A Hybrid Genetic Algorithm for a Variant of Two-Dimensional Packing Problem

Jin Kim
School of Computer Science and Engineering
Seoul National University
599 Gwanak-ro, Gwanak-gu, Seoul 151-744, Korea
kimjin@soar.snu.ac.kr

Byung-Ro Moon
School of Computer Science and Engineering
Seoul National University
599 Gwanak-ro, Gwanak-gu, Seoul 151-744, Korea
moon@snu.ac.kr

ABSTRACT

A variant of two-dimensional packing problem was given in the GECCO'2008 competition. This paper describes the genetic algorithm that produced the best result and thus won the No. 1 prize. As the problem is naturally represented by a two-dimensional chromosome, two-dimensional crossovers are used to generate more diverse chromosomes and effectively maintain geographical linkage among genes. We developed a local search heuristic based on the breadth-first search algorithm; we describe how to implement the heuristic efficiently using problem-specific knowledge. The local search was combined with a steady-state genetic algorithm and the combination showed strong synergy.

Categories and Subject Descriptors

G.1.6 [**Numerical Analysis**]: Optimization—*Constrained optimization*

General Terms

Algorithms, Experimentation

Keywords

Breadth-first search, geographic crossover, hybrid genetic algorithm, local search, packing, two-dimensional

1. INTRODUCTION

The 2008 Genetic and Evolutionary Computation Conference (GECCO, Atlanta, July 12–16, 2008) announced a *2D Packing Problem* competition. [1] The problem is a two-dimensional variant of packing problem. The goal of the problem is to pack a grid with numbers so that the sum of the weights between adjacent cells on the grid is maximized. This is different from other packing problems such as two-dimensional bin packing, strip packing, or cutting stocks which usually pack small rectangles into large rectangles.

Given an $l \times m$ grid and n numbers in $\{0, 1, \ldots, n-1\}$, a non-negative integer weight is given to every pair of the numbers. The two-dimensional packing problem for the competition is to assign a number to every cell of the grid in order to maximize the sum of the pair weights between all adjacent cells, where the term *adjacent* means horizontal, vertical, or diagonal adjacency. This sum corresponds to the *fitness* of the grid, a solution. When summing up the pair weights, the weight of a particular pair is counted only once; without this constraint, the problem becomes trivial, because the best grid can be easily obtained by filling the grid repeatedly with the pair having the largest weight.

3	1	4
1	5	3

pair		weight
1	1	5
1	3	2
1	4	9
2	2	8
2	4	4
4	4	11
3	1	7
3	3	9
4	5	3
4	3	4
5	1	5

Table 1: A sample grid and number pairs that are weighted. The weight of a pair not shown in this table is regarded as zero.

Table 1 shows an example. A sample 2×3 grid is shown on the left and a table of pair weights is shown on the right; all the other pairs not appearing in the table has weight 0. The fitness of the grid is computed as follows. For each pair of adjacent cells on the grid, add the corresponding pair weight if the pair has not been counted yet. For example, we may start with the left upper cell with the number 3. It has three adjacent cells having numbers 1, 5, and 1. The pairs of adjacent cells are $(3, 1)$, $(3, 5)$, and $(3, 1)$; their corresponding pair weights are 7, 0, and 7; although the pair $(3, 1)$ appears more than once, it is only counted once; so we add $7 + 0$ for these pairs. This process is repeated until all pairs of adjacent cells have been considered. In this example, the fitness is 35.

[1] The competition site is accessible at http://www.sigevo.org/gecco-2008/competitions.html.

GECCO'09, July 8–12, 2009, Montréal, Québec, Canada.

As can be seen in the example, if a pair (a, b) is considered once, the symmetric pair (b, a) is eventually considered. Therefore, we can regard the pairs as unordered and the weight of each pair as the sum of the two pair weights. Since we can save the time of computing fitnesses by this change of table, we hereafter modify the weight table in this way.

