Individual Project

Song Shihong, 2015011267

June 11, 2016

Contents

1 Overview

This document will briefly introduce the structure of my project and the testing results of it.

My project implemented three algorithms of it which only differs in the optimizing part.

2 Architecture

This project is divided into following parts:

First, I created some classes representing the base structure of it.

Point donates the information of the position of the given point and the index of it, where 0 means nothing, a positive number means a certain source or terminal and a negative number means there is a block here.

Graph donates the graph's information, where the array index suggests the current occupation of each grid and size means the size of the graph. The points vector means those sets of source and terminal which I'm supposed to connect and blocks means those places that cannot be occupied by any index.

Random is a generator of a graph with given size, given number of sets and blocks. It is implemented using rand().

Test is a test program checking if the solution is legal.

SAT is the implemention of the main algorithm which will be discussed below.

3 Algorithm

Algorithm of formulating the path finding problem into a SAT problem:

Use the method given by the paper:

3.1 Symbols

Let x(i,x,y) donate the i th set's connection situation. x(i,x,y) == 1 suggests that this specific point is occupied by the i th set.

Since z3 can't handle the array over one dimension, I choose to use an one-dimension vector x[] to represent it.

3.2 Some constraints

3.2.1 Blockages

if this specific point(x,y) is occupied by a block, every set will have a !x(i,x,y) added to donate that this point cannot be occupied.

3.2.2 Fluidic constraints

One specific point(x,y) can only be occupied by one set, namely, there should be only one x(p,x,y) true among all the x(i,x,y)(i = 1,2,...,number of set).

3.2.3 Droplet movements

Unlike the method given in the attached paper, our project doesn't take time in account. So the move of the Droplet can be divided into two parts.

The key goal of it is to make sure that every route is connected.

- For those points on the route which are not the source point or the terminal point, it should always be on the route and it should have exactly two degrees. Otherwise it should not be on the route.
- For the source point or the terminal point, it should have exactly one degree to make sure that this set is chosen.

3.2.4 Optimize to get the best result

This can be achieved using the z3 optimizer.

3.2.5 Algorithm 1

We set the optimization goal to be minimizing the variable z, which donate the set which haven't been connect * 2000 + the sum of the length of the route where the set have been connected.

3.2.6 Algorithm 2

We continuously decrease the bound of the number of sets connected and each time, we decrease it by one and set all the included points connected. It's an naive enumerating algorithm but gets pretty good results.

3.2.7 Algorithm 3

We use a clever guess to get the answer more efficiently. Having tested several cases, we can find that the final connected number is close to num = (the number of points)/4 + 1. So we first guess there is a solution where connected number is num, and check if there is one, if we get one, we increase the bound and continuously check until we cannot find an answer. Then we can find the minimal length just with the connected number num.

4 Design Pattern

Because I used too many algorithms, I use strategy design pattern to encapsulate 3 algorthms. And this can be seen in the Strategy folder, where users can change the algorithm through way.

5 Tests : Algorithm 1

NOTE: every certain size, certain points number and certain block number takes 10 test cases.

Unit: s

You will find that those examples where all the source and terminals can be connected together will cause very little time which can always be less than

0.3s, while the case where some certain sets of sources and terminals cannot be connected together will spend incredible long time.

For more detailed statistics of the test case, see the attached *.txts for more information.

5.1 Average time

5.1.1 8*8 tests

Blocks\Pairs	2	3	4
2	0.1460	0.9049	1.1664
3	0.2478	0.3873	1.0725
4	0.0974	0.2887	0.3751

Total average time : 0.5207 s

5.1.2 9*9 tests

Blocks\Pairs	2	3	4
2	0.2960	2.7609	13.1305
3	0.3184	0.6647	14.9980
4	0.1741	1.0185	1.5402

Total average time : 3.8779 s

5.1.3 10*10 tests

Blocks\Pairs	2	3	4
2	8.6234	4.3048	114.0338
3	0.3994	10.8888	58.5512
4	0.5446	2.5268	37.8051

Total average time : 26.4087 s

5.2 Max time

5.2.1 8*8 tests

Blocks\Pairs	2	3	4
2	0.2108	3.4639	3.2401
3	0.7183	1.0214	3.1959
4	0.1133	0.5880	0.8107

5.2.2 9*9 tests

Blocks\Pairs	2	3	4
2	0.9288	15.4157	64.8979
3	0.5175	1.0193	97.9316
4	0.2340	7.3122	3.8715

5.2.3 10*10 tests

Blocks\Pairs	2	3	4
2	67.0383	32.4627	300.6880
3	0.7662	52.9562	228.1820
4	2.4143	9.8274	159.3660

5.3 Min time

5.3.1 8*8 tests

Blocks\Pairs	2	3	4
2	0.0955	0.2202	0.4445
3	0.1081	0.1467	0.4036
4	0.0831	0.1875	0.1913

5.3.2 9*9 tests

Blocks\Pairs	2	3	4
2	0.0859	0.2905	0.6429
3	0.1665	0.4021	0.7086
4	0.1371	0.1388	0.5060

5.3.3 10*10 tests

Blocks\Pairs	2	3	4
2	0.4862	0.5125	10.3992
3	0.1704	0.3655	7.1833
4	0.1066	0.1667	0.8384

6 Tests : Algorithm 2

NOTE: every certain size, certain points number and certain block number takes 10 test cases.

