarrow x_k^0 \}$$

    • We can represent all constraints in $C$ as disjunction of conjunctions:

      $$\vee (V_1 \leftarrow x_1^0) \land \cdots \land (V_k \leftarrow x_k^0)$$
      $$\vee (V_1 \leftarrow x_1^1) \land \cdots \land (V_k \leftarrow x_k^1)$$
      $$\vdots$$
      $$\vee (V_1 \leftarrow x_1^\ell) \land \cdots \land (V_k \leftarrow x_k^\ell)$$
Propositional Encoding

– Exclusivity: each variable assigned only a single value, and we can represent this also:

\[ V \leftarrow x_1 \Rightarrow (V \not\leftarrow x_2 \land \ldots \land V \not\leftarrow x_k) \]

\[ V \leftarrow x_2 \Rightarrow (V \not\leftarrow x_1 \land V \not\leftarrow x_3 \land \ldots \land V \not\leftarrow x_k) \]

\[ \cdots \]

\[ V \leftarrow x_k \Rightarrow (V \not\leftarrow x_1 \land \ldots \land V \not\leftarrow x_{k-1}) \]
Propositional Encoding

– Exhaustiveness: Each variable must be assigned a value.

\[ V \leftarrow x_1 \lor V \leftarrow x_2 \lor \ldots \lor V \leftarrow x_k \]

– All three combined, we get:

\[
(C \land X \land E) \Rightarrow (V_1 \leftarrow x_1^0 \land \cdots \land V_n \leftarrow x_n^0) \land \ldots \land (V_1 \leftarrow x_1^\ell \land \cdots \land V_n \leftarrow x_n^\ell)
\]
No-good reasoning

• Resolution:

\[(p \lor q) \land (\neg p \lor r) \Rightarrow (q \lor r)\]

• We can use resolution to union together no-goods.

<table>
<thead>
<tr>
<th>No Goods</th>
<th>Equivalent Clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ( V \leftarrow a, V_1 \leftarrow x_1 )</td>
<td>1. ( V \nLeftarrow a \lor V_1 \nLeftarrow x_1 )</td>
</tr>
<tr>
<td>2. ( V \leftarrow b, V_2 \leftarrow x_2, V_4 \leftarrow x_4 )</td>
<td>2. ( V \nLeftarrow b \lor V_2 \nLeftarrow x_2 \lor V_4 \nLeftarrow x_4 )</td>
</tr>
<tr>
<td>3. ( V \leftarrow c, V_3 \leftarrow x_3 )</td>
<td>3. ( V \nLeftarrow c \lor V_3 \nLeftarrow x_3 )</td>
</tr>
<tr>
<td>4. ( V \leftarrow d, V_3 \leftarrow x_3, V_4 \leftarrow x_4 )</td>
<td>4. ( V \nLeftarrow d \lor V_3 \nLeftarrow x_3 \lor V_4 \nLeftarrow x_4 )</td>
</tr>
</tbody>
</table>
No-good reasoning

• Resolution:
  \[(p \lor q) \land (\neg p \lor r) \Rightarrow (q \lor r)\]

• We can use resolution to union together no-goods.

No Goods

1. \(\{V \leftarrow a, V_1 \leftarrow x_1\}\)
2. \(\{V \leftarrow b, V_2 \leftarrow x_2, V_4 \leftarrow x_4\}\)
3. \(\{V \leftarrow c, V_3 \leftarrow x_3\}\)
4. \(\{V \leftarrow d, V_3 \leftarrow x_3, V_4 \leftarrow x_4\}\)

Exhaustiveness

5. \(V \leftarrow a \lor V \leftarrow b \lor V \leftarrow c \lor V \leftarrow d\)

Equivalent Clauses

1. \(V \not\models a \lor V_1 \not\models x_1\)
2. \(V \not\models b \lor V_2 \not\models x_2 \lor V_4 \not\models x_4\)
3. \(V \not\models c \lor V_3 \not\models x_3\)
4. \(V \not\models d \lor V_3 \not\models x_3 \lor V_4 \not\models x_4\)

New Unioned No-Good

\[\{V_1 \leftarrow x_1, V_2 \leftarrow x_2, V_3 \leftarrow x_3, V_4 \leftarrow x_4\}\].
No-good reasoning

