CMPT 417

Project Report

Zhixin Huang

301326521

Zha48@sfu.ca

Part 1

I chose to work on the Games Problem from LP/CP Programming Contest 2015, and I use the IDP system and MiniZinc for the solver.

Here is the problem specification:

You have decided to come to a block party in Brooklyn, where there are n game booths G1, G2, ..., Gn from one end of the street to the other. You want to play all of the games in the order from G1 to Gn without skipping any. The games are not free, and each game costs 1 token to play once. You can play a game as many times as you can, as long as you have enough tokens. Your pocket can hold at most T tokens, and the pocket is full in the beginning. After you finish one game Gi and before you start Gi+1, you will be refilled with K tokens. However, since the capacity of your pocket is T, any extra tokens will be wasted.

Some games are more fun than others. For each game Gi, you have an integer value Vi that indicates how fun this game is to you. For a game that you don't enjoy, the value can be negative. Note that you have to play each game at least once, even if the game is not fun to you. The total fun you gain from a game is the number of times you play the game times the fun value. You want to manage your tokens so that you gain the most fun from the games.

Input Formats

An input file for LP systems contains a fact of the form num(N), which specifies the number of games N (4 ⩽ N ⩽ 10), a fact of the form cap(T), which means that the capacity of your pocket is T (3 ⩽ T ⩽ 10), a fact of the form refill(K), which gives the number to tokens you receive after each game booth (0 < K ⩽ T); N facts of the form fun(i,Vi), which indicates that the fun value of game Gi is Vi (1 ⩽ i ⩽ N, -10 ⩽ Vi ⩽ 10).

An input file for Minizinc specifies the following constants: num, the number of games N; cap, the capacity of your pocket; refill, the number of tokens you receive after each game booth; fun, an array of fun values of the N games.

Output format

The output should contain exactly one fact of the form total_fun(V), where V is the maximum fun you can gain from playing these games. For ASP systems, the output may consist of multiple answer sets, and only the final one is treated as a solution.

I test ten instances for each of the solver, three of them are from the question's website (http://picat-lang.org/lp_cp_pc/Games.html), and I random write the rest instances. For this problem, if the instances meet the input formats requirements, the result will not be unsatisfiable, so all my ten instances are satisfiable.

I used all default settings for the two solvers.

Here are the results:

IDP input	MiniZinc input	output	time (in msec)

num(4).
cap(5).
refill(2).
fun(1,4).
fun(2,1).
fun(3,2).
fun(4,3).
	num = 4;
	cap = 5;
	refill = 2;
	fun = [4,1,2,3];
		35	144

num(4).
cap(5).
refill(2).
fun(1,4).
fun(2,-1).
fun(3,-2).
fun(4,3).
	num = 4;
	cap = 5;
	refill = 2;
	fun = [4,-1,-2,3];
		29	181

num(5).
cap(3).
refill(2).
fun(1,4).
fun(2,1).
fun(3,-2).
fun(4,3).
fun(5,4).
	num = 5;
	cap = 3;
	refill = 2;
	fun = [4,1,-2,3,4];
		30	176

num(5).
cap(5).
refill(2).
fun(1,1).
fun(2,2).
fun(3,3).
fun(4,4).
fun(5,5).
	num = 5;
	cap = 5;
	refill = 2;
	fun = [1,2,3,4,5];
		45	177

IDP input	MiniZinc input	output	time (in msec)

num(6).
cap(5).
refill(3).
fun(1,1).
fun(2,-2).
fun(3,3).
fun(4,-4).
fun(5,5).
fun(5,5).
fun(6,-6).
	num = 6;
	cap = 5;
	refill = 3;
	fun = [1,-2,3,-4,5,-6];
		33	151

num(7).
cap(10).
refill(5).
fun(1,1).
fun(2,2).
fun(3,3).
fun(4,4).
fun(5,5).
fun(6,6).
fun(7,7).
	num = 7;
	cap = 10;
	refill = 5;
	fun = [1,2,3,4,5,6,7];
		175	173

num(8).
cap(2).
refill(1).
fun(1,1).
fun(2,2).
fun(3,3).
fun(4,4).
fun(5,5).
fun(6,6).
fun(7,7).
fun(8,8).
	num = 8;
	cap = 2;
	refill = 1;
	fun = [1,2,3,4,5,6,7,8];
		44	151

num(10).
cap(5).
refill(4).
fun(1,1).
fun(2,2).
fun(3,3).
fun(4,4).
fun(5,5).
fun(6,6).
fun(7,7).
fun(8,8).
fun(9,9).
fun(10,10).
	num = 10;
	cap = 5;
	refill = 4;
	fun = [1,2,3,4,5,6,7,8,9,10];
		230	149

