Jackie Chan
301310345

Project Part 1 Report

Problem Choice and Solver

I chose to solve the Pizza Problem. My specification was written for MiniZinc's solver, and test instances were tested with MiniZinc's built-in Chuffed 0.10.4 solver configuration.

I worked on the decision version of the problem, and my solver outputs all costs which satisfy the test instance, with the last output being the most optimal cost my solver could find.

How I Solved the Problem

After familiarizing myself with MiniZinc's language, I simply translated the specifications given in class to work with MiniZinc's solver. I used int arrays to hold the prices of each pizza, and the buy/free properties of each coupon. Paid pizzas and used coupons were stored in sets of pizzas and coupons, respectively. Justifies and UsedFor were stored as 2d boolean arrays, with each index corresponding to a coupon and a pizza. The constraints were directly translated into MiniZinc's language, and an additional constraint was added: do not reuse coupons.

Test Instances

After testing with the instances provided on the contest website, I constructed new instances similar to the instances provided, with random price, buy/free values, and pizza/coupon numbers. I also constructed instances with more extreme values, such as a very large amount of pizzas, or very few coupons, in order to ensure that the solver still works properly with more radical instances.

Parameters for the Solver

Instances were tested with the built-in Chuffed 0.10.4 solver configuration, within MiniZinc's IDE.

Jackie Chan
301310345

Results

Note: The most optimal result is being recorded here, but the solver outputs all satisfying costs.

Instance #	Runtime	Result
1	152ms	cost(50)
2	184ms	cost(122)
3	141ms	cost(75)
4	2s 485ms	cost(228)
5	229ms	cost(197)
6	141ms	=====UNSATISFIABLE=====
7	121ms	=====UNSATISFIABLE=====
8	154ms	=====UNSATISFIABLE=====
9	139ms	=====UNSATISFIABLE=====
10	1s 827ms	=====UNSATISFIABLE=====
11	2s 840ms	cost(377)
12	153ms	cost(155)

Jackie Chan
301310345

Specification

```
% Pizza Problem
int: n; % num pizzas
int: m; % num coupons
int: k; % cost bound
set of int: pizzas = 1..n;
set of int: coupons = 1..m;

array[pizzas] of var int: price;
array[coupons] of var int: buy;
array[coupons] of var int: free;

var set of pizzas: paid; % set of pizzas we pay for
var set of coupons: used; % set of coupons we use

array[coupons, pizzas] of var bool: justifies; % true where pizza is purchased to
justify using coupon
array[coupons, pizzas] of var bool: usedFor; % true where pizza is free by using
coupon

% Constraints from Pizza Problem notes
constraint forall(p in pizzas) ((p in paid) <-> not exists(c in coupons) (usedFor[c,
p]));
constraint forall(c in coupons) ((c in used) <-> exists(p in pizzas) (usedFor[c,
p]));
constraint forall(c in coupons) ((c in used) <-> sum(p in
pizzas)(bool2int(justifies[c,p])) >= buy[c]);
constraint forall(c in coupons) (sum(p in pizzas)(bool2int(usedFor[c,p])) <=
free[c]);
constraint forall(c in coupons) (forall(p1, p2 in pizzas where p1 != p2)
((usedFor[c,p1] /\ justifies[c,p2]) -> (price[p1] <= price[p2])));
constraint forall(p in pizzas) (forall(c in coupons) (justifies[c,p] -> (p in
paid)));
var int: cost = sum(p in pizzas) (bool2int(p in paid) * price[p]);
constraint cost <= k;
constraint forall(c in coupons) (forall(p in pizzas) (justifies[c,p] -> ((c in 1..m)
/\ (p in 1..n))));
constraint forall(c in coupons) (forall(p in pizzas) (usedFor[c,p] -> ((c in 1..m) /\
(p in 1..n))));
% Don't reuse coupons
constraint forall(c1, c2 in coupons) (forall(p in pizzas) ((justifies[c1,p] /\
justifies[c2,p]) -> c1 = c2));

solve minimize cost;
output ["cost(" ++ show(cost) ++ ")"];
```