Look Back Search

Cedric Geissmann

University Basel

October 31, 2014

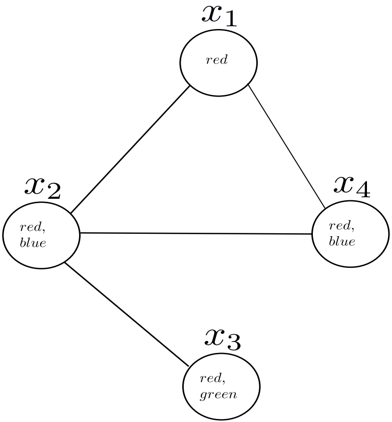
Outline

1. Backtracking
2. Backjumping
   - Gaschnig's Backjumping
   - Graph-Based Backjumping
   - Conflict-Directed Backjumping
3. Learning
   - Graph-Based Learning
   - Conflict-Directed Learning

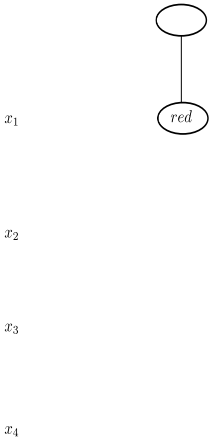
Backtracking

- Backtracking is a basic algorithm used to solve a CSP
- Backtracking has to explore every node in a search tree
- Search space can be reduced by Look-ahead

