Sets with Bounded Cardinality

Jimmy Lee & Peter Stuckey

SetSelect Question Revised (baguaBounded-10-8.dzn)

- Given a subset of numbers `1..nSpots` for each symbol in `SYMB`, choose a subset of `1..nSpots` of at most size `size` which includes at most one from each subset and maximize the damage points of the chosen set

```
nSpots = 10;
damage = [10, 8, 4, 2, 6, 9, 5, 3, 8, 10];
size = 3;
SYMB = {'天','澤','火','雷','風','水','山','地'};
group = [{1,4,6}, {1,2,6,7}, {1,3,6,8}, {1,2,3},
     {2,9,10}, {5,6,8,10}, {7,8,10}, {1,3,5}];
```

2

Bounded Cardinality Model (baguaBoundedSet.mzn)

```
int: nSpots;
set of int: SPOT = 1..nSpots;
array[SPOT] of int: damage;
enum SYMB;
array[SYMB] of set of SPOT: group;
int: size;

var set of SPOT: attacks;

constraint forall(s in SYMB)
   (card(attacks intersect group[s]) <= 1);
constraint card(attacks) <= size;

var int: totalDamages =
   sum(p in attacks)(damage[p]);
solve maximize (totalDamages);
```

3

Bounded Cardinality Model (baguaBoundedSet.mzn)

- Executing the model

```
attacks: {1,10} & damage: 20;
```

4

Deciding a Set of Bounded Cardinality

- What about an integer model?
- An array of `size` values

```
array[1..size] of var SPOTx: attacks
```

  - extended SPOT: SPOTx = SPOT ∪ { extra-value }
  - extra value represents: no element
- For example: SPOT = 1..nSpots
  - SPOTx = 0..nSpots

5

Two Critical Issues

- Each solution in the model represents a solution in the problem
  - [3,0,3] ❌ no repeated values
  - [0,2,0] ✅ repeated extra values are OK
- Each solution in the problem has just one solution representative in the model
  - [0,2,0], [0,0,2], [2,0,0] = {2} ❌
  - [0,1,2], [0,2,1], [1,0,2], [1,2,0], [2,0,1], [2,1,0] ❌
- Add constraints to fix these issues

6

Bounded Cardinality Constraints

- Constraints to fix

```
array[1..size] of var SPOTx: attacks;
```

- Just order the values (decreasing)

```
forall(i in 1..size-1)
   (attacks[i] > attacks[i+1]);
```

  - ✗ No representative left for {2} e.g. [2,0,0]

- No strict ordering

```
forall(i in 1..size-1)
   (attacks[i] >= attacks[i+1]);
```

  - ✗ solutions with repeats [3,2,2]

7

Bounded Cardinality Constraints

- Combine the two: repeats of 0 allowed

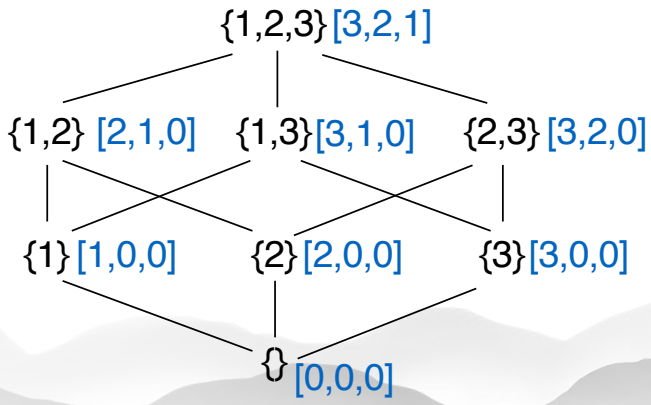
```
forall(i in 1..size-1)
   (attacks[i] >=
      (attacks[i]!=0)+attacks[i+1]);
```

8

Bounded Cardinality Representation

- Representing `var set of {1,2,3}: x;`

```
array[1..3] of var 0..3: x;
```



9

Bounded Cardinality Model (baguaBoundedIntW.mzn)

```
int: nSpots;
set of int: SPOT = 1..nSpots;
array[SPOT] of int: damage;
enum SYMB;
array[SYMB] of set of SPOT: group;
int: size;

set of int: SPOTx = {0} union SPOT;
array[1..size] of var SPOTx: attacks;

constraint forall(i in 1..size-1)(attacks[i] >=
   (attacks[i] != 0) + attacks[i+1]);
constraint forall(s in SYMB)(sum(i in 1..size)
   (attacks[i] in group[s]) <= 1);

var int: totalDamages =
   sum(p in attacks)(damage[p]);
solve maximize (totalDamages);
```