The competition site announced a 20×20 grid problem with a corresponding pair weight table. The table of pair weights is too large to show here; see the competition site [1] for details. In this paper, we describe the algorithm of the competition winner. It is based on a traditional framework of hybrid steady-state genetic algorithm (GA). Two-dimensional crossovers are used because a solution is naturally represented by a two-dimensional chromosome. Comparing the attractiveness of those crossovers, we chose one of them, the geographic crossover [5]. We also devised a local search heuristic which improves the chromosomes by a breadth-first search (BFS)-based search. The GA showed strong synergy with the local search heuristic.

The competition site shows the history of records in the competition. Starting from a random solution of fitness 1.538×10^8 and the first non-trivial solution of fitness 8.627×10^8, the site eventually published the authors' final winner solution 10.32×10^8. The runner-up solution was 10.27×10^8. After the competition, we eventually improved the solution up to 10.35×10^8. We show both of the winner solution and the improved solution in the experimental report.

This paper is organized as follows. In the next section, we introduce the framework of our GA and the two-dimensional crossovers. In Section 3, we describe our local search heuristic and explain how to implement it efficiently. Experimental results are given in Section 4. The effect of various crossovers and that of hybridization are also reported. Finally, Section 5 gives the concluding remarks.

2. PRELIMINARIES

2.1 Genetic Algorithm

We use a typical hybrid steady-state GA, that produces only one offspring for each generation. Figure 1 shows the flow of a conventional hybrid GA.

```
create initial population;
while stopping condition is not satisfied {
    choose parent0 and parent1 from population;
    offspring ← crossover(parent0, parent1);
    offspring ← mutation(offspring);
    local-search(offspring);
    replace(population, offspring);
}
return the best individual;
```

Figure 1: A conventional hybrid steady-state GA

2.2 Two-Dimensional Crossover

For one-dimensional chromosomes, one-point crossover, multi-point crossover, and uniform crossover [8] are representative. However, for two-dimensional chromosomes, such traditional crossovers have limit to reflect the geographical linkage of genes and make various cutting patterns.

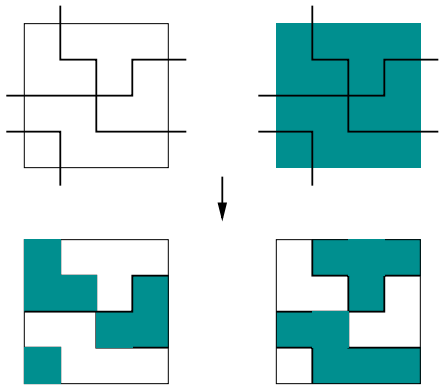

Figure 2: Example mask generation of geographic crossover with three cuts. Two offspring are generated from the masks.

Thus several crossovers were proposed for higher dimensional problems. The first two-dimensional crossover was proposed by Cohoon and Paris [3]. It randomly selects rectangular region from one parent chromosome, and combine it with the other parent. Anderson *et al.* [1] proposed block-uniform crossover. Similar to the uniform crossover, it tessellates parents and copies each part from a randomly selected parent. Bui and Moon [2] proposed Z3 crossover, which is a multi-dimensional generalization of the multi-point crossover. It chooses a specified number of cutting surfaces on multi-dimensional chromosomes and alternately copies areas of parent chromosomes which are made by the cutting surfaces.

Kahng and Moon proposed geographic crossover [5]. In fact, geographic crossover does not indicate only one crossover scheme, but a general class of crossovers. They found that cuts divide the chromosome to two equivalent classes. Thus one can guarantee that cuts properly divide a chromosome. It makes several zigzag cuts on a grid, and copies region by region into offspring. As the cuts can form arbitrary patterns, the geographic crossover produces various cutting patterns. The possible number of k-cut geographic crossover in $l \times l$ two-dimensional chromosomes is $\frac{f(l)}{k} - d$ where $f(l) = 4\frac{2l-1}{l-1} - 2(l+1)$ and d is the number of duplications (i.e., when the same classification is possible via different combinations of cuttings). The geographic crossover produced good result in both a non-hybrid GA and a hybrid GA [6]. Figure 2 shows an example of geographic crossover with three cuts.