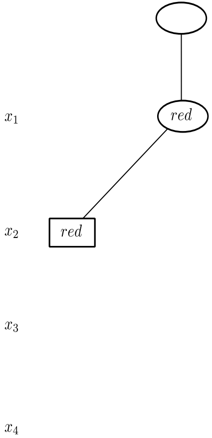
Motivation Example



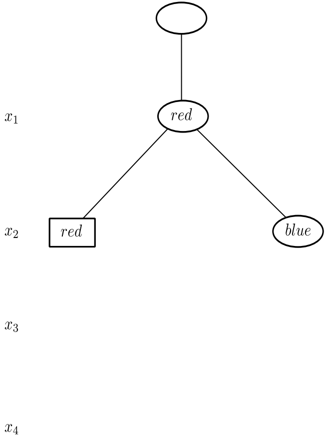
Motivation Example



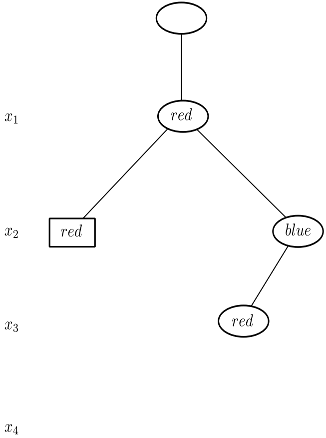
Motivation Example



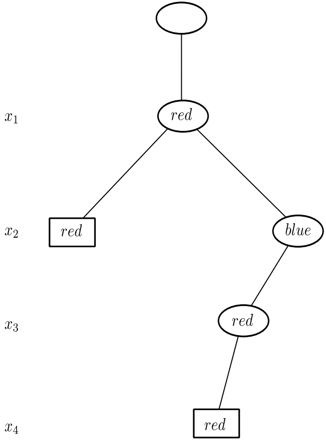
Motivation Example



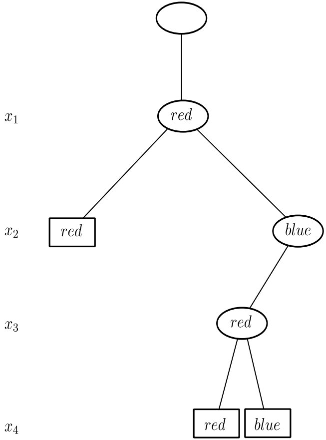
Motivation Example



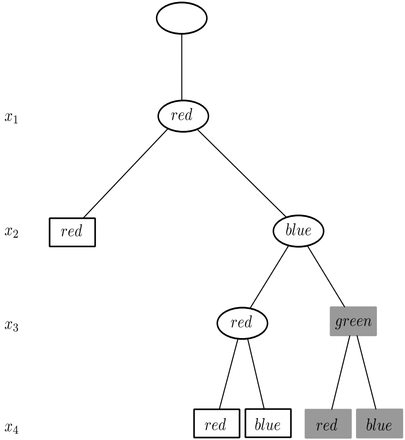
Motivation Example



Motivation Example



Motivation Example



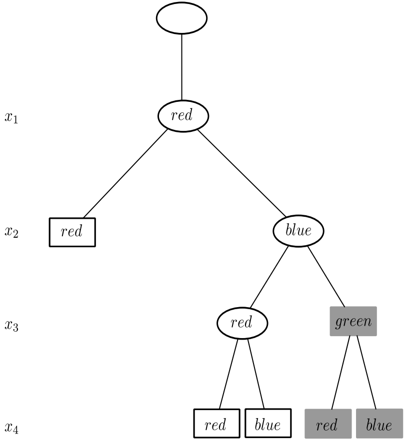
Gaschnig's Backjumping

Dead-End

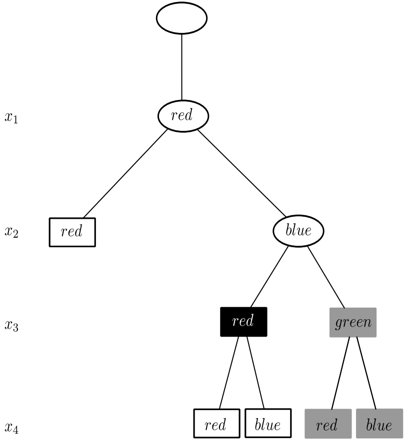
Definition (Dead-End)

- *A dead-end state at level i indicates that a current partial instantiation $\vec{a}_i = (a_1, \ldots, a_i)$ conflicts with every possible value of x_{i+1}.*
- $(a_1, \ldots, a_i)$ *is called a dead-end state, and x_{i+1} is called a dead-end variable.*

Dead-End Example



Dead-End Example



Leaf Dead-End

Definition (Leaf Dead-End)

Let $\vec{a}_i = (a_1, \ldots, a_i)$ be a consistent tuple. If $\vec{a}_i$ is in conflict with x_{i+1}, it is called a leaf dead-end and x_{i+1} is a leaf dead-end variable.

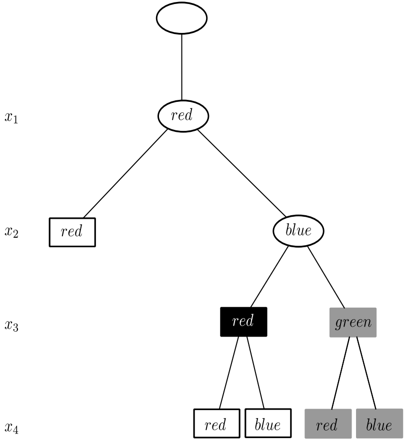
Culprit Variable

Definition (Culprit Variable)

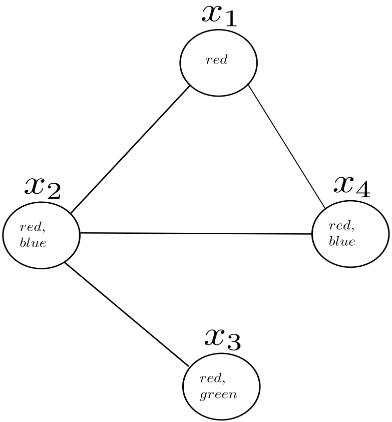
Let $\vec{a_i} = (a_1, \ldots, a_i)$ *be a leaf dead-end.*

- *The culprit index relative to* $\vec{a_i}$ *is defined by* $b = min\{j \leq i \mid \vec{a_j} \text{ conflicts with } x_{i+1}\}$.
- *We define the culprit variable of* $\vec{a_i}$ *to be* x_b.

