The magic square challenge

Kranzl, Manuel
ai22m038@technikum-wien.at

Rahmani, Saifur Rahman
ai22m055@technikum-wien.at

January 24, 2023

1 Specification of the task

Implement the a EA version of the "Magic Square" finder of size N. Test your program from size N= 3 up to 9 (and more if possible) and find 5 new solutions, that differ from wikipedia/internet.

Upload a working project (preferable compilable gnu C++ including a makefile (make test shall test all cases)) with calculated solutions (PDF) and a README.txt (with instructions on how to use it) in a ZIP or TGZ file.

Our group had to implement the following magic square version:

> *Semi-magic square when its rows and columns sum to give the magic constant.*

2 Parent selection

The fitness function is the sum of absolute errors of all rows and columns. After sorting the squares by their fitness the following classification is made:

- The best 25% of all squares will be parent elements.
  - 40% of them will not be changed due to elitism.
  - The others will be mutated by changing two random elements.
- The other 75% will be overwritten by either a mutation of one parent element or a crossover of two parent elements.

3 Crossover method

For better understanding of the code, here is a brief description of the used method for Crossover.

Note that every square can be stored as vector of length $n\dot{n}$. As every number from 1 to $n\dot{n}$ must be unique in a magic square a simple crossover methods like one-point, N-point or cut-and-splice are not suitable. These methods would create squares with non unique numbers.

Instead of using one-point crossover directly on the two parent squares we used it on their inversion sequence. As the building of such an inversion sequence (described in the next paragraph) is reversible the created inversion sequence gives us the new child square.

An inversion sequence of a permutation is an array of the same length. The i-th entry of this sequence denotes the number of entries in the original permutation which are higher than i before i itself.

2	**1**	6	4	5	3	⇒	**1**					
2	1	6	4	5	3	⇒	1	**0**				
2	1	*6*	*4*	*5*	**3**	⇒	1	0	**3**			
2	1	6	4	5	3	⇒	1	0	3	1	1	0

4 Results

Like described in the README we recommend a population of 100.000. For smaller magic squares (like $N = 3...6$) one can also change this value for a smaller number. For the larger ones this number should not be decreased, as the risk of getting stuck in a local minima gets too high.

Because of many performance enhancing strategies (like using OpenMP and implementing a quick sorting algorithm) as well as the use of 'elitism' it was able to find magic squares up to dimension 11 with acceptable computing times. We believe that with enough time even higher dimensions should be possible with this program.

The following squares were found:

4.1 N=3

4	2	9
8	6	1
3	7	5

7	5	3
6	1	8
2	9	4

8	4	3
6	2	7
1	9	5

4.2 N=4

14	12	7	1
13	11	8	2
3	6	10	15
4	5	9	16

14	2	8	10
3	13	6	12
16	4	9	5
1	15	11	7

7	9	14	4
13	3	8	10
12	16	1	5
2	6	11	15

4.3 N=5

9	6	24	23	3
16	12	5	17	15
7	25	4	11	18
19	2	22	1	21
14	20	10	13	8

1	25	17	20	2
21	15	7	4	18
24	5	22	11	3
6	12	10	14	23
13	8	9	16	19

10	1	23	11	20
25	14	12	6	8
13	5	9	22	16
15	21	18	7	4
2	24	3	19	17

4.4 N=6

30	8	5	15	18	35
9	13	3	36	16	34
24	19	26	4	32	6
21	25	22	31	10	2
20	29	27	11	23	1
7	17	28	14	12	33

32	35	8	31	1	4
9	18	15	28	5	36
17	2	24	10	33	25
29	23	14	16	26	3
11	6	20	19	34	21
13	27	30	7	12	22

31	9	16	28	4	23
12	8	34	18	7	32
30	21	6	10	29	15
14	25	20	3	36	13
5	26	11	35	33	1
19	22	24	17	2	27

4.5 N=7

36	28	22	23	20	5	41
39	21	4	8	27	43	33
47	26	2	32	24	34	10
18	48	35	15	11	17	31
7	38	40	3	12	46	29
19	1	30	49	37	14	25
9	13	42	45	44	16	6

1	40	34	10	20	42	28
2	13	39	33	38	15	35
45	5	11	37	36	16	25
12	43	27	14	26	46	7
48	3	9	41	6	21	47
44	22	24	8	30	18	29
23	49	31	32	19	17	4

4.6 N=8

40	5	46	1	16	60	36	56
61	18	4	58	43	8	54	14
22	41	11	44	34	45	12	51
7	53	49	55	32	30	15	19
50	25	23	9	38	35	47	33
20	28	26	29	39	17	59	42
3	63	37	2	48	52	31	24
57	27	64	62	10	13	6	21

4.7 N=9

40	50	51	42	74	64	26	6	16
81	63	33	19	29	61	67	14	2
18	41	39	31	28	9	68	56	79
36	43	35	58	1	57	76	38	25
7	78	66	34	30	24	27	59	44
69	13	10	53	60	55	37	23	49
45	3	12	80	22	75	15	52	65
62	70	46	47	54	4	21	48	17
11	8	77	5	71	20	32	73	72