Although geographic crossover has been used in many papers [4, 5, 6, 7], they do not explicitly noticed the performance according to the number of cuts. We give some experimental results in Section 4.4.

3. LOCAL SEARCH

It is necessary to define the neighborhood structure of the two-dimensional packing problem before describing local search. For a given grid, we can consider its neighborhood as the matrices that have only one different cell with it, e.g., (a)–(b), (b)–(c), (c)–(d), and (e)–(f) in Figure 4. A local search may consist of replacing each cell with the "best" number, i.e., the number that maximizes the fitness of the grid when it is assigned to the cell. Right after replacing

a cell, it is natural to replace the adjacent cells of the cell. Here, the order to visit each cell affects much to the quality. This is the motivation of our local search described below.

```
// M: the given grid (matrix)
// M_ij: the (i, j)-th cell of M
// Q: an empty queue
Choose a starting cell (x, y) randomly;
Q.enqueue(x, y);
while Q is not empty {
    (x, y) ← Q.dequeue();
    n ← a number for M_xy maximizing the fitness of M;
    M_xy ← n;
    for each (x', y') adjacent to (x, y)
        if (x', y') has not been considered
            Q.enqueue(x', y');
}
```

Figure 3: BFS-based local search

The local search algorithm works like the breadth-first-search (BFS). It first randomly chooses a starting cell on the given grid, and replaces the value of the chosen cell to a number that maximizes the fitness of the grid. Then it repeat the process in the order of BFS traversal in the grid until all the cells are visited. Figure 3 shows the pseudo code. This is a round of the local search; it is repeated until no improvement is observed. In our experiment, the number of repetition was usually between 5 to 10 in early generations, and decreases to 1 or 2 in later generations.

Figure 4 illustrates the progress of our local search on a sample grid. It starts from a randomly selected cell of number 351 that is marked by thick border. After replacing the number of the cell to the best number 347, it adds the adjacent cells into the BFS queue in arbitrary order. Then the right, below, and left-below cells are replaced by the most appropriate numbers and their adjacent cells are added to the queue in turn. Figure (e) shows the grid after visiting the cells of distance one from the starting cell. Then the cells of distance two are visited and so on.

3.1 Speedup

The most frequently used operation in this local search algorithm is to find the best number for a cell. A naïve finding algorithm just tries to assign every possible number to the cell and calculates the sum of pair weights of the assigned number and the numbers of adjacent cells. The process takes $18 \times 18 \times 8 + 18 \times 4 \times 5 + 4 \times 3 = 2964$ addition operations.

Faster algorithm can be derived by considering that there are many zero-weight pairs in the weight table. There are only about 20 percent of weighted pairs and the others are weighted zero. Now we may consider positive-weight pairs only. For each number, we prepare a list of numbers that have positive pair weight with the number. Since the weight table is fixed, we can prepare this one time in advance while reading the weight table. A list will have about 80 (400×0.2) numbers on average. Then, we can refer to the list to compute the sum of weights.

The improved finding algorithm takes advantage of the weighted number lists. This algorithm is similar to the naïve algorithm in that the sum of score for each number is calculated. However, it does not bother to consider the zero-degree number pairs. It calculates scores only referring the numbers in the lists. The pseudo code of this algorithm is given in Figure 5.