10

Bounded Cardinality Model (baguaBoundedIntW.mzn)

```
int: nSpots;
set of int: SPOT = 1..nSpots;
array[SPOT] of int: damage;
enum SYMB;
array[SYMB] of set of SPOT: group;
int: size;

set of int: SPOTx = {0} union SPOT;
array[1..size] of var SPOTx: attacks;

constraint forall(i in 1..size-1)(attacks[i] >=
   (attacks[i] != 0) + attacks[i+1]);
constraint forall(s in SYMB)(sum(i in 1..size)
   (attacks[i] in group[s]) <= 1);

var int: totalDamages =
   sum(p in attacks)(damage[p]);
solve maximize (totalDamages);
```

Decisions

11

Bounded Cardinality Model (baguaBoundedIntW.mzn)

```
int: nSpots;
set of int: SPOT = 1..nSpots;
array[SPOT] of int: damage;
enum SYMB;
array[SYMB] of set of SPOT: group;
int: size;

set of int: SPOTx = {0} union SPOT;
array[1..size] of var SPOTx: attacks;

constraint forall(i in 1..size-1)(attacks[i] >=
   (attacks[i] != 0) + attacks[i+1]);
constraint forall(s in SYMB)(sum(i in 1..size)
   (attacks[i] in group[s]) <= 1);

var int: totalDamages =
   sum(p in attacks)(damage[p]);
solve maximize (totalDamages);
```

Valid representations

12

Bounded Cardinality Model (baguaBoundedIntW.mzn)

```
int: nSpots;
set of int: SPOT = 1..nSpots;
array[SPOT] of int: damage;
enum SYMB;
array[SYMB] of set of SPOT: group;
int: size;

set of int: SPOTx = {0} union SPOT;
array[1..size] of var SPOTx: attacks;

constraint forall(i in 1..size-1)(attacks[i] >=
   (attacks[i] != 0) + attacks[i+1]);
constraint forall(s in SYMB)(sum(i in 1..size)
   (attacks[i] in group[s]) <= 1);

var int: totalDamages =
   sum(p in attacks)(damage[p]);
solve maximize (totalDamages);
```

At most one intersection

13

Bounded Cardinality Model (baguaBoundedIntW.mzn)

```
int: nSpots;
set of int: SPOT = 1..nSpots;
array[SPOT] of int: damage;
enum SYMB;
array[SYMB] of set of SPOT: group;
int: size;

set of int: SPOTx = {0} union SPOT;
array[1..size] of var SPOTx: attacks;

constraint forall(i in 1..size-1)(attacks[i] >=
   (attacks[i] != 0) + attacks[i+1]);
constraint forall(s in SYMB)(sum(i in 1..size)
   (attacks[i] in group[s]) <= 1);

var int: totalDamages =
   sum(p in attacks)(damage[p]);
solve maximize (totalDamages);
```

Objective

14

Solving the Model

- Executing the model

```
attacks: [9,7,5] & damage: 19;
```

15

Solving the model

- Executing the model

```
attacks: [9,7,5] & damage: 19;
```

Wait a minute …

- Shouldn't we get instead the following?

```
attacks = [10,1,0] & damage: 20;
```

16

Bounded Cardinality Model (baguaBoundedIntW.mzn)

```
int: nSpots;
set of int: SPOT = 1..nSpots;
array[SPOT] of int: damage;
enum SYMB;
array[SYMB] of set of SPOT: group;
int: size;

set of int: SPOTx = {0} union SPOT;
array[1..size] of var SPOTx: attacks;

constraint forall(i in 1..size-1)(attacks[i] >=
   (attacks[i] != 0) + attacks[i+1]);
constraint forall(s in SYMB)(sum(i in 1..size)
   (attacks[i] in group[s]) <= 1);

var int: totalDamages =
   sum(p in attacks)(damage[p]);
solve maximize (totalDamages);
```

17

Bounded Cardinality Model (baguaBoundedInt.mzn)

```
int: nSpots;
set of int: SPOT = 1..nSpots;
array[SPOT] of int: damage;
enum SYMB;
array[SYMB] of set of SPOT: group;
int: size;

set of int: SPOTx = {0} union SPOT;
array[1..size] of var SPOTx: attacks;

constraint forall(i in 1..size-1)(attacks[i] >=
   (attacks[i] != 0) + attacks[i+1]);
constraint forall(s in SYMB)(sum(i in 1..size)
   (attacks[i] in group[s]) <= 1);

var int: totalDamages =
   sum(p in attacks where p > 0)(damage[p]);
solve maximize (totalDamages);
```

18

Summary

- There are multiple ways to represent sets
  - var set of OBJ
    - good if the solver natively supports sets
    - good when OBJ is not too big
  - array[OBJ] of var bool / 0..1
    - good when OBJ is not too big
  - array[1..u] of var OBJ
    - only for fixed cardinality u
    - good when u is small
  - array[1..u] of var OBJx
    - need to represent the “null” object

19

Summary

- The SetSelect problem (without cardinality constraints) is known as the weighted set packing problem a well studied NP complete problem from combinatorics
- Set packing is the dual of set covering, one of the most studied combinatorics problems

20

Image Credits

All graphics by Marti Wong, ©The Chinese University of Hong Kong and the University of Melbourne 2016