- Other forms of no-goods can be combined together as well.
- Most forms of no-good processing is done more implicitly using various heuristics.
- Key is that there are too many no-goods to remember, even after unioning.
- We must strive for ways to remember what will be useful in future, and forget what we won’t ever need again, but never remember too much so that memory requirements becomes exponential.
No-good reasoning

- No-goods are encountered during search
  - depth-first search (standard for CSP)
  - When values found inconsistent, they become nogood.
  - Logic can explain what we do:
    - if at tree-node n, all values of a variable V fail, we can use resolution above to create a new no-good not containing V, but containing all other reasons variable has failed.
- No-goods are stored as a set of levels
  - i.e., a tree-node n, a set of variables have been assigned. To store the no-good, we store the current set of levels (implicitly storing the values of those variables). We also store the current variable’s assignment. E.g.,
    - (1,3,5,9,V← a) is a no good that says the current assignments to variables at levels 1,3,5, and 9 + assigning V to a is a no good.
No-good reasoning

• What are no-goods good for?
  – For Intelligent Backtracking
    • logic can justify intelligent backtracking (e.g., to previous level or to much earlier in search tree)
  – For domain pruning
    • if we find a no-good \{1,2,4,V\leftarrow a\}, where V is a future variable, we can prune a from the domain of V at the current time (but we need to restore it when we return from level 4)
Generic Tree-Search Template

TreeSearch(level)
  if all variables are assigned
    current assignments is a solution, enumerate it
      if FINDALL
        set \{1, \ldots, level-2\} as a new no-good for assignment
        at level-1
        return (level-1)
      else
        return (0)
  else
    return (0)

V := pickNextVariable()
BTLevel := level
for x ∈ CurrentDom[V]
  assign(V, x, level)
  if checkConsistent(V, x, level)
    propagateConstraints(V, x, level)
    BTLevel := TreeSearch(level+1)
  undo(V, x, level)
  if BTLevel < level
    return (BTLevel)

BTlevel := computeBackTrackLevel(V, level)
setNoGoodOfAssignmentatBTLevel(BTLevel, V, level)
return (BTLevel)
Generic Tree-Search Template

- **FINDALL**: if we should find all solutions. Note that if we find a solution, it is not another solution so it becomes a no-good for the rest of the search.
- **pickNextVariable**: determines next variable to split on
  - this is important, can have large impact on search. “Fail First” principle later
  - In any case, picks variables with empty domains first to find deadends, and next picks variables with only one value in the domains
- **checkConsistent**: checks that newly assigned value is consistent
- **propagateConstraints**: propagate any constraints now available by the newly assigned variable (e.g., forward checking or arc consistency to future vars)
- **TreeSearch**: might want to backtrack below current level and if so, we backtrack back to that level right there, in inner loop
- **computeBacktrackLevel**: if all values of V failed, we might want to backtrack to level earlier than previous level
- **setNoGoodOfAssignmentAtBTLevel**: if any no-goods can be computed, they are done so here.
Generic Backtracking

\[
BT(\text{level})
\]
\[
\text{if all variables are assigned}
\]
\[
\text{current assignments is a solution, enumerate it}
\]
\[
\text{if FINDALL}
\]
\[
\text{return (level-1)}
\]
\[
\text{else}
\]
\[
\text{return (0)}
\]
\[
V := \text{pickNextVariable()}
\]
\[
\text{BTLevel := level}
\]
\[
\text{for } x \in \text{Dom}[V]
\]
\[
\text{assign}(V,x,\text{level})
\]
\[
\text{if checkConsistent}(V,x,\text{level})
\]
\[
\text{BTLevel := BT(level+1)}
\]
\[
\text{undo}(V,x,\text{level})
\]
\[
\text{if BTLevel < level } //\text{Only occurs when BTLevel == 0}
\]
\[
\text{return (BTLevel)}
\]
\[
\text{return (level-1)}
\]
Generic Backjumping