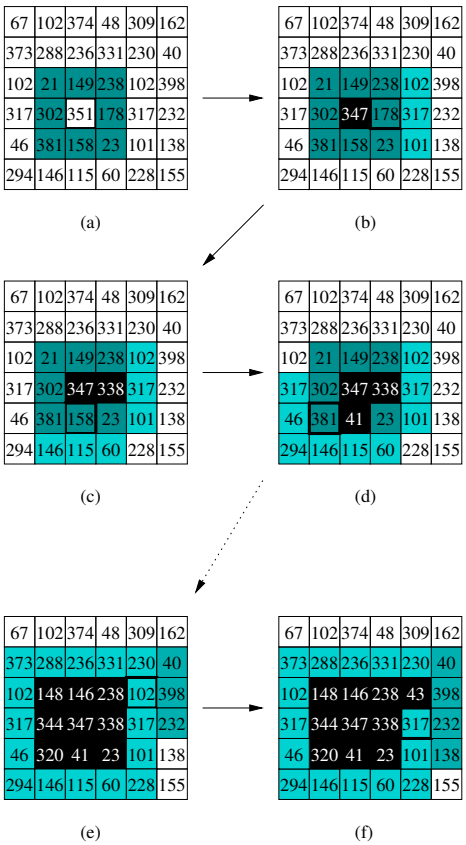

Figure 4: Example of BFS-based local search. Enqueued cells are marked by shade and visited cells are colored black. The degree of shade corresponds to the distance from the starting cell.

```
for each a adjacent to x {
    for each i in the weighted number list of a
        if (a, i) has not been considered
            score[i] ← score[i] +w(a, i);
}
return the index of the maximum score;
```

Figure 5: The improved algorithm finding the best number for a cell

This algorithm requires just 8 times the average degree (80), i.e., about 640 addition operations, which is 5 times faster than the naïve algorithm. Thus the time complexity is reduced from $O(8n)$ to $O(8d)$ where n is the number of rows of weight matrix, and d is the average size of adjacency list. Sparse weight matrices particularly benefit from this.

Since the local search must visit every cell once and the number of cells is $ltimesm$ as we mentioned earlier, the time complexity of local search is $O(8d) \times O(lm) = O(8dlm) = O(dlm)$.

When calculating the fitness, we should check that a pair has been counted. To quickly find out whether a pair was used or not, we use a 400×400 matrix to count the number of references to each number pair. Every element of the matrix is initialized to zero when a local search started, increased by one whenever a number pair corresponding to the element is found, and decreased by one whenever a number pair no longer exists in the grid. When we encounter a pair in the course of optimization, we only add the pair weight only when the corresponding cell has value 0 in the counting matrix.

4. EXPERIMENTAL RESULTS

4.1 Test Instance

The test case was given by the competition website. It is to pack a 20×20 grid with numbers in $\{0, 1, \ldots, 399\}$ Among all possible pairs of numbers, about 10% was selected uniform randomly and assigned weights which are random integers uniformly distributed between 1 and 999,999. Figure 6 shows some part of the pair weights given in the problem; since the whole matrix is 400×400, it shows only a fraction of the matrix. Since a 20 by 20 grid has $2964/2 = 1482$ pairs of adjacent cells, we can obtain a rough upper-bound of the fitness by summing up the largest 1482 pairs: about 15.06×10^8. Although far from easy, a tighter upper-bound may be drown considering duplication and conflicts between pairs.

4.2 Experimental Settings

We conducted 100 runs for each experiment on Core^{TM}2 Duo CPU 2.66GHz with Linux operating system. The hybrid GA was programmed in C++ language and the program was compiled by GNU g++ compiler. A run of the GA took about 20 minutes. The genetic operators and their parameters that we used are summarized in the following:

- Encoding — We encode the grid directly, that is, a chromosome is a two-dimensional matrix that corresponds to a grid and each gene of the chromosome corresponds to each cell of the grid.
- Population Initialization — Every gene of a chromosome is assigned a random number between 0 and 399.
- Population Size — We chose 100.
- Selection — Tournament selection with size of 16. Contrary to typical settings, in each competition, the better individual is selected as the winner with probability 0.2 and the worse is selected with probability 0.8. This is rather an anomaly in the perspective of general practice: the better has at least more than 0.5 of chance to win. The traditional rates, more than 0.5 to the better, were not comparable to the final setting. We do not claim or recommend this setting as a general one. It must be highly related to other parts of the GA. In this case, we strongly conjecture that a relatively weak mutation and a strong local optimization affected the anomaly. Rather than correcting the rate to a common-sense one, we decided to accept the situation as a context.
- Crossover — Various crossover operators are used. See Section 4.4.