Culprit Variable Example



Culprit Variable Example



No-Good

Definition (No-Good)

Given a network $\mathcal{R} = (X, D, C)$

- *Any partial instantiation $\bar{a}$ that does not appear in any solution of $\mathcal{R}$ is called a no-good.*
- *Minimal no-goods have no no-good subtuples.*

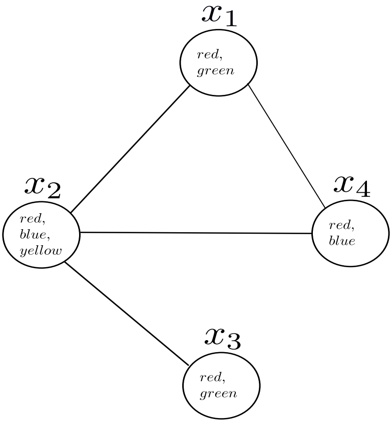
Safe Jump

Definition (Safe Jump)

Let $\vec{a_i} = (a_1, \ldots, a_i)$ *be a leaf dead-end.* x_j *is save when*

- $j \leq i$ *and*
- $\vec{a_j} = (a_i, \ldots, a_j)$ *is a no-good*

Safe Jump Example



Gaschnig's Backjumping Algorithm

Gaschnig's Backjumping Algorithm

Gaschnig's Backjumping Algorithm

$i \leftarrow 1;\ D'_i \leftarrow D_i;\ latest_i \leftarrow 0;$

while $1 \leq i \leq n$ **do**
  instantiate $x_i \leftarrow$ SELECT-VALUE-GBJ;
  if x_i *is null* **then**
    $i \leftarrow latest_i;$
  else
    $i \leftarrow i + 1;\ D'_i \leftarrow D_i;\ latest_i \leftarrow 0;$
  end
end
if $i = 0$ **then**
  return *"inconsistent"*
else
  return *instantiated values of* $\{x_1, \ldots, x_n\}$
end

Gaschnig's Backjumping Algorithm

$i \leftarrow 1$; $D'_i \leftarrow D_i$; $latest_i \leftarrow 0$;
while $1 \leq i \leq n$ **do**
    instantiate $x_i \leftarrow$ SELECT-VALUE-GBJ;
    if x_i *is null* **then**
        $i \leftarrow latest_i$;
    else
        $i \leftarrow i + 1$; $D'_i \leftarrow D_i$; $latest_i \leftarrow 0$;
    end
end
if $i = 0$ **then**
    return *"inconsistent"*
else
    return *instantiated values of* $\{x_1, \ldots, x_n\}$
end

Gaschnig's Backjumping Algorithm

$i \leftarrow 1$; $D'_i \leftarrow D_i$; $latest_i \leftarrow 0$;
while $1 \leq i \leq n$ **do**
    instantiate $x_i \leftarrow$ SELECT-VALUE-GBJ;
    if x_i *is null* **then**
        $i \leftarrow latest_i$;
    else
        $i \leftarrow i + 1$; $D'_i \leftarrow D_i$; $latest_i \leftarrow 0$;
    end
end
if $i = 0$ **then**
    return *"inconsistent"*
else
    return *instantiated values of* $\{x_1, \ldots, x_n\}$
end

Gaschnig's Backjumping Algorithm

$i \leftarrow 1$; $D'_i \leftarrow D_i$; $latest_i \leftarrow 0$;
while $1 \leq i \leq n$ **do**
    instantiate $x_i \leftarrow$ SELECT-VALUE-GBJ;
    if x_i *is null* **then**
        $i \leftarrow latest_i$;
    else
        $i \leftarrow i + 1$; $D'_i \leftarrow D_i$; $latest_i \leftarrow 0$;
    end
end
if $i = 0$ **then**
    return *"inconsistent"*
else
    return *instantiated values of* $\{x_1, \ldots, x_n\}$
end

Gaschnig's Backjumping Algorithm

$i \leftarrow 1$; $D_i' \leftarrow D_i$; $latest_i \leftarrow 0$;
while $1 \leq i \leq n$ **do**
    instantiate $x_i \leftarrow$ SELECT-VALUE-GBJ;
    if x_i *is null* **then**
        $i \leftarrow latest_i$;
    else
        $i \leftarrow i + 1$; $D_i' \leftarrow D_i$; $latest_i \leftarrow 0$;
    end
end
if $i = 0$ **then**
    return *"inconsistent"*
else
    return *instantiated values of* $\{x_1, \ldots, x_n\}$
end