4.8 N=10

59	9	4	90	71	96	48	53	42	33
79	99	16	67	32	35	52	49	22	54
15	5	87	19	81	20	72	65	43	98
14	8	47	24	85	100	93	46	61	27
55	97	83	62	3	10	84	73	13	25
80	92	78	51	12	11	2	76	74	29
64	21	36	58	91	60	7	6	94	68
40	95	38	28	45	18	34	30	89	88
17	56	39	75	41	86	63	70	1	57
82	23	77	31	44	69	50	37	66	26

4.9 N=11

91	3	32	101	22	51	6	99	93	94	79
41	90	25	23	102	29	98	42	9	105	107
58	56	108	17	67	70	62	109	30	74	20
106	35	12	45	77	34	117	27	43	54	121
103	68	57	55	115	100	28	71	50	13	11
48	104	31	76	37	52	61	86	40	72	64
19	66	111	21	63	87	10	33	92	89	80
8	83	49	75	110	59	116	97	26	4	44
47	114	73	95	7	118	39	1	88	5	84
38	36	60	85	2	18	120	82	119	65	46
112	16	113	78	69	53	14	24	81	96	15

4.10 N=12

29	136	61	41	79	69	51	70	82	31	96	125
112	33	36	113	6	122	92	104	15	120	100	17
111	133	32	80	86	44	71	18	83	68	55	89
90	143	126	108	50	12	9	45	67	114	5	101
97	91	76	129	66	46	14	53	141	84	60	13
99	49	105	7	65	10	134	117	98	43	123	20
30	87	39	42	131	140	132	52	40	19	37	121
78	109	116	26	124	94	56	57	27	77	4	102
130	11	75	88	28	35	2	110	119	21	144	107
22	3	8	115	139	106	93	48	73	103	137	23
34	59	138	74	95	64	81	142	62	72	24	25
38	16	58	47	1	128	135	54	63	118	85	127

4.11 N=13

87	101	100	51	58	79	138	12	113	73	22	132	139
80	55	13	64	156	65	90	36	123	135	82	122	84
47	20	149	109	27	75	52	151	115	88	93	169	10
69	91	114	130	28	158	99	38	86	78	67	39	108
77	146	23	118	102	120	1	89	59	43	97	76	154
24	127	112	148	141	3	34	32	125	145	92	16	106
166	21	70	129	81	161	153	14	6	37	165	45	57
30	95	9	26	136	134	164	144	98	11	107	41	110
15	162	4	96	66	150	8	116	119	105	128	111	25
159	17	152	7	155	19	2	157	46	103	18	133	137
83	40	142	54	61	49	167	72	31	160	35	163	48
147	62	143	56	44	63	126	104	60	85	68	53	94
121	168	74	117	50	29	71	140	124	42	131	5	33

4.12 Log of N=11

This is the output log of the program (for the final programm this output is put in comments) searching for a 11x11 magic square. It shows that because of the elitism the fitness is constantly decreasing. It also serves as proof that our program did find the larger magic squares itself.

```
[saif@lenovo-p14s magic-square (master)]$ ./msfinder -n 11 -p 100000
0:  fitness: 710
10:  fitness: 391
20:  fitness: 286
30:  fitness: 224
40:  fitness: 150
50:  fitness: 111
60:  fitness: 95
70:  fitness: 84
440:  fitness: 11
450:  fitness: 11
460:  fitness: 10
470:  fitness: 10
480:  fitness: 9
490:  fitness: 9
500:  fitness: 9
510:  fitness: 9
520:  fitness: 9
530:  fitness: 8
540:  fitness: 8
550:  fitness: 8
560:  fitness: 8
570:  fitness: 8
580:  fitness: 6
590:  fitness: 6
600:  fitness: 6
610:  fitness: 6
620:  fitness: 6
630:  fitness: 5
640:  fitness: 5
650:  fitness: 5
660:  fitness: 5
670:  fitness: 5
680:  fitness: 5
690:  fitness: 5
700:  fitness: 4
710:  fitness: 4
720:  fitness: 4
730:  fitness: 4
740:  fitness: 4
750:  fitness: 4
760:  fitness: 4
```

```
770:  fitness: 4
780:  fitness: 4
790:  fitness: 4
800:  fitness: 3
810:  fitness: 3
820:  fitness: 3
830:  fitness: 3
840:  fitness: 2
850:  fitness: 2
860:  fitness: 2
870:  fitness: 2
880:  fitness: 2
890:  fitness: 2
900:  fitness: 2
910:  fitness: 2
920:  fitness: 2
930:  fitness: 2
940:  fitness: 2
950:  fitness: 2
960:  fitness: 2
970:  fitness: 2
980:  fitness: 2
990:  fitness: 2
1000:  fitness: 2
1010:  fitness: 2
1020:  fitness: 1
1030:  fitness: 1
1040:  fitness: 1

Magic Square Alarm!
73 3 22 24 28 60 104 98 61 106 92
9 71 109 74 76 86 32 59 39 4 112
35 78 96 97 8 31 17 110 117 72 10
53 102 99 19 50 18 51 6 95 85 93
42 56 25 116 49 118 88 13 33 94 37
46 20 77 84 29 41 103 43 83 115 30
101 27 2 105 111 5 15 113 66 62 64
36 119 54 12 107 70 90 68 58 34 23
79 65 121 11 87 40 55 114 1 7 91
108 63 52 48 69 120 16 26 80 45 44
89 67 14 81 57 82 100 21 38 47 75
```