Jackie Chan
301310345

Project 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) ++ ")"];
```

Jackie Chan
301310345

Part II

Question Explored

Upon testing my specification with MiniZinc's built-in solvers, I noticed significant discrepancies in efficiency between the different solvers. Furthermore, it seemed that some solvers were simply unable to solve specific test instances, while the Chuffed solver seemed to perform better than other solvers with all the instances I used in part 1. Thus, I wanted to investigate whether this would remain true for all instance types, or if there was some quality of test instance where the Chuffed solver would not be superior.

With this experiment, I attempt to compare and contrast the strengths and weaknesses with respect to test instance qualities of each of the following solvers:

Chuffed 0.10.4	COIN-BC 2.10/1.17	Gecode 6.1.1

findMUS 0.5.0 was omitted because it only lists unsatisfiable models.

Strategy

Each test instance differs in 5 main qualities:

Pizza count n	Pizza pricing p	Coupon count m	Coupon free : buy ratio c	Cost bound k

Each test instance will output multiple total costs, which will be defined as the set $cost$. Cost bound k will be defined based on the lowest cost outputted, $min(cost)$.

In order to discover the strengths and weaknesses of each solver, I tested each solver with different groups of instances categorized by a set of qualities, utilizing my original specification.

An example of a categorization is such:

Example group: $5 \leq n \leq 7,\ 10 \leq p \leq 20,\ 1 \leq m \leq 3,\ (1:1) \leq c \leq (1:3),\ k = min(cost)$

In the above example, the test instances in the group have between 5 and 7 pizzas, each costing between 10 and 20, between 1 and 3 coupon(s), each having a free : buy ratio of between 1:1 and 1:3, and a cost bound equal to the minimum output.

Jackie Chan
301310345

Test Instance Groups

When defining each test group, all criteria between groups were kept equal except for one quality, in order to isolate any qualities which affect performance more significantly.

The groups of test instances used for this experiment are as follows:

1. Control:
   $$8 \le n \le 10,\ 15 \le p \le 25,\ 6 \le m \le 8,\ (1:3) \le c \le (1:2),\ k = min(cost)$$
2. High pizza count:
   $$n = 15,\ 15 \le p \le 25,\ 6 \le m \le 8,\ (1:3) \le c \le (1:2),\ k = min(cost)$$
3. High prices:
   $$8 \le n \le 10,\ 15000000 \le p \le 20000000,\ 6 \le m \le 8,\ (1:3) \le c \le (1:2),\ k = min(cost)$$
4. No coupons:
   $$100 \le n \le 110,\ 15 \le p \le 25,\ m = 0,\ c\ empty,\ k = min(cost)$$
   a. Note: Greatly increased the number of pizzas in order to increase workload.
5. High coupon count:
   $$8 \le n \le 10,\ 15 \le p \le 25,\ 20 \le m \le 30,\ (1:3) \le c \le (1:2),\ k = min(cost)$$
6. Valuable coupons:
   $$8 \le n \le 10,\ 15 \le p \le 25,\ 6 \le m \le 8,\ (2:1) \le c \le (3:1),\ k = min(cost)$$
7. High pizza and coupon count
   $$n = 12,\ 15 \le p \le 25,\ m = 10,\ (1:3) \le c \le (1:2),\ k = min(cost)$$
8. Infinite cost bound:
   $$n = 15,\ 15 \le p \le 25,\ 6 \le m \le 8,\ (1:3) \le c \le (1:2),\ k = \infty$$
   a. Note: To better isolate the effects of an unlimited cost bound on efficiency, this uses the exact same instances as the High Pizza and Coupon Count group, with the only difference being the infinite cost bound.
9. Unsatisfiable cost bound:
   $$8 \le n \le 10,\ 15 \le p \le 25,\ 6 \le m \le 8,\ (1:3) \le c \le (1:2),\ k = min(cost) - 20$$

Jackie Chan
301310345

Experiment

Each group of test instances contains 5 instances with randomized values within the bounds. Each instance corresponds to a unique .dzn file, generated with the instance_generator.py script included in the project package. I created this script in order to ease instance generation, and every quality is generated by this script except k, which I determine by running each instance once first.

Each instance in each group was run 9 times, 3 times per solver. The average time was then taken of the 3, and recorded in the tables below. For time purposes, instances that took longer than 2 minutes were labelled as “NO RESULT”.

The test machine this was run on is a Windows PC with the following relevant specifications:

Operating System	CPU	RAM
Windows 10	Intel i5 7600K	16 GB 3000MHz

Each cell per table contains the average time it took over 3 runs to reach a conclusion.

Control

Instance #	Chuffed 0.10.4	COIN-BC 2.10/1.17	Gecode 6.1.1
1	290ms	561ms	414ms
2	280ms	578ms	461ms
3	385ms	2s 32ms	3s 187ms
4	256ms	1s 440ms	374ms
5	277ms	2s 157ms	771ms

Jackie Chan
301310345

High Pizza Count

Instance #	Chuffed 0.10.4	COIN-BC 2.10/1.17	Gecode 6.1.1
1	1s 567ms	2s 290ms	1m 14s
2	3s 657ms	5s 312ms	1m 21s
3	54s 473ms	5s 186ms	NO RESULT
4	31s 196ms	18s 271ms	NO RESULT
5	13s 119ms	6s 518ms	NO RESULT

High Prices

Instance #	Chuffed 0.10.4	COIN-BC 2.10/1.17	Gecode 6.1.1
1	404ms	1s 411ms	408ms
2	170ms	677ms	177ms
3	560ms	445ms	575ms
4	162ms	283ms	170ms
5	218ms	1s 80ms	327ms

No Coupons

Instance #	Chuffed 0.10.4	COIN-BC 2.10/1.17	Gecode 6.1.1
1	122ms	84ms	126ms
2	118ms	88ms	123ms
3	130ms	88ms	126ms
4	126ms	80ms	126ms
5	132ms	94ms	130ms

Jackie Chan
301310345

High Coupon Count

Instance #	Chuffed 0.10.4	COIN-BC 2.10/1.17	Gecode 6.1.1
1	1s 165ms	21s 299ms	23s 379ms
2	1s 316ms	28s 662ms	30s 906ms
3	35s 116ms	NO RESULT	NO RESULT
4	17s 853ms	1m 29s	1m 28s
5	2s 921ms	37s 243ms	1m 20s

Valuable Coupons

Instance #	Chuffed 0.10.4	COIN-BC 2.10/1.17	Gecode 6.1.1
1	154ms	1s 923ms	170ms
2	156ms	1s 892ms	181ms
3	157ms	1s 894ms	187ms
4	160ms	1s 845ms	217ms
5	154ms	1s 441ms	143ms

High Pizza + Coupon Count

Instance #	Chuffed 0.10.4	COIN-BC 2.10/1.17	Gecode 6.1.1
1	3s 379ms	6s 775ms	1m 4s
2	3s 776ms	10s 777ms	44s 439ms
3	2s 416ms	4s 295ms	50s 969ms
4	53s 664ms	6s 700ms	NO RESULT
5	1s 553ms	9s 793ms	42s 413ms

Infinite Cost Bound

Instance #	Chuffed 0.10.4	COIN-BC 2.10/1.17	Gecode 6.1.1
1	7s 710ms	6s 281ms	1m 3s
2	2s 846ms	9s 923ms	45s 408ms
3	3s 431ms	2s 344ms	1m 17s
4	1m 21s	8s 921ms	NO RESULT
5	1s 505ms	12s 384ms	42s 562ms

Unsatisfiable Cost Bound

Instance #	Chuffed 0.10.4	COIN-BC 2.10/1.17	Gecode 6.1.1
1	143ms	168ms	142ms
2	172ms	299ms	895ms
3	164ms	271ms	210ms
4	166ms	266ms	261ms
5	151ms	216ms	242ms

Observations

Several conclusions can be drawn from the results gathered from the experiment.

The pricing of pizzas seems to have little to no impact on solver efficiency, with similar averages across all solvers between the control and the high pricing group. Notably, there were instances where the Chuffed and Gecode solvers performed better with the high pricing instances than with the control instances, although it is likely due to the random generation of the instances.

Coupon count had a significant impact on solver performance. All solvers experienced significant slowdowns when running the instances in the high coupon count group. While the Chuffed solver was able to complete all instances in less than 40 seconds, both the COIN-BC solver and the Gecode solver were unable to complete all instances in under 2 minutes, showing significantly worse performance with a high coupon count.

Pizza count had the most significant impact on solver performance. The Chuffed solver experienced heavy slowdowns, taking more than 100x longer to solve instance 3 of the high pizza count group than its slowest performance in the control group. The Gecode solver also experienced severe slowdowns with higher pizza counts, being unable to solve 3 of the 5 instances in the high pizza count group within 2 minutes. The COIN-BC solver experienced slowdowns as well, although not to the same degree as the other solvers.

One strange observation from these experiments was the presence of outliers within each test group. Despite the criteria for each group being fairly tight, and value randomization being kept to a small range, there were instances in some test groups which took significantly longer to solve than others. For example, in the high pizza + coupon count group, the 4th instance took the Chuffed solver more than 14 times longer to solve than the next slowest instance, and took the Gecode solver more than 2 minutes as well. Upon inspection of the instance file, no noticeable discrepancies were found, and inspection of other outliers within test groups did not provide much more information.

Some observations were also gathered from the experiment results regarding the strengths and weaknesses of each solver.

Gecode 6.1.1

Despite variations in all qualities of the pizza problem test instance, the Gecode solver was never the most efficient of the three solvers. However, the Gecode solver still displayed competence when solving some groups of instances.

When solving instances in the valuable coupons group, the Gecode solver was very efficient, displaying results nearly identical to the Chuffed solver and averaging 179.6ms per instance. The Gecode solver was also efficient at solving unsatisfiable instances.

The Gecode solver was not efficient at solving instances with high pizza counts, high coupon counts, or any mixtures of both. In all groups involving either high pizza counts or high coupon counts, there was an instance that the Gecode solver was unable to solve. It is likely that this solver was not designed for the specification provided.

Jackie Chan
301310345

COIN-BC 2.10/1.17

While the COIN-BC solver was not always the most efficient of the 3 solvers, it was able to show its strength in certain niche scenarios.

The COIN-BC solver was very efficient when solving instances in the high pizza count group. While the Chuffed solver took nearly 1 minute to solve the third instance of the group, and took an average of 20.8 seconds to solve each instance, the COIN-BC solver was much more efficient, taking on average only 7.5 seconds to solve each instance. When solving instances with high pizza count, the COIN-BC solver outperformed the Chuffed solver and the Gecode solver significantly. The COIN-BC solver also performed considerably better when solving instances with no coupons, with a 30% increase in speed over the Chuffed solver.

However, the COIN-BC solver displayed much weaker performances in all other test groups. The solver had a higher overhead than the other solvers in the control group, and was significantly slower with instances from the high coupon count group and the valuable coupons group. It seems that the COIN-BC solver performs poorly with instances involving high coupon counts and large free : buy ratios.

Chuffed 0.10.4

Of the 3 solvers, the Chuffed solver was the most consistent and efficient solver for the pizza problem. While not without weaknesses, the Chuffed solver was most efficient in most scenarios.

The Chuffed solver was the most efficient with instances from the control group, the unsatisfiable group, the valuable and high coupon count groups, and the high pizza + coupon count group, excluding the 4th instance. It averaged only 11.67 seconds per instance in the high coupon count group, while both the COIN-BC solver and the Gecode solver were unable to solve all instances within 2 minutes.

Despite its speed at solving instances with high coupon counts, the Chuffed solver is not very efficient at solving instances with high pizza counts. This is its largest weakness based on the results gathered from the experiments, but the Chuffed solver is otherwise very efficient at solving the pizza problem based on my specification.

Jackie Chan
301310345

Conclusion

The exploration performed for this experiment shed some light on the abilities of the different solvers provided in the MiniZinc IDE. While I had originally thought that the Chuffed solver would simply be the fastest of the 3 solvers, my experiments showed that this was not always the case.

After analyzing all of the results, I can conclude that the Chuffed solver is more efficient at pizza problems with a large number of coupons, but the COIN-BC solver is the best of the three at solving pizza problems with a large number of pizzas. Meanwhile, the Gecode solver is likely not suited for any instance of the pizza problem.

If I were to continue this study in the future, I would look into the discrepancies between test instances within each test group. While there is no apparent difference between the instances that take significantly longer and the other instances in the same test group, there is likely some numerical pattern or quality that is causing the solvers to take longer to solve it.

Jackie Chan
301310345

Appendix

- project.zip
  - /control_instances
    - Folder containing instances in the control group
  - /h_coupon_count_instances
    - Folder containing instances in the high coupon count group
  - /h_pizza_count_instances
    - Folder containing instances in the high pizza count group
  - /h_pizza_coupon_instances
    - Folder containing instances in the high pizza + coupon count group
  - /high_prices_instances
    - Folder containing instances in the high prices group
  - /inf_cost_instances
    - Folder containing instances in the infinite cost bound group
  - /no_coupons_instances
    - Folder containing instances in the no coupons group
  - /part1_instances
    - Folder containing instances from part 1 of the project
  - /unsat_cost_instances
    - Folder containing instances in the unsatisfiable cost bound group
  - /valuable_coupons_instances
    - Folder containing instances in the valuable coupons group
  - instance_generator.py
    - Python script responsible for generating the instances in each group
  - pizza_problem_specification.mzn
    - MiniZinc model file, containing the specification for the pizza problem
  - Project Report.pdf
    - PDF file for this report