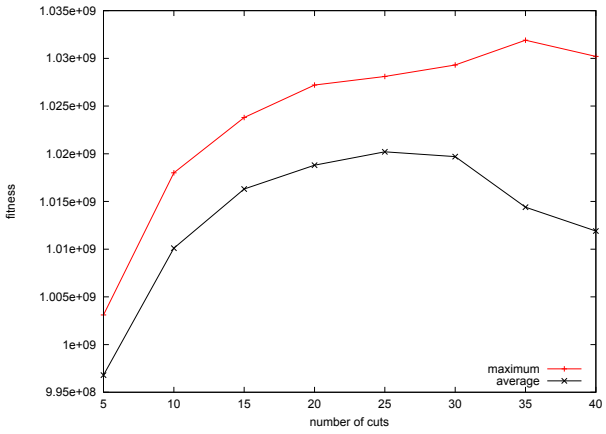

Figure 7: Fitness of GA using geographic crossover with 5 to 40 cuts.

- Mutation — With probability 0.01, each cell of a grid is changed to a random number between 0 and 399.
- Replacement — In a steady-state GA, usually one individual of the population is replaced out for each generation. However our GA replaces two individuals. Two replacement policies are applied simultaneously. First, the worst individual in the population is replaced out. Second, the worse one of the two parents is replaced out.
- Local Search — We developed a BFS-based local search heuristic. Details are described in Section 3.
- Stop Condition — The GA stops when one hundred thousand generations reached.

4.3 The Results

Our GA produced the solution of fitness 10.32×10^8 which won the first prize; Figure 9 shows the solution in detail. After the competition, we eventually improved the solution up to fitness 10.35×10^8; Figure 10 shows the solution in detail. In the best grid, each cell have about 6.32 adjacent cells with positive weight on average, and the average weight is 437,912.

4.4 Crossover

Table 2 shows that solution qualities generated by GAs with a number of different crossovers. Among them, geographic crossover turned out to be the most attractive. Geographic crossover is known to provide diverse recombination of chromosomes and effectively preserve the geographical linkages between genes. If the diversity of recombination is the only important factor, the uniform crossover must be the choice; but it did not perform attractively. The result implies that the diversity of recombination is not the only important factor in crossover.

After choosing geographic crossover, we also observed the performance over various numbers of cuts. Figure 7 shows the result in the range of 5 to 40 and Figure 8 shows the detailed graph of cuts in the range from 20 to 30. The above line shows the maximum fitness of 100 runs and the below shows the average. Among them, 20 to 30 showed similar result and 25 showed the best average result, but 5 to 10

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	171654	0	0	0	0	0	0	0	0	0	765315	0	0	0	0
1	0	0	0	0	0	0	0	726814	548819	0	0	0	0	0	151665	0
2	0	0	0	0	889588	0	660929	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	435241	389017	0	0	0	0	0	0	0	0
4	0	0	898036	0	0	0	0	0	0	0	803512	0	0	0	0	0
5	0	0	0	19413	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	321431	0	0	0
8	0	0	0	0	604908	0	0	0	0	0	0	248494	0	0	0	0
9	0	0	0	0	694318	0	0	0	0	0	0	0	0	0	30414	533000
10	632728	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	141901	0	0	0	17066	0	561301	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	293977	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	426559	0	0	0	523107	0	0
14	0	0	129837	0	0	0	0	0	0	0	0	0	409335	0	594046	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6: Part of weight table.

crossover	number of possible cuts	maximum fitness	average fitness
one-point	$l-1$	9.90×10^8	9.79×10^8
multi-point	$\binom{l-1}{k}$	10.01×10^8	9.92×10^8
Z3	$\binom{2(l-1)}{k}$	10.20×10^8	10.14×10^8
geographic	see Section 2.2	10.28×10^8	10.20×10^8
uniform	2^{l^2-1}	10.03×10^8	09.05×10^8

Table 2: Comparison of crossovers. In this table, $l = m$ is assumed and k denotes the cut size. This shows cut sizes that produce the best average performance. The cut size of multi-point crossover is 5, that of Z3 is 10, and that of geographic crossover is 25.

