Overview

Search + Inference

- Search + Incomplete Inference
- Forward Checking
- Maintaining Arc Consistency
- Search + Complete Inference
- Variable Elimination Search
Hybrids: Search + Incomplete Inference

Idea:
- **Search**: backtracking (could be non-chronological)
- **Inference**: at each node, *incomplete inference* on some constraints
  - New nogoods (implicit constraints) discovered
  - If nogood in current branch \( \rightarrow \) backtrack

Effect:
- Search tree reduced: *less nodes* to explore
- Inference at each node: *more work* per node
- Trade-off to find the right balance

---

Hybrids: Search + AC

Idea:
- **Search**: backtracking (could be non-chronological)
- **Inference**: at each node, **AC** on some constraints
  - AC discovers nogoods of size 1
  - Values not AC are eliminated

Effect:
- Future domains reduced: *less nodes* to explore
- AC at each node: *more work* per node
- Very beneficial: *reduces* thrashing
**Forward Checking**

FC is a combination of:
- Search: backtracking
- Inference: at each node, AC on constraints with assigned and unassigned variables

When a domain becomes *empty*:
- No solutions following current branch
- Prune current branch and backtrack

Caution:
- Values removed by AC at level $i$, have to be restored when backtracking at level $i$ or above

---

**Example: FC on 4-queens**

[Diagram showing a tree structure with variables $x_1$ to $x_4$ and a grid for the 4-queens problem, illustrating the search process and solution.]
Binary Forward Checking: Code

```plaintext
function FC (i,Past,[D_1,...,D_n]): bool;
    for all a \in D_i do
        x_i := a;
        if i=n then return TRUE;
        else
            C' := \{C_{ij} | C_{ij} \subseteq C, i<j\};
            NewD := AC({x_1,...,x_n},[{a},D_{i+1},...,D_n],C');
            if Ø \subseteq NewD then
                if FC(i+1,Past\{x_i\},NewD) return TRUE;
                return FALSE;
    return FALSE;
```

Lex variable ordering

Maintaining Arc Consistency

MAC is a combination of:
- Search: backtracking
- Inference: at each node, AC on all constraints
- Preprocess: subproblems are AC

When a domain becomes empty:
- No solutions following current branch
- Prune current branch and backtrack

Caution:
- Values removed by AC at level i, have to be restored when backtracking at level i or above
MAC vs FC: AC on futures

Example: MAC on 4-queens

5 nodes
MAC: Binary Search Tree

Binary tree of subproblems:
- at each level
  - variable $x$, two options: $x = a$ (assignment)  
  - $x \neq a$ (refutation)
- DFS traversal: at each node, AC of current subproblem
- you can change variable without exhausting values!

Example:

```
Example: MAC on 4-queens
```

```
x_1 = 1
```

```
x_1 = 2
```

```
x_2 = 4
```

```
x_3 = 1
```

```
x_4 = 3
```

```
solution
```
**MAC: Code**

```python
function MAC (i, [D_1, ..., D_n]): bool;
    for j:=i+1,..,n do D'_j:=D_j;
    for all a∈D_i do /* x_i:=a */
        D'_i:={a};
        if i=n then return TRUE
    else
        NewD:=AC(X, [D_1, ..., D_i−1, D'_i, ..., D'_n], C);
        if Ø∈NewD then
            if MAC(i+1,NewD) return TRUE;
        D_i := D_i − {a}; /* x_i≠a */
        D'_i:= D_i;
        NewD:=AC(X, [D_1, ..., D_i−1, D'_i, ..., D'_n], C);
        if Ø∈NewD then exit loop
        else for j:=i+1,..,n do D'_j:=NewD[j];
        return FALSE;
```

---

**Search + Complete Inference**

Solution process: sequence of variable processing

How to process a variable? Decision:
- by search
- by complete inference (variable elimination)

If search:
- Tree, branching, backtracking
- After branching, lookahead, new subproblem

If complete inference:
- New problem
VES: Variable Elimination Search

Variable elimination

branching

join, projection

lookahead

graph topology changes

Eliminating $X_2$

X₁

X₂

X₃

X₄

Elim($X₂$)

Elim($X₂$)

Elim($X₂$)

Elim($X₂$)

CSP: Search + Inference
**VES \((k)\)**

- At each node:
  - \(x_i\) \[ select a future variable \]
  - \textbf{if} \(\text{degree}(x_i) \leq k\) \textbf{then} eliminate \(x_i\)
  - \textbf{else} branch on the values of \(x_i\)
    - perform lookahead on branching

- Property:
  - VES(-1) is search
  - VES\((w^*)\) is complete inference

**Example: VES(2)**
Example: VES(2)
Example: VES(2)
Example: VES(2)
Example: VES(2)
Example: VES(2)

VES(k): complexity

**Space:** $O(\exp(k))$

**Time:** $O(\exp(k + z(k)))$

- $z(k)$: number of branched variables
- $z(k)$: it can be computed out of the $k$-restricted induced graph $G^*(k, o)$