\textbf{BJ}(level)
\begin{align*}
&\text{if all variables are assigned} \\
&\quad \text{current assignments is a solution, enumerate it} \\
&\quad \text{if FINDALL} \\
&\quad \quad \text{maxBJLevel[level-1] = level-2} \\
&\quad \quad \text{return level-1} \\
&\quad \text{else} \\
&\quad \quad \text{return 0} \\
\end{align*}
\begin{align*}
V &:= \text{pickNextVariable}() \\
\text{BTLevel} &:= \text{level} \\
\text{maxBJLevel[level]} &:= 0 \\
\text{for } x \in \text{Dom}[V] \\
&\text{assign}(V,x,\text{level}) \\
&\quad \text{if } (M := \text{checkConsistentBJ}(V,x,\text{level})) == \text{level} \\
&\quad \quad \text{BTLevel} := BJ(\text{level}+1) \\
&\quad \text{else} \\
&\quad \quad \text{maxBJLevel[level]} = \max(\text{maxBJLevel[level]}, M) \\
&\quad \quad \text{undo}(V,x,\text{level}) \\
&\quad \quad \text{if BTLevel < level} \\
&\quad \quad \text{return BTLevel} \\
\text{BTLevel} &:= \text{maxBJLevel[level]} \\
\text{maxBJLevel[BTLevel]} &:= \max(\text{maxBJLevel[BTLevel]}, \text{BTLevel}-1) \\
&\quad \quad // \text{Note that this max is equal to BTLevel-1.} \\
\text{return BTLevel} \\
\end{align*}

\textbf{checkConsistencyBJ}(V,x,\text{level})
\begin{align*}
&\text{for } i := 1 \text{ to } \text{level-1} \\
&\quad \text{forall } C \text{ such that} \\
&\quad \quad 1. V \in \text{VarsOf}[C] \\
&\quad \quad 2. \text{All other variables of } C \text{ are assigned at level } i \text{ or above} \\
&\quad \quad \quad \text{if } !\text{checkConstraint}(C) //\text{Check against current assignment.} \\
&\quad \quad \text{return } i \\
&\text{return } (\text{level})
\end{align*}
Conflict-Directed Backjumping

\[ \text{CBJ}(\text{level}) \]
\[
\text{if all variables are assigned} \\
\quad \text{current assignments is a solution, enumerate it} \\
\quad \text{if FINDALL} \\
\quad \quad \text{NoGoodSet}[\text{level}-1] := \{1, \ldots, \text{level}-2\} \\
\quad \quad \text{return level-1} \\
\quad \text{else} \\
\quad \quad \text{return (0)}
\]

\[ V := \text{pickNextVariable}() \]
\[ \text{BTLevel} := \text{level} \]
\[ \text{NoGoodSet}[\text{level}] := \emptyset \]
\[ \text{for } x \in \text{Dom}[V] \]
\[ \quad \text{assign}(V,x,\text{level}) \]
\[ \quad \text{if} (\text{NoGood} := \text{checkConsistentCBJ}(V,x,\text{level})) = \emptyset \]
\[ \quad \quad \text{BTLevel} := \text{CBJ}(\text{level}+1) \]
\[ \text{else} \]
\[ \quad \quad \text{NoGoodSet}[\text{level}] = \text{NoGoodSet}[\text{level}] \cup (\text{NoGood} - \{\text{level}\}) \]
\[ \quad \quad \text{undo}(V,x,\text{level}) \]
\[ \quad \text{if BTLevel} < \text{level} \]
\[ \quad \quad \text{return} \text{(BTLevel)} \]

\[ \text{BTLevel} := \max(\text{NoGoodSet}[\text{level}]) \]
\[ \text{NoGoodSet}[\text{BTLevel}] := \text{NoGoodSet}[\text{BTLevel}] \cup (\text{NoGoodSet}[\text{level}]-\{\text{BTLevel}\}) \]
\[ \text{return} \text{(BTLevel)} \]

\[ \text{checkConsistencyCBJ}(V,x,\text{level}) \]
\[ \text{for } i := 1 \text{ to level}-1 \]
\[ \text{forall } C \text{ such that} \]
\[ \quad 1. \ V \in \text{VarsOf}[C] \]
\[ \quad 2. \ \text{All other variables of } C \text{ are assigned at level } i \text{ or above} \]
\[ \quad \quad \text{if} \ \neg \text{checkConstraint}(C) \quad // \text{Check against current assignment}. \]
\[ \quad \quad \text{return} (\text{Levels VarsOf}[C] \text{ were assigned}) \]
\[ \quad \text{return} (\emptyset) \]
Value Specific CBJ

$$\text{vSCBJ}(\text{level})$$

if all variables are assigned
current assignments is a solution, enumerate it
if FINDALL
NoGoodSet[VarAt[level-1],ValAt[level-1]] := \{1,\ldots,\text{level-2}\}
return level-1
else
return (0)