Gaschnig's Backjumping Algorithm

$i \leftarrow 1$; $D'_i \leftarrow D_i$; $latest_i \leftarrow 0$;
while $1 \leq i \leq n$ **do**
    instantiate $x_i \leftarrow$ SELECT-VALUE-GBJ;
    if x_i *is null* **then**
        $i \leftarrow latest_i$;
    else
        $i \leftarrow i + 1$; $D'_i \leftarrow D_i$; $latest_i \leftarrow 0$;
    end
end
if $i = 0$ **then**
    return *"inconsistent"*
else
    return *instantiated values of* $\{x_1, \ldots, x_n\}$
end

Gaschnig's Backjumping Algorithm

$i \leftarrow 1$; $D'_i \leftarrow D_i$; $latest_i \leftarrow 0$;
while $1 \leq i \leq n$ **do**
    instantiate $x_i \leftarrow$ SELECT-VALUE-GBJ;
    if x_i *is null* **then**
        $i \leftarrow latest_i$;
    else
        $i \leftarrow i + 1$; $D'_i \leftarrow D_i$; $latest_i \leftarrow 0$;
    end
end
if $i = 0$ **then**
    return *"inconsistent"*
else
    return *instantiated values of* $\{x_1, \ldots, x_n\}$
end

SELECT-VALUE-GBJ

SELECT-VALUE-GBJ

SELECT-VALUE-GBJ

while D'_i *is not empty* **do**
  select an arbitrary element $a \in D'_i$, and remove a from D'_i;
  $consistent \leftarrow true$; $k \leftarrow 1$;
  while $k < i$ *and consistent* **do**
    if $k > latest_i$ **then** $latest_i \leftarrow k$;
    if *not* $consistent(\vec{a}_k, x_i = a)$ **then**
      $consistent \leftarrow false$
    else
      $k \leftarrow k + 1$
    end
  end
  if *consistent* **then** **return** a ;
end
return *null*

SELECT-VALUE-GBJ

```
while D'_i is not empty do
    select an arbitrary element a ∈ D'_i, and remove a from D'_i;
    consistent ← true; k ← 1;
    while k < i and consistent do
        if k > latest_i then latest_i ← k ;
        if not consistent(a⃗_k, x_i = a) then
            consistent ← false
        else
            k ← k + 1
        end
    end
    if consistent then return a ;
end
return null
```

SELECT-VALUE-GBJ

while D'_i *is not empty* **do**
    select an arbitrary element $a \in D'_i$, and remove a from D'_i;
    $consistent \leftarrow true$; $k \leftarrow 1$;
    while $k < i$ *and consistent* **do**
        if $k > latest_i$ **then** $latest_i \leftarrow k$;
        if *not* $consistent(\vec{a}_k, x_i = a)$ **then**
            $consistent \leftarrow false$
        else
            $k \leftarrow k + 1$
        end
    end
    if *consistent* **then** **return** a ;
end
return *null*

SELECT-VALUE-GBJ

```
while D'_i is not empty do
    select an arbitrary element a ∈ D'_i, and remove a from D'_i;
    consistent ← true; k ← 1;
    while [k < i and consistent] do
        if k > latest_i then latest_i ← k ;
        if not consistent(a⃗_k, x_i = a) then
            consistent ← false
        else
            k ← k + 1
        end
    end
    if consistent then return a ;
end
return null
```

SELECT-VALUE-GBJ

while D'_i *is not empty* **do**
  select an arbitrary element $a \in D'_i$, and remove a from D'_i;
  $consistent \leftarrow true; k \leftarrow 1$;
  while $k < i$ *and consistent* **do**
    if $k > latest_i$ **then** $latest_i \leftarrow k$;
    if *not* $consistent(\vec{a}_k, x_i = a)$ **then**
      $consistent \leftarrow false$
    else
      $k \leftarrow k + 1$
    end
  end
  if *consistent* **then** **return** a ;
end
return *null*

SELECT-VALUE-GBJ

```
while D'_i is not empty do
    select an arbitrary element a ∈ D'_i, and remove a from D'_i;
    consistent ← true; k ← 1;
    while k < i and consistent do
        if k > latest_i then  latest_i ← k ;
        if not consistent(a⃗_k, x_i = a) then
            consistent ← false;
        else
            k ← k + 1;
        end
    end
    if consistent then  return a ;
end
return null
```

