NOPT042 Constraint programming: Tutorial 9 - Implicit constraints

What was in Lecture 6

Higher levels of consistencies

- arc consistency: vertex extends to an edge
- path consistency: edge extends to a triangle
- **k-consistency**: any consistent assignment of (k-1) different variables can be extended to one additional variable
- **strong k-consistency**: j-konsistency for all $j \leq k$
- NC is 1-consistency=strong1-consistency, AC is (strong) 2-consistency, PC is (strong) 3-consistency, NC+AC+PC = **strong path consistency**
- example: n-vertex graphs, neither n-consistency nor k-consistency for all $n \leq k$ is enough
- **backtrack-free search**: "for some order of variables we can find a value for each variable compatible with the values of already assigned variables" (k backward edges needs (k+1)-consistency)
- graph width w: minimal width (max number of backward edges) among all node orders
- **Theorem**: strongly $(w+1)$-consistent => there is an order of variables giving a backtrack-free solution
- **directional k-consistency**: in some order of vars, any consistent assignment of k-1 variables can be extended to any kth variable coming after
- **adaptive consistency**: ensure directional i-consistency where i depends on node width
- **[strong] (i,j)-consistency**: extend any consistent assignment on i variables to j additional variables
- **inverse consistency** is (1,k)-consistency: look for support of a value in other variables, neighborhood inverse consistency
- **singleton consistency**: assign value to a variable, then test consistency
- nonbinary constraints, **generalized arc consistency** (GAC), AC-3 adapted for GAC
- **bounds consistency**: GAC only for boundary values of the domains (often used in practice)

What was in Lecture 7?

Symmetry breaking

- example: tournament scheduling (match symmetry, round symmetry)

Global constraints

- faster GAC/filtering algorithm, arbitrary arity, exploit semantics
- **all_different**: domain filtering based on matching in bipartite graphs (remove edges that do not belong to any maximum matching)
- **global_cardinality**: similar, based on network flows
- **lex**: filtering using two pointers)
- **regular:** filtering using "state DAG", rostering (scheduling with sequence constraints)
- **grammar:** sequence generated by CFG? filtering using CYK algorithm
- **slide:** generalizes lex, regular

Scheduling

- how to represent (resources, disjunctive, precedence), disjunction bad (almost no filtering)
- edge_finding, not_first filtering rules

What was in Lecture 8?

Combining search and inference

- search (complete, slow) + consistency (incomplete, fast)
- integrate other solving techniques (e.g. integer programming)
- look-back: maintain consistency among already instantiated vars
- forward-checking: (partial/full) look-ahead, preventing failure better than checking

Variable ordering

- important!
- FAIL FIRST (smaller domain first)
- harder first: most constrained variables, more constraints to past variables

Value ordering

- SUCEED FIRST (prefer values belonging to a solution)
- prefer value with more supports (from AC-4)
- prefer value leading to less domain reduction (compute singleton consistency)
- problem-driven heuristics are better

Branching strategies

- enumeration (X#=0 or X#=1 or ... or X#=N-1)
- step labeling (X#=i or X#!=i)
- bisection/domain-splitting (X#<=i or X#>i)

Cycle cutset

- acyclic constraint network can be solved by backtrack-free AC
- make it acyclic labeling variables on cycles
- cycle cutset = set of vertices whose removal splits all cycles
- heuristics to find: order vertices by degrees, while G cyclic remove first V
- MAC Extended (MACE): combine AC with cycle cutset

In [1]:
```
%load_ext ipicat
```

```
Picat version 3.9
```

Exercise: Seesaw

Adam, Boris, and Cecil want to sit on a 10-feet long seesaw such that they are at least 2 feet apart and the seesaw is balanced. Adam weighs 36 lbs, Boris 32 lbs, and Cecil 16 lbs. Write a general model. (You can assume that the length is even, the distance is integer, and that they can only sit at integer points.)

(Problem from Marriott & Stuckey "Programming with Constraints", page 257. Instance from R. Barták's tutorial.)

Possible decision variables?

- Position on the seesaw for each person.
- Distances between persons, position of the first person, and order of persons.
- Person or empty for each position on the seesaw.

Global constraints? Symmetry breaking? Multiple modeling? Search strategies?

In [2]:
```
!ls seesaw
!cat seesaw/instance1.pi
```

```
instance1.pi  instance2.pi  instance3.pi  instance4.pi  instance5.pi
% sample instance
instance(NumPeople, Length, Distance, Weights) =>
    NumPeople = 3,
    Length = 10,
    Distance = 2,
    Weights = [36, 32, 16].
```

In [3]:
```
!time picat seesaw/.solution/seesaw1.pi seesaw/instance4.pi
!time picat seesaw/.solution/seesaw4.pi seesaw/instance4.pi
```

```
[-16,-15,-14,-13,-12,-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,6,8,7,9,14,15,10,16,1
1,12,13]

real	0m25.095s
user	0m25.078s
sys	0m0.008s
[-8,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,-7,-6,-5,-4,-3,-9,-10,-13,-12,-11,-16,-
14]

real	0m0.024s
user	0m0.013s
sys	0m0.004s
```

Exercise: Golomb's ruler

A Golomb's ruler is an imaginary ruler with n marks such that the distance between every two marks is different. Find the shortest possible ruler for a given n.

(The solution for N=28 was announced on Nov 23, 2022! The length is 585.)

- What length are you able to solve in reasonable time?
- Add suitable implicit constraints. (We will discuss this in class.)

Redundant (implicit) constraints

Redundant constraints do not restrict the solution set but rather express properties of a solution from a different viewpoint. This can lead to

- faster domain reduction,
- a significant boost in propagation,
- improved communication between variables.

We have already seen one example last week in the Magic sequence problem: adding the `scalar_product` constraint.

Implicit constraints based on the following:

$$dist[i,j] = dist[i,i+1] + dist[i+1,i+2] + \ldots + dist[j-1,j]$$

Now estimate distances by 1, sum from i to j:

```
foreach(I in 1..N-1, J in I+1..N)
    Distances[I,J-I] #>= (J-I)*(J-I+1) div 2,
```

```
            Distances[I,J-I] #<= Length - (N-J+I-1)*(N-J+I) div 2
        end
```

In [4]:
```
!picat golomb/.solution/golomb.pi 10
```

```
CPU time 110.194 seconds. Backtracks: 14554575

length = 55
[0,1,6,10,23,26,34,41,53,55]
```

In [5]:
```
!picat golomb/.solution/golomb-improved 10
```

```
CPU time 0.236 seconds. Backtracks: 17432

length = 55
[0,1,6,10,23,26,34,41,53,55]
```

In [6]:
```
!picat golomb/.solution/golomb-improved 11
```

```
CPU time 24.753 seconds. Backtracks: 1224484

length = 72
[0,1,4,13,28,33,47,54,64,70,72]
```