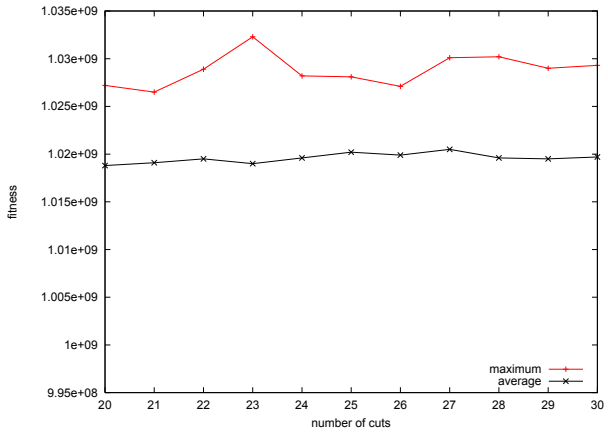

Figure 8: Fitness of GA using geographic crossover with 20 to 30 cuts. (A detailed version of Figure 7)

performed poor. It suggests that sufficiently many cuts are necessary in the context of the GA here. The best number of cuts 25 is rather surprising because it is significantly destructive. If we did not use a strong local optimization, the result would have been different.

4.5 Hybridization

Table 3 shows the effect of hybridization. For fair comparison, each method is run for similar time. The non-hybrid GA and hybrid GA have the same settings except for the former does not include the local search. The input grid of local search is initialized by random numbers in $\{0, 1, \ldots, 399\}$.

Usually the non-hybrid GA did not produce better solution than the local search. Although the local search itself is better than the non-hybrid GA, it is highly probable to get stuck in local optima. Hybridization of the local search and the GA improved the fitness significantly.

method	best	average
non-hybrid GA	7.68×10^8	7.60×10^8
local search	8.79×10^8	8.55×10^8
hybrid GA	10.28×10^8	10.20×10^8

Table 3: Performance of hybrid GA

5. CONCLUDING REMARKS

We presented a hybrid GA for a variant of two-dimensional packing problem that were announced for competition. A BFS-based local search heuristic was proposed and combined with GA. The combination of the local search and the GA improved the fitness significantly. We could observe notable difference in performance among different crossovers; we believe that the difference is due to the abilities to generate diverse new chromosomes and maintaining geographical linkages between genes. Our final choice was two-dimensional geographic crossover. We should also mention that we used a relatively large number of cutting lines in the final geographic crossover. We strongly believe that it has something to do with the low degree of mutation in our setting. Since a strong local optimization algorithm is supporting in every generation, relatively strong perturbation is allowed; so the crossover can try stronger perturbation instead of the weak mutation. We expect that our approach could be also applied to other two-dimensional problems in grid form.

6. ACKNOWLEDGEMENT

We are grateful to Kim Jin Hyun for his useful comments. The ICT at Seoul National University provides research facilities for this study. This work was supported by the Brain Korea 21 Project and the Engineering Research Center of Excellence Program of Korea Ministry of Education, Science and Technology(MEST) / Korea Science and Engineering Foundation(KOSEF), grant number R11-2008-007-2002-0.