SELECT-VALUE-GBJ

```
while D'_i is not empty do
    select an arbitrary element a ∈ D'_i, and remove a from D'_i;
    consistent ← true; k ← 1;
    while k < i and consistent do
        if k > latest_i then latest_i ← k ;
        if not consistent(a⃗_k, x_i = a) then
            consistent ← false;
        else
            k ← k + 1;
        end
    end
    if consistent then return a ;
end
return null
```

SELECT-VALUE-GBJ

while D'_i *is not empty* **do**
    select an arbitrary element $a \in D'_i$, and remove a from D'_i;
    $consistent \leftarrow true; k \leftarrow 1$;
    while $k < i$ *and consistent* **do**
        if $k > latest_i$ **then** $latest_i \leftarrow k$;
        if *not* $consistent(\vec{a}_k, x_i = a)$ **then**
            $consistent \leftarrow false$;
        else
            $k \leftarrow k + 1$;
        end
    end
    if *consistent* **then** **return** a ;
end
return *null*

SELECT-VALUE-GBJ

while D'_i *is not empty* **do**
  select an arbitrary element $a \in D'_i$, and remove a from D'_i;
  $consistent \leftarrow true; k \leftarrow 1$;
  while $k < i$ *and consistent* **do**
    if $k > latest_i$ **then** $latest_i \leftarrow k$;
    if *not* $consistent(\vec{a}_k, x_i = a)$ **then**
      $consistent \leftarrow false$;
    else
      $k \leftarrow k + 1$;
    end
  end
  if *consistent* **then** **return** a ;
end
return *null*

Gaschnig's Backjumping

- Jumps back to its culprit variable at leaf dead-ends.
- Only performs safe jumps.
- Performs a maximal jump but only in leaf dead-ends.
- Uses simple backtracking on internal dead-ends.

Graph-Based Backjumping

Ancestor, Parent

Definition (Ancestor, Parent)

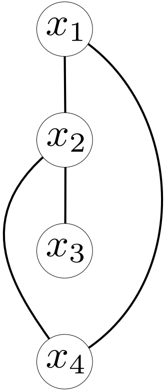
Ancestor set of a variable x

- *all variables that precede x*
- *and are connected to x*

The parent of a variable x

- *is the most recent variable in the ancestor set of x*

Ancestor Example



Invistit, Session

Definition (Invisit, Session)

Invisit of x_i

- *Processing a variable coming from an earlier variable.*

Session of x_i

- *starts on invisit of* x_i
- *ends when at least as high as* x_i
- *holds all variables processed since invisit of* x_i
- x_i *is included in its session*

See example on whiteboard.

Relevant Dead-Ends

Definition (Relevant Dead-Ends)

The relevant dead-ends of x_i's session are

- *just x_i if x_i is a leaf dead-end.*
- *the union of its current relevant dead-ends and the ones encountered in the session.*

See example on whiteboard.

Induced Ancestors, Graph-Based Culprit

Definition (Induced Ancestors, Graph-Based Culprit)

The induced ancestor set of x_i

- *is the union of all ancestors for every relevant dead-end in* x_i*'s session.*

The graph-based culprit

- *is the induced parent of* x_i
- *or the latest variable in* x_i*'s induced ancestor set.*

Induced Ancestors, Graph-Based Culprit Examples

Example

Induced Ancestors, Graph-Based Culprit Examples

Example

- $I_4(\{x_4\})$

Induced Ancestors, Graph-Based Culprit Examples

Example

- $I_4(\{x_4\}) = \{x_1, x_2\}$

Induced Ancestors, Graph-Based Culprit Examples

Example

- $I_4(\{x_4\}) = \{x_1, x_2\}$
- $I_4(\{x_4, x_6\})$

Induced Ancestors, Graph-Based Culprit Examples

Example

- $I_4(\{x_4\}) = \{x_1, x_2\}$
- $I_4(\{x_4, x_6\}) = \{x_1, x_2, x_3\}$

Graph-Based Backjumping Algorithm

Graph-Based Backjumping Algorithm

Graph-Based Backjumping Algorithm

compute $anc(x_i)$ for each x_i;

