Solving Constraint Programs using Conflicts and Backjumping

Slides draw upon material from: Prof. Patrick Prosser, Glasow University

Brian C. Williams
16.410-13
September 29th, 2010

Search Performance on N Queens

- Standard Search
- Backtracking
- BT with Forward Checking
- Dynamic Variable Ordering
- Iterative Repair
- Conflict-directed Back Jumping

- A handful of queens
- About 15 queens
- About 30 queens
- About 1,000 queens
- About 10,000,000 queens (except truly hard problems)
Back Jumping

**Backtracking** At dead end, backup to the most recent variable.

**Backjumping** At dead end, backup to the most recent variable that eliminated some value in the domain of the dead end variable.

---

**Example of a CSP**

Slide progression due to Prosser [4C presentation, 2003]
Example of a CSP

Variables and Instantiation Order

Checking back

Find solution using Backtracking

Example of a CSP

Variables and Instantiation Order

Checking back

1 = red
2 = blue
3 = green
Example of a CSP

| A | B | C | D | E | F | G | H |

1 = red
2 = blue
3 = green

Example of a CSP

| A | B | C | D | E | F | G | H |

1 = red
2 = blue
3 = green

Variables and Instantiation Order

Checking back

V_a
V_b
V_c
V_d
V_e
V_f
V_g
V_h
Example of a CSP

Variables and Instantiation Order

Checking back

Fast forward
Example of a CSP

Variables and Instantiation Order

Checking back

Example of a CSP

Variables and Instantiation Order

Checking back

1 = red
2 = blue
3 = green
Example of a CSP

Variables and Instantiation Order

Checking back

Example of a CSP

Variables and Instantiation Order

Checking back

1 = red
2 = blue
3 = green
Example of a CSP

Variables and Instantiation Order

Checking back

Why did it backtrack to E?

1 = red
2 = blue
3 = green

That was dumb!
What would have happened if we had the E\* intermediate variables? i.e. it falls back on E4, then E3, towards E?

Example of a CSP

Variables and Instantiation Order

Checking back

1 = red
2 = blue
3 = green

Why did it backtrack to E? That was dumb!

Whats better and why?
Why backtrack to D?
D ⇒ not F

Why backtrack to D?
D ⇒ not F
B ⇒ not F
Example of a CSP

Variables and Instantiation Order

Checking back

Why backtrack to D?

D ➔ not F
B ➔ not F
C ➔ not F
F or F or F

not D or not B or not C

Its safe to remove the deepest assignment.

Moving Forward:

confSet[i] denotes past variables that conflict with values in the domain of v[i]

past variable v[h]

conflict with v[h]

current variable v[i]

Move down like Backtrack Search:

- Instantiate v[i] := x, for next x in D[i]
- Check constraint (v[i], v[h]), if fails
  - say “v[i] is in conflict with v[h]”
  - add h to the set confSet[i]
Moving Forward

Variables and Instantiation Order

Checking back

ConfSet[F] = {D, B, G, H}

Moving Forward

Variables and Instantiation Order

Checking back

ConfSet[F] = {D, B, G, H}

D → not F
B → not F
Moving Forward

Variables and Instantiation Order

Checking back

ConfSet[F] = {D, B, C}

Moving Forward

Backtrack when v[i] domain exhausted:
- Jump to deepest var h in ConfSet[i].

Image by MIT OpenCourseWare.
Backtracking: Conflict-directed Backjumping

1. Backtrack when \( v[i] \) domain exhausted:
   - Jump to deepest var \( h \) in ConfSet\([i]\).
   - Update ConfSet\([h]\).

2. Add \{1, 0\} to confset[4]

3. Backtrack when \( v[i] \) domain exhausted:
   - Jump to deepest var \( h \) in ConfSet\([i]\).
   - Update ConfSet\([h]\] with ConfSet\([i]\) / \( h \).
Backjumping: Conflict-directed Back Jumping

1. Backtrack when v[i] domain exhausted:
   - Jump to deepest var h in ConfSet[i].
   - Update ConfSet[h] with ConfSet[i] / h.
   - Reset Domains, ConfSets below h.
   - Move forward: try next h assignments.

2. When jumping back from v[i] to v[h],
   1. Update conflict sets:
      ```
      confSet[h] := confSet[h] \ confSet[i] \ {h}
      confSet[i] := {0}
      ```
      - This means:
        if we later jump back from v[h],
        jump back to a variable that is
        in conflict with v[h] or with v[i].
   2. Throw away everything CBJ knows about v[i].
   3. Reset all variables from v[h+1] to v[i] (i.e. domain and confSet).
Variables and Instantiation Order

Checking back

D \rightarrow \text{not } F
B \rightarrow \text{not } F
C \rightarrow \text{not } F

ConfSet[F] = \{D, B, C\}

Backing Up: Conflict-directed Back Jumping

What do we do if variable 4’s domain is exhausted as well?

STOP
Backtrack when v[h] domain exhausted:
- Jump to deepest var g in ConfSet[h].
- Update ConfSet[g] with ConfSet[h] / g.
- Reset Domains, ConfSets below g.
- Move forward: try next g assignments.
CBJ Supports Successive Jumps

If there are no values remaining for \( v[i] \)
Jump back to \( v[h] \), the deepest variable in conflict with \( v[i] \).
The hope: re-instantiating \( v[h] \) will allow us to find a good value for \( v[i] \)

If there are no values remaining for \( v[h] \)
Jump back to \( v[g] \), the deepest variable in conflict with \( v[i] \) or \( v[h] \).
The hope: re-instantiating \( v[g] \) will allow us to find a good value for \( v[i] \) or a good value for \( v[h] \) that will be good for \( v[i] \)

If there are no values remaining for \( v[g] \)
Jump back to \( v[f] \), the deepest variable in conflict with \( v[i] \) or \( v[h] \) or \( v[g] \).
The hope: re-instantiating \( v[f] \) will allow us to find a good value for \( v[i] \) or a good value for \( v[h] \) that will be good for \( v[i] \) or a good value for \( v[g] \) that will be good for \( v[h] \) and \( v[i] \)

CBJ: Prosser’s Original Formulation

```plaintext
consistent := false
confSet[i] := {0}
for x in domain[i] while not(consistent)  // find a consistent value
begin
consistent := true
v[i] := x
for h in (1 .. i-1) while consistent  // check backwards
begin
consistent := (check(v[i],v[h])
if not(consistent) then confSet[i] := confSet[i] \cup \{h\}
end
if not(consistent)
then delete(x,domain[i])
end
```
CBJ: Dechter Formulation

Pseudocode for conflict-directed backjumping removed due to copyright restrictions.

Conflict-directed Back Jumping:
Supporting definitions

Supporting definitions (earlier constraint, earlier minimal conflict set, jumpback set) removed due to copyright restrictions.
To Solve CSP \(<X,D,C>\) We Combine:

1. Reasoning - Arc consistency via constraint propagation
   - Eliminates values that are shown locally to not be a part of any solution.

2. Search
   - Explores consequences of committing to particular assignments.

Methods That Incorporate Search:
- Standard Search
- Back Track Search (BT)
- BT with Forward Checking (FC)
- Dynamic Variable Ordering (DV)
- Iterative Repair (IR)
- Conflict-directed Back Jumping (CBJ)