

# NOPT042 Constraint programming: Tutorial 11 - Tabling

## 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

- SUCCEED 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  $X\#\neq i$ )
- bisection/domain-splitting ( $X\#\leq i$  or  $X\#\gt i$ )

### 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

## Dynamic programming with tabling

The "t" in Picat stands for "tabling": storing and resusing subcomputations, most typically used in dynamic programming (divide & conquer). We have already seen the following classical example of usefulness of tabling:

## Example: Fibonacci sequence

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

Picat version 3.7

```
In [2]: %%picat -n fib
fib(0, F) => F = 0.
fib(1, F) => F = 1.
fib(N, F), N > 1 => fib(N - 1, F1), fib(N - 2, F2), F = F1 + F2.
```

```
In [3]: %%picat -n fib_tabled
table
fib_tabled(0, F) => F = 0.
fib_tabled(1, F) => F = 1.
fib_tabled(N, F), N > 1 => fib_tabled(N - 1, F1), fib_tabled(N - 2, F2), F = F1 + F2.
```

Compare the performance:

```
In [4]: %%picat
main =>
    time(fib_tabled(42, F)),
    println(F),
    time(fib(42, F)),
    println(F).
```

CPU time 0.0 seconds.

267914296

CPU time 30.419 seconds.

267914296

## Example: shortest path

Find the shortest path from source to target in a weighted digraph. Code from [the book](#):

```
table(+,+, -,min)

sp(X,Y,Path,W) ?=>
    Path = [(X,Y)],
    edge(X,Y,W).

sp(X,Y,Path,W) =>
```

```

Path = [(X,Z)|Path1],
edge(X,Z,Wxz),
sp(Z,Y,Path1,W1),
W = Wxz+W1.

```

Recall that `?=>` means a backtrackable rule. Consider the following simple instance:

```

index (+, -, -)
edge(a,b,5).
edge(b,c,3).
edge(c,a,9).

source(a).
target(c).

```

In [5]: `!cat shortest-path/instance2.pi`

```

edge(1, 2, 1).
edge(1, 4, 8).
edge(1, 7, 6).
edge(2, 4, 2).
edge(3, 2, 14).
edge(3, 4, 10).
edge(3, 5, 6).
edge(3, 6, 19).
edge(4, 5, 8).
edge(4, 8, 13).
edge(5, 8, 12).
edge(6, 5, 7).
edge(7, 4, 5).
edge(8, 6, 4).
edge(8, 7, 10).

source(1).
target(6).

```

In [6]: `!picat shortest-path/shortest-path instance2`

```

path = [(1,2),(2,4),(4,8),(8,6)]
w = 20

```

In [7]: `!cat shortest-path/shortest-path.pi`

```
% Adapted from Constraint Solving and Planning with Picat, Springer  
% by Neng-Fa Zhou, Hakan Kjellerstrand, and Jonathan Fruhman
```

```
main([Filename]) =>  
  cl(Filename),  
  source(S),  
  target(T),  
  sp(S,T,Path,W),  
  println(path = Path),  
  println(w = W).
```

```
table(+,+, -,min)
```

```
sp(X,Y,Path,W) ?=>  
  Path = [(X,Y)],  
  edge(X,Y,W).
```

```
sp(X,Y,Path,W) =>  
  Path = [(X,Z)|Path1],  
  edge(X,Z,Wxz),  
  sp(Z,Y,Path1,W1),  
  W = Wxz+W1.
```

## Table mode declaration

We can tell Picat what to table using a *table mode declaration*:

```
table(s1,s2,...,sn)  
my_predicate(X1,...,Xn) => ...
```

where `si` is one of the following:

- `+` : input, the row/column/etc. where to store
- `-` : output, the value to store
- `min` or `max` : objective, only store outputs with smallest/largest value of this
- `nt` : not tabled, as if this argument was not passed; last coordinate only, you can use this for global data that do not change in the subproblems, or for arguments functionally dependent (1-1, easily computable) on the `+` arguments

For example:

```
table(+,+, -,min)  
sp(X,Y,Path,W)
```

means for every  $X$  and  $Y$  store (only) the  $Path$  with minimum weight  $W$  (only rewrite  $Path$  if its  $W$  is smaller).