$i \leftarrow 1$; $D'_i \leftarrow D_i$; $I_i \leftarrow anc(x_i)$;

while $1 \leq i \leq n$ **do**

  instantiate $x_i \leftarrow$ SELECT-VALUE;

  if x_i *is null* **then**

    $iprev \leftarrow i$; $i \leftarrow$ latest index in I_i; $I_i \leftarrow I_i \cup I_{iprev} - \{x_i\}$;

  else

    $i \leftarrow i + 1$; $D'_i \leftarrow D_i$; $I_i \leftarrow anc(x_i)$;

  end

end

if $i = 0$ **then**

  return *"inconsistent"*

else

  return *instantiated values of* $\{x_1, \ldots, x_n\}$

end

Graph-Based Backjumping Algorithm

compute $anc(x_i)$ for each x_i;

$i \leftarrow 1$; $D'_i \leftarrow D_i$; $I_i \leftarrow anc(x_i)$;

while $1 \leq i \leq n$ **do**
  instantiate $x_i \leftarrow$ SELECT-VALUE;
  if x_i *is null* **then**
    $iprev \leftarrow i$; $i \leftarrow$ latest index in I_i; $I_i \leftarrow I_i \cup I_{iprev} - \{x_i\}$;
  else
    $i \leftarrow i + 1$; $D'_i \leftarrow D_i$; $I_i \leftarrow anc(x_i)$;
  end
end
if $i = 0$ **then**
  return *"inconsistent"*
else
  return *instantiated values of* $\{x_1, \ldots, x_n\}$
end

Graph-Based Backjumping Algorithm

compute $anc(x_i)$ for each x_i;
$i \leftarrow 1$; $D'_i \leftarrow D_i$; $I_i \leftarrow anc(x_i)$;
while $1 \leq i \leq n$ **do**
  instantiate $x_i \leftarrow$ SELECT-VALUE;
  if x_i *is null* **then**
    $iprev \leftarrow i$; $i \leftarrow$ latest index in I_i; $I_i \leftarrow I_i \cup I_{iprev} - \{x_i\}$;
  else
    $i \leftarrow i + 1$; $D'_i \leftarrow D_i$; $I_i \leftarrow anc(x_i)$;
  end
end
if $i = 0$ **then**
  return *"inconsistent"*
else
  return *instantiated values of* $\{x_1, \ldots, x_n\}$
end

Graph-Based Backjumping Algorithm

compute $anc(x_i)$ for each x_i;
$i \leftarrow 1$; $D'_i \leftarrow D_i$; $I_i \leftarrow anc(x_i)$;
while $1 \leq i \leq n$ **do**
  instantiate $x_i \leftarrow$ SELECT-VALUE;
  if x_i *is null* **then**
    $iprev \leftarrow i$; $i \leftarrow$ latest index in I_i; $I_i \leftarrow I_i \cup I_{iprev} - \{x_i\}$;
  else
    $i \leftarrow i + 1$; $D'_i \leftarrow D_i$; $I_i \leftarrow anc(x_i)$;
  end
end
if $i = 0$ **then**
  return *"inconsistent"*
else
  return *instantiated values of* $\{x_1, \ldots, x_n\}$
end

Graph-Based Backjumping Algorithm

compute $anc(x_i)$ for each x_i;
$i \leftarrow 1$; $D'_i \leftarrow D_i$; $I_i \leftarrow anc(x_i)$;
while $1 \leq i \leq n$ **do**
  instantiate $x_i \leftarrow$ SELECT-VALUE;
  if x_i *is null* **then**
    $iprev \leftarrow i$; $i \leftarrow$ latest index in I_i; $I_i \leftarrow I_i \cup I_{iprev} - \{x_i\}$;
  else
    $i \leftarrow i + 1$; $D'_i \leftarrow D_i$; $I_i \leftarrow anc(x_i)$;
  end
end
if $i = 0$ **then**
  return *"inconsistent"*
else
  return *instantiated values of* $\{x_1, \ldots, x_n\}$
end