129	84	196	293	35	225	2	74	281	161	317	81	272	20	323	35	302	219	99	365
101	115	58	227	38	173	394	30	284	8	180	43	6	282	328	320	347	155	336	370
66	270	301	200	185	364	182	127	79	237	369	94	98	181	278	26	13	238	140	73
276	361	189	296	41	105	303	391	234	280	249	201	103	119	325	33	129	53	261	207
344	245	229	376	234	167	201	156	389	318	77	209	266	238	80	125	38	112	206	81
249	374	367	49	215	68	233	377	236	227	9	245	144	28	56	198	40	324	101	47
22	154	381	262	148	105	350	264	1	252	242	231	333	374	83	396	213	28	113	271
387	130	256	342	348	164	356	355	243	73	334	327	260	143	373	149	395	145	370	261
153	327	83	71	268	161	219	84	221	118	235	243	332	249	174	255	281	241	362	155
244	164	201	87	357	365	286	332	388	339	391	277	208	214	350	112	226	196	133	108
272	140	360	138	148	18	191	159	367	87	291	364	136	93	398	65	195	139	166	126
383	346	157	302	124	152	48	42	116	45	163	202	251	192	86	277	99	114	59	115
168	359	67	203	363	373	19	30	382	172	218	61	77	352	381	379	137	305	361	22
379	341	235	27	229	338	312	114	282	29	82	32	269	74	37	392	143	36	14	372
30	94	49	290	70	76	106	44	24	18	59	69	306	90	109	243	326	177	328	56
280	140	249	113	196	159	3	274	214	375	135	228	127	93	362	397	180	257	348	61
264	12	72	285	138	268	254	258	173	360	310	40	319	144	384	188	174	117	146	135
142	389	230	346	244	55	142	314	248	383	294	179	279	92	211	27	220	178	17	219
393	65	4	2	91	129	329	337	182	51	38	333	380	81	394	224	331	363	345	321
291	210	273	194	294	376	287	213	117	347	393	184	351	11	16	253	259	128	153	247

Figure 9: The grid that won the competition; its fitness is 1,032,295,097**.**

322	389	245	365	286	216	148	127	79	137	90	37	238	82	115	361	95	28	113	271
377	222	43	161	226	31	319	22	292	10	148	74	338	58	59	301	78	324	101	370
383	378	317	196	1	236	249	372	4	163	87	229	293	114	227	9	352	112	226	297
311	344	203	48	381	264	364	2	242	180	83	367	245	276	361	77	251	307	255	191
36	62	391	392	10	41	394	277	310	243	60	374	332	205	64	189	217	373	48	18
139	118	235	102	105	86	398	286	332	383	206	88	6	94	282	98	45	239	68	214
59	154	73	167	68	350	33	219	38	292	19	51	316	280	30	382	103	385	173	360
182	358	360	201	233	26	355	347	155	129	294	194	123	281	321	212	28	145	261	23
337	267	0	64	105	182	328	1	238	376	91	57	204	121	3	312	15	370	269	338
287	213	81	318	17	245	61	56	200	226	159	225	35	189	283	20	155	340	76	106
351	11	141	178	173	242	144	28	348	268	241	196	81	200	296	387	323	128	38	159
380	75	104	248	94	319	83	71	342	38	293	133	139	188	125	207	22	53	42	116
234	68	259	220	341	49	157	256	333	179	40	251	166	114	278	306	13	274	64	367
215	224	331	363	168	359	140	231	327	294	136	93	163	19	32	320	214	331	311	262
328	67	27	365	383	346	72	12	141	339	214	208	373	188	220	398	179	172	184	115
315	229	203	235	339	386	230	389	3	326	332	249	374	300	7	185	189	243	257	264
363	302	263	150	20	7	65	268	234	159	143	333	260	155	147	270	363	176	356	166
45	347	381	298	272	273	55	142	254	137	379	37	302	310	375	276	329	153	48	55
198	395	182	236	252	143	129	30	34	199	168	183	352	360	135	371	280	159	138	279
396	213	149	228	177	281	19	312	44	24	375	127	30	74	284	61	163	250	193	144

Figure 10: The new best grid of fitness 1,034,734,528**.**

7. REFERENCES

[1] C. A. Anderson, K. F. Jones, and J. Ryan. A two-dimensional genetic algorithm for the Ising problem. *Complex Systems*