Unit: s

In experiment, Algorithm 2 performs much better than Algorithm 1. It still can be seen that the time will increase in terms of the size of the graph.

6.1 Average time

6.1.1 8*8 tests

Blocks\Pairs	2	3	4
2	0.0424	0.1312	0.1054
3	0.0343	0.1276	0.0676
4	0.0342	0.1040	0.0668

Total average time : 0.0793 s

6.1.2 9*9 tests

Blocks\Pairs	2	3	4
2	0.1567	0.2411	0.3433
3	0.1608	0.2281	0.3503
4	0.0706	0.0935	0.3224

Total average time : 0.2185 s

6.1.3 10*10 tests

Blocks\Pairs	2	3	4
2	0.1758	1.3724	4.2317
3	0.2774	0.2838	1.7544
4	0.5859	1.1333	2.2308

Total average time : 1.3384 s

6.2 Max time

6.2.1 8*8 tests

Blocks\Pairs	2	3	4
2	0.0736	0.2721	0.1230
3	0.0366	0.2206	0.0711
4	0.0383	0.1092	0.0734

6.2.2 9*9 tests

Blocks\Pairs	2	3	4
2	0.2335	0.6656	0.5167
3	0.5280	0.4031	1.7451
4	0.0744	0.1129	0.7542

6.2.3 10*10 tests

Blocks\Pairs	2	3	4
2	0.2437	5.2511	16.1256
3	0.3382	0.5476	3.7979
4	3.3917	3.2579	10.0276

6.3 Min time

6.3.1 8*8 tests

Blocks\Pairs	2	3	4
2	0.0325	0.0923	0.0612
3	0.0299	0.1009	0.0635
4	0.0308	0.0918	0.0564

6.3.2 9*9 tests

Blocks\Pairs	2	3	4
2	0.0658	0.1244	0.1847
3	0.0701	0.1044	0.1235
4	0.0654	0.0813	0.0785

6.3.3 10*10 tests

Blocks\Pairs	2	3	4
2	0.1265	0.1070	1.0627
3	0.1497	0.1022	0.4128
4	0.0949	0.3700	0.7604

7 Tests : Algorithm 3

NOTE: every certain size, certain points number and certain block number takes 10 test cases.

Unit: s

In experiment, Algorithm 3 just like Algorithm 2(Maybe a little slower than 2) although they are not the same in implementation.

Actually, this algorithm performs quite well when the number of pairs is not as small as below. When the number of pairs is about ten, this algorithm will perform better than algorithm 2.

7.1 Average time

7.1.1 8*8 tests

Blocks\Pairs	2	3	4
2	0.0603	0.2227	0.1748
3	0.0411	0.1330	0.0957
4	0.0548	0.1418	0.1042

Total average time : 0.1143 s

7.1.2 9*9 tests

Blocks\Pairs	2	3	4
2	0.1192	0.1475	0.3513
3	0.0465	0.2438	0.8307
4	0.0647	0.2923	0.2268

Total average time : 0.2581 s

7.1.3 10*10 tests

Blocks\Pairs	2	3	4
2	0.8559	2.1951	3.3289
3	0.0748	1.6030	1.7227
4	0.1782	0.6291	1.6541

Total average time : 1.3602 s

7.2 Max time

7.2.1 8*8 tests

Blocks\Pairs	2	3	4
2	0.0821	0.3919	0.5779
3	0.0454	0.2232	0.1119
4	0.0585	0.1510	0.1286

7.2.2 9*9 tests

Blocks\Pairs	2	3	4
2	0.1803	0.1996	0.5650
3	0.0537	0.3295	2.7955
4	0.0910	0.5956	0.2913

7.2.3 10*10 tests

Blocks\Pairs	2	3	4
2	4.0431	4.9762	13.4645
3	0.0817	6.5204	4.0616
4	0.2217	2.6345	4.1065

7.3 Min time

7.3.1 8*8 tests

Blocks\Pairs	2	3	4
2	0.0463	0.1620	0.1094
3	0.0375	0.1147	0.0874
4	0.0438	0.1362	0.0831

7.3.2 9*9 tests

Blocks\Pairs	2	3	4
2	0.0516	0.1250	0.2352
3	0.0436	0.1775	0.3288
4	0.0428	0.0881	0.1892

7.3.3 10*10 tests

Blocks\Pairs	2	3	4
2	0.0662	0.3548	0.4987
3	0.0709	0.3304	0.5519
4	0.1489	0.1676	0.1877