CMPT 417 – Intelligent Systems

Project Report: Sequential Games

Presented by:

Bowen Wang – 301267523

Dec. 2nd, 2019

Introduction

This project aims to solving the Sequential Games problem given in *LP/CP Programming Contest 2015.* There are n game booths G1, G2, ..., Gn from one end of the street to the other. A player wants to play all of the games in the order from G1 to Gn without skipping any. The goal is with some rules of game playing, help the player manage his tokens so that he can gain total fun at least K.

Solving for solution that can exceed K is a decision problem, but it makes more sense to solve for maximum of total fun can be attain with certain setting of the playground. As we are looking for the sequence of game playing with maximum total fun, the problem is an optimization problem combine with decision problem. The solver solves for the max fun only if at least K fun has been attained. In this report, I use MiniZinc solver to help define vocabulary and find satisfiable instances to Sequential Games problem.

Problem Specification

This problem is defined in course notes section 12:

you are to play a sequence of games G1, G2, . . ., Gn, under the following conditions:

1. You may play each game multiple times, but all plays of game Gi must be made after playing Gi−1 and before playing Gi+1.
2. You pay 1 token each time you play a game, and you may play a game at most as many times as the number of tokens you have when you begin playing that game.
3. You must play each game at least once.
4. You have C tokens when you start playing G1. After your last play of Gi, but before you begin playing Gi+1, you receive a “refill” of up to R tokens. However, you have some “pocket capacity” C, and you are never allowed to have more then C tokens.
5. Each game has a “fun” value for you, which may be negative.

The goal is to decide how many times to play each game, so that the total fun can be maximized.

The problem can be described as:

Input Vocabulary [*num, fun, cap, refill,* K]**:**

– Number *num* ∈ N of games;
– Fun value *fun*[i] ∈ Z for each game i ∈ [n];
– Pocket Capacity *cap* ∈ N;
– Refill amount *refill* ∈ N;
– Goal K ∈ N.

Output Vocabulary [*num, fun, cap, refill,* K, *p, t*]**:**

–*p*[i] ∈ Z for each game i∈ Z, times of playing game i.

–*t*[i] ∈ Z for each game i∈ Z, number of tokens before playing game i

Goal: Find a number of plays *p*[i] for each game i ∈ N such that maximize the total fun if total fun is at least K.

The constraints on t and p are:

1. Total fun gained by playing all n games must be at least K

$$\sum_{i=1}^{n} \quad sum\,(i, i \in N, play(i) \cdot fun(i)) \geq K$$

2. The number of tokens ti available to play game i is C when we start playing the first game, and for i > 1 is the minimum of C and ti−1 − pi−1 + R:

   t(1)=C ∧ ∀i[1<i≤n→∃x((x=t(i−1)−p(i−1)+R)∧

   (x > C → t(i) = C) ∧ (x ≤ C → t(i) = x))]

3. We play each game at least once, and at most t[i] times:

   ∀i[(1 < i ≤ n) → (1 ≤ p(i) ≤ t(i))]

Testing

The constraints are implemented in *MiniZinc* file *Sequential_Games.mzn* with 14 test cases named *test_case1.dzn* to *test_case14.dzn.* All the testing in this project is under default setting.

The test case 1-3 is testing the functionality and correctness of the code with small set of positive and negative values of input. Then, I designed test 4-7 with identical input value changed to find the relationship between input values and satisfiability. Lastly, test 8-14 are testing the running time of the solver when solving problem with different refill amount to find out whether there is a relationship between refill amount and solving time. While doing test 8-14, I find there is a significant increase on running time when increase refill amount from 2 to 3. After that, the running time trend to be linearly increasing. This is the interesting observation I find in part 1 of this project.

The detailed list of test instances is as below:

- Test1: Given in *LP/CP Programming Contest 2015*

  num = 4;

  cap = 5;

  refill = 2;

  fun = [4,1,2,3];

  K = 25;

- Test2: Given in *LP/CP Programming Contest 2015*

num = 4;
cap = 5;
refill = 2;
fun = [4,-1,-2,3];
K = 25;

- Test3: *Similar to test 1, but change refill from 2 to 3*

  num = 4;
  cap = 5;
  refill = 3;
  fun = [4,1,2,3];
  K = 25;

- Test4: *add more games into the playground*

  num = 6;
  cap = 3;
  refill = 2;
  fun = [1,-2,9,0,-1,3];
  K = 20;

- Test5: *similar to test 4, change the order of negative fun value*

  num = 6;
  cap = 3;
  refill = 2;
  fun = [1,-2,-1,0,9,3];
  K = 20;

- Test6: *similar to test 4, change the value of K larger*

  num = 6;
  cap = 3;
  refill = 2;
  fun = [1,-2,9,0,-1,3];
  K = 45;

- Test7: *similar to test 1, with more negative fun values*

  num = 4;
  cap = 5;
  refill = 2;
  fun = [-4,1,-2,-3];
  K = 25;

- Test8: *test for run time and refill amount*

  num = 10;
  cap = 8;
  refill = 2;
  fun = [-2,1,2,-3,0,2,-4,5,1,-2];
  K = 79;

- Test9: *test for run time and refill amount*

  num = 10;
  cap = 8;
  refill = 3;
  fun = [-2,1,2,-3,0,2,-4,5,1,-2];
  K = 79;

- Test10: *test for run time and refill amount*

  num = 10;
  cap = 8;
  refill = 4;
  fun = [-2,1,2,-3,0,2,-4,5,1,-2];
  K = 79;

- Test11: *test for run time and refill amount*

  num = 10;
  cap = 8;
  refill = 5;
  fun = [-2,1,2,-3,0,2,-4,5,1,-2];
  K = 79;

- Test12: *test for run time and refill amount*

num = 10;
cap = 8;
refill = 6;
fun = [-2,1,2,-3,0,2,-4,5,1,-2];
K = 79;

- Test13: *test for run time and refill amount*

  num = 10;
  cap = 8;
  refill = 7;
  fun = [-2,1,2,-3,0,2,-4,5,1,-2];
  K = 79;

- Test14: *test for run time and refill amount*

  num = 10;
  cap = 8;
  refill = 8;
  fun = [-2,1,2,-3,0,2,-4,5,1,-2];
  K = 79;

Results

	num	cap	refill	Max Fun	Min Fun	Neg. Fun	Pos. Fun	K	Run Time (ms)	Satisfiable	Max Total Fun
Test 1	4	5	2	4	1	0	4	25	66	Sat	35
Test 2	4	5	2	4	-1	2	2	25	68	Sat	29
Test 3	4	5	3	4	1	0	4	25	70	Sat	42
Test 4	6	3	2	9	-2	2	4	20	64	Sat	36
Test 5	6	3	2	9	-2	2	4	35	65	Sat	40
Test 6	6	3	2	9	-2	2	4	45	65	Un-Sat	41
Test 7	4	5	2	1	-4	3	1	25	61	Un-Sat	-4
Test 8	10	8	2	5	-4	4	6	79	378	Un-Sat	47
Test 9	10	8	3	5	-4	4	6	79	139000	Un-Sat	59
Test 10	10	8	4	5	-4	4	6	79	146000	Un-Sat	67
Test 11	10	8	5	5	-4	4	6	79	155000	Un-Sat	67
Test 12	10	8	6	5	-4	4	6	79	157000	Un-Sat	73
Test 13	10	8	7	5	-4	4	6	79	165000	Un-Sat	67
Test 14	10	8	8	5	-4	4	6	79	174000	Un-Sat	77

Stat of each test and result