# Index declaration

The *index declaration* `index (+,-,-)` does not change semantics but facilitates faster lookup when unifying e.g. terms `edge(a,X,W)`, see [Wikipedia](#). The `+` means that the corresponding coordinate is indexed ("an input"), `-` means not indexed ("an output"). There can be multiple index patterns, e.g. an undirected graph can be given as:

```
index (+,-) (-,+)
edge(a,b).
edge(a,c).
edge(b,c).
edge(c,b).
```

if we want to traverse the edges in both ways. (This example is from [the guide](#).)

```
In [8]: !cat table-mode-example.pi
```

```
% Adapted from Constraint Solving and Planning with Picat, Springer
% by Neng-Fa Zhou, Hakan Kjellerstrand, and Jonathan Fruhman
main ?=>
    p(a,Y),
    println("Y" = Y).

table(+,max)
index (-,+)
p(a,2).
p(a,1).
p(a,3).
p(b,3).
p(b,4).
```

```
In [9]: !picat table-mode-example
```

```
Y = 3
```

## Exercise: shortest shortest path

Modify the above example so that among the minimum-weight paths, only one with minimum *length*, meaning number of edges, is chosen.

```
In [10]: !cat shortest-path/instance.pi
```

```
index (+,-,-)
```

```
edge(a,b,5).
```

```
edge(b,c,3).
```

```
edge(c,a,9).
```

```
source(a).
```

```
target(c).
```

```
In [11]: !picat shortest-path/shortest-shortest-path instance
```

```
path = [(a,b),(b,c)]
```

```
w = (8,2)
```

```
In [12]: !cat shortest-path/instance3.pi
```

```
!picat shortest-path/shortest-shortest-path instance3
```

```
% this instance is unsatisfiable
```

```
edge(2, 4, 2).
```

```
edge(3, 2, 14).
```

```
edge(3, 4, 10).
```

```
edge(3, 5, 6).
```

```
edge(3, 6, 19).
```

```
edge(4, 5, 8).
```

```
edge(4, 8, 13).
```

```
edge(5, 8, 12).
```

```
edge(6, 5, 7).
```

```
edge(7, 4, 5).
```

```
edge(8, 6, 4).
```

```
edge(8, 7, 10).
```

```
source(1).
```

```
target(6).
```

```
*** error(failed,main/1)
```

```
In [13]: !cat shortest-path/shortest-shortest-path.pi
```

```
% Adapted from Constraint Solving and Planning with Picat, Springer  
% by Neng-Fa Zhou, Hakan Kjellerstrand, and Jonathan Fruhman
```

```
main([Filename]) =>  
  cl(Filename),  
  source(S),  
  target(T),  
  ssp(S,T,Path,W),  
  println(path = Path),  
  println(w = W).
```

```
table(+,+,-,min)
```

```
ssp(X,Y,Path,WL) ?=>  
  Path = [(X,Y)],  
  WL = (Wxy,1),  
  edge(X,Y,Wxy).
```

```
ssp(X,Y,Path,WL) =>  
  Path = [(X,Z)|Path1],  
  edge(X,Z,Wxz),  
  ssp(Z,Y,Path1,WL1),  
  WL1 = (Wzy,Len1),  
  WL = (Wxz+Wzy,Len1+1).
```

```
% The order in `WL = (Weight, Length)` matters, otherwise we would choose minimum-weight path among minimum-edges paths.
```

## Exercise: edit distance

Find the (length of the) shortest sequence of edit operations that transform `Source` string to `Target` string. There are two types of edit operations allowed:

- insert: insert a single character (at any position)
- delete: delete a single character (at any position)

Once you can compute the distance, try also outputting the sequence of operations.

```
In [14]: # this should output 4  
!picat edit-distance/edit.pi saturday sunday
```

```
dist = 4  
[del(2,a),del(2,t),ins(3,n),del(4,r)]
```

```
In [15]: !cat edit-distance/edit.pi
```

```
% Adapted from Constraint Solving and Planning with Picat, Springer  
% by Neng-Fa Zhou, Hakan Kjellerstrand, and Jonathan Fruhman
```

```
main([Source, Target]) =>  
    edit(Source, Target, Distance, Seq, 1),  
    writeln(dist=Distance),  
    writeln(Seq).
```

```
table(+,+,min)
```

```
% base  
edit([],[],D,Seq, I) =>  
    D=0,  
    Seq=[].
```

```
% match  
edit([X|P],[X|T],D,Seq,I) =>  
    edit(P,T,D,Seq,I+1).
```

```
% insert  
edit(P,[X|T],D,Seq,I) ?=>  
    edit(P,T,D1,Seq1,I+1),  
    Seq=[$ins(I,X)|Seq1],  
    D=D1+1.
```

```
% delete  
edit([X|P],T,D,Seq,I) =>  
    edit(P,T,D1,Seq1,I),  
    Seq=[$del(I,X)|Seq1],  
    D=D1+1.
```

## Exercise: 01-knapsack

Write a dynamic program for the 01-knapsack problem.

```
In [16]: !cat knapsack/instance.pi
```

```
instance(ItemNames, Capacity, Values, Weights) =>  
    ItemNames = {"tv", "desktop", "laptop", "tablet", "vase", "bottle", "jacket"},  
    Capacity = 23,  
    Values = {500,350,230,115,180,75,125},  
    Weights = {15,11,5,1,7,3,4}.
```

```
In [17]: !picat knapsack/knapsack instance
```

```
total = 845  
(tv,500,15)  
(laptop,230,5)  
(tablet,115,1)
```

```
In [18]: !cat knapsack/knapsack
```

```
cat: knapsack/knapsack: No such file or directory
```