V := pickNextVariable()
BTLevel := level
for x \in Dom[V]
assign (V,x,level)
if (NoGood := checkConsistentCBJ(V,x,level)) == \emptyset
BTLevel := vSCBJ(level+1)
else
NoGoodSet[V,x] = NoGood - \{level\}
undo(V,x,level)
if BTLevel < level
return (BTLevel)

BTLevel := max_{x \in Dom[V]} (NoGoodSet[V,x])
C := Constraint between VarAt[BTLevel] and V
//C is the universal constraint if there is no constraint
//C, such that VarsOf[C] = \{VarAT[BTLevel],V\},
//between these two variables exists.

NoGoodSet[VarAt[BTLevel],ValAt[BTLevel]] :=
\bigcup_{x \in Dom[V]} \text{and } C(ValAt[BTLevel],x)\text{NoGoodSet}[V,x] - \{BTLevel\}
return (BTLevel)
More advanced options

• Domain pruning
  – keep a current domain of values for each variable, prune out values we know are impossible, restore them when they become possible again.

• Constraint propagation
  – Use forward checking or AC-3 to for each sub-CSP prune from the domains of variables values that we know are not possible.
  – Other more advanced algorithms are:
    • MAC: Maintain Arc Consistency: It does (1,1)-consistency in each sub-CSP. Can be combined with CBJ.
    • GAC: generalized arc consistency (rather than waiting for constraint that has all but one value instantiated (FC) or two values instantiated (MAC), we can do more).
Dynamic Variable Orderings

• The `pickNextVar` routine need not always choose the next variable in all circumstances.

• In the most general case, we choose both the next variable and a value of the next variable at the same time, and each time we choose this, it can be different and might depend on anything that has previously happened during the search.

• This is shown on algorithm in next slide:
General Dynamic Variable order

FlexTreeSearch(level)
   if all variables are assigned
      current assignments is a solution, enumerate it
      if FINDALL
         \{1,...,level-2\} is a new no-good for assignment
         at level-1
         prune(VarAt[level-1], ValAt[level-1], level-2)
         return(level-1)
   else
      return(0)

NEXT:
   (V,x) := pickNextVarVal()
   // V is unassigned, and x is from CurrentDom[V].
   // Furthermore,
   // 1. If there is a variable with an empty CurrentDom
   // return such a variable and NIL as value x.
   // 2. Otherwise if there is a variable with a singleton
   // CurrentDom, then return such a variable with x
   // set to its single remaining value.

   if(x != NIL)
      assign(V,x,level)
      BTLevel := level
      if checkConsistent(V,x,level)
         propagateConstraints(V,x,level)
         BTLevel := FlexTreeSearch(level+1)
      undo(V,x,level)
      if BTLevel < level
         return(BTLevel)
      goto NEXT
   // We drop out here only if we have a variable with an empty domain.
   NoGood := ComputeNoGoodFromExhaustedVariable(V)
   BTLevel := max(NoGood)
   setNoGoodOfAssignmentatBTLevel(BTLevel, NoGood-{BTLevel})
   prune(VarAT[BTLevel], ValAT[BTLevel], max(NoGood-{BTLevel}))
   return(BTLevel)
Dynamic Variable order

• Current Practice
  – In graph coloring, a heuristic that works well is the “Brelaz heuristic”
    • first color vertex of maximum degree.
    • Next, select uncolored vertex of maximum saturation (maximum number of different colored vertices connected to it).
      – break ties by the degree of the vertex intersected with the uncolored nodes.
  – Maximizing number of different colored vertices puts most constraints on new variable choosen.
Dynamic Variable order

• Brelaz heuristic is essentially same as:
  – choose next variable with minimum remaining consistent values
  – break ties by number of constraints variable has with set of currently unassigned variables.

• This is example of the “fail first” principle
  – the best search order is one that minimizes the expected length (depth) of each branch (i.e., prefer short trees since search is exponential in the depth).
Other good sources