Graph-Based Backjumping Algorithm

compute $anc(x_i)$ for each x_i;
$i \leftarrow 1$; $D'_i \leftarrow D_i$; $I_i \leftarrow anc(x_i)$;
while $1 \leq i \leq n$ **do**
  instantiate $x_i \leftarrow$ SELECT-VALUE;
  if x_i *is null* **then**
    $iprev \leftarrow i$; $i \leftarrow$ latest index in I_i; $I_i \leftarrow I_i \cup I_{iprev} - \{x_i\}$;
  else
    $i \leftarrow i + 1$; $D'_i \leftarrow D_i$; $I_i \leftarrow anc(x_i)$;
  end
end
if $i = 0$ **then**
  return *"inconsistent"*
else
  return *instantiated values of* $\{x_1, \ldots, x_n\}$
end

Graph-Based Backjumping Algorithm

compute $anc(x_i)$ for each x_i;
$i \leftarrow 1$; $D'_i \leftarrow D_i$; $I_i \leftarrow anc(x_i)$;
while $1 \leq i \leq n$ **do**
    instantiate $x_i \leftarrow$ SELECT-VALUE;
    if x_i *is null* **then**
        $iprev \leftarrow i$; $i \leftarrow$ latest index in I_i; $I_i \leftarrow I_i \cup I_{iprev} - \{x_i\}$;
    else
        $i \leftarrow i + 1$; $D'_i \leftarrow D_i$; $I_i \leftarrow anc(x_i)$;
    end
end
if $i = 0$ **then**
    return *"inconsistent"*
else
    return *instantiated values of* $\{x_1, \ldots, x_n\}$
end

Graph-Based Backjumping Algorithm

compute $anc(x_i)$ for each x_i;
$i \leftarrow 1$; $D'_i \leftarrow D_i$; $I_i \leftarrow anc(x_i)$;
while $1 \leq i \leq n$ **do**
  instantiate $x_i \leftarrow$ SELECT-VALUE;
  if x_i *is null* **then**
    $iprev \leftarrow i$; $i \leftarrow$ latest index in I_i; $I_i \leftarrow I_i \cup I_{iprev} - \{x_i\}$;
  else
    $i \leftarrow i + 1$; $D'_i \leftarrow D_i$; $I_i \leftarrow anc(x_i)$;
  end
end
if $i = 0$ **then**
  return *"inconsistent"*
else
  return *instantiated values of* $\{x_1, \ldots, x_n\}$
end

SELECT-VALUE

SELECT-VALUE

SELECT-VALUE

while D'_i *is not empty* **do**
  select an arbitrary element $a \in D'_i$, and remove a from D'_i;
  if $consistent(\vec{a}_{i-1}, x_i = a)$ **then**
    return a
  end
end
return *null*

SELECT-VALUE

while D'_i *is not empty* **do**
    select an arbitrary element $a \in D'_i$, and remove a from D'_i;
    if $consistent(\vec{a}_{i-1}, x_i = a)$ **then**
        return a
    end
end
return *null*

SELECT-VALUE

while D'_i *is not empty* **do**
  select an arbitrary element $a \in D'_i$, and remove a from D'_i;
  if *consistent*$(\vec{a}_{i-1}, x_i = a)$ **then**
    return a
  end
end
return *null*

SELECT-VALUE

while D'_i *is not empty* **do**
  select an arbitrary element $a \in D'_i$, and remove a from D'_i;
  if $consistent(\vec{a}_{i-1}, x_i = a)$ **then**
    return a
  end
end
return *null*

Graph-Based Backjumping

- Can also jump on internal dead-ends.
- Only relies on information from the constraint graph.

Conflict-Directed Backjumping

Conflict-Directed Backjumping

- Combines the ideas of Gaschnig's and Graph-Based Backjumping.
- When detecting a dead-end x_{i+1}, jump back to the latest variable in its jumpback set.

Learning

Learning

- Learn from dead-ends to exclude further occur of the same dead-ends.
- Add no-goods to the constraints.
- Better than backjumping, leads to a smaller tree.

Graph-Based Learning

- Uses graph information collected during the search
- Learn the no-good from a dead-end, that the previous conflicting variables may not be instantiated like this.
- Small overhead, information does not need to be computed.
- Disadvantage: No-goods can be very long and appear late in the search tree.

Conflict-Directed Learning

- Uses information gathered during the search.
- Learn the no-good from a dead-end, that the previous conflicting variables may not be instantiated like this.
- Small overhead, information are already computed.
- Better than Graph-Based Learning, no-goods occur earlier in the search.

Questions?