IDP input	MiniZinc input	output	time (in msec)

num(10).
cap(10).
refill(4).
fun(1,1).
fun(2,2).
fun(3,3).
fun(4,4).
fun(5,5).
fun(6,6).
fun(7,7).
fun(8,8).
fun(9,9).
fun(10,10).
	num = 10;
	cap = 10;
	refill = 4;
	fun = [1,2,3,4,5,6,7,8,9,10];
		280	3797

num(10).
cap(10).
refill(1).
fun(1,1).
fun(2,2).
fun(3,3).
fun(4,4).
fun(5,5).
fun(6,6).
fun(7,7).
fun(8,8).
fun(9,9).
fun(10,10).
	num = 10;
	cap = 10;
	refill = 1;
	fun = [1,2,3,4,5,6,7,8,9,10];
		145	160

Part 2

I am exploring the relationship between refill number and running time, from the last two rows for part1 result, the only difference is refill number, we can see a huge difference between the running time.

I plan to answer this question by trying different input, and find the pattern.

Here is the experiment and result:

input	output	time (in msec)

num = 10;
cap = 10;
refill = 1;
fun = [1,2,3,4,5,6,7,8,9,10];
	145	155

num = 10;
cap = 10;
refill = 2;
fun = [1,2,3,4,5,6,7,8,9,10];
	192	312

num = 10;
cap = 10;
refill = 3;
fun = [1,2,3,4,5,6,7,8,9,10];
	235	1120

num = 10;
cap = 10;
refill = 4;
fun = [1,2,3,4,5,6,7,8,9,10];
	280	3853

num = 10;
cap = 10;
refill = 5;
fun = [1,2,3,4,5,6,7,8,9,10];
	325	5630

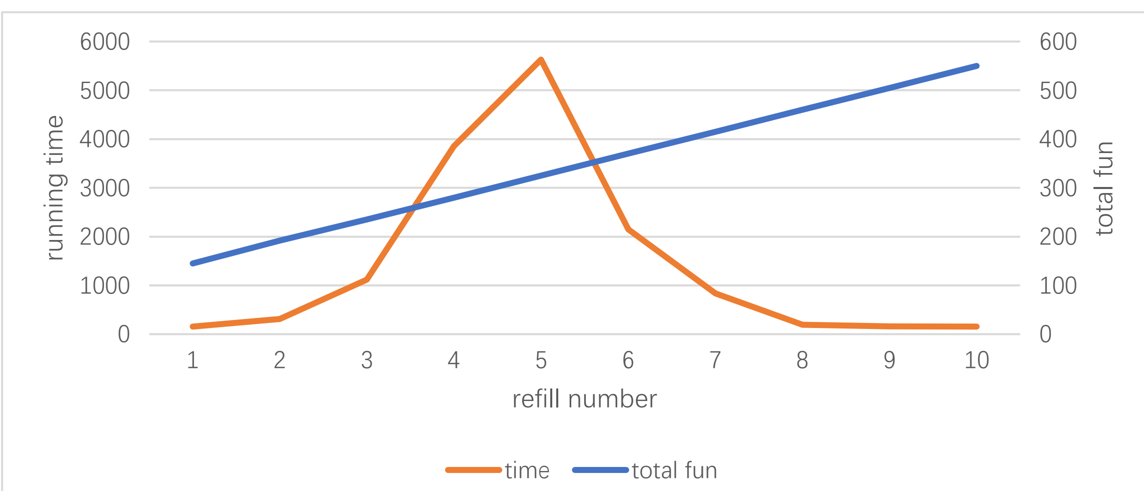
num = 10;
cap = 10;
refill = 6;
fun = [1,2,3,4,5,6,7,8,9,10];
	370	2150

num = 10;
cap = 10;
refill = 7;
fun = [1,2,3,4,5,6,7,8,9,10];
	415	834

num = 10;
cap = 10;
refill = 8;
fun = [1,2,3,4,5,6,7,8,9,10];
	460	192

num = 10;
cap = 10;
refill = 9;
fun = [1,2,3,4,5,6,7,8,9,10];
	505	159

num = 10;
cap = 10;
refill = 10;
fun = [1,2,3,4,5,6,7,8,9,10];
	550	155



My exploration goes some way toward answering my question. From the above figure, we can see that the closer the refill number to 5, which is half of capacity, the longer the running time. From 1 to 5, the time grows near exponentially, and from 5 to 10, the time decreases near exponentially. This may because when the refill number near 5, the solver need more time to find out it should spend how many tokens in each game to maximize the total fun, but for those refill number near 1, the solver just speed more tokens on higher fun value games, and for those refill number near 10, the solver can use almost all tokens in each game, because it get refilled to full capacity after each game, those two strategy should speed less time. We can also see that the total fun increases linearly with the increase of refill number. I may exploring the relationship between capacity and running time in the future.

Part 3

game.idp

game.mzn

readme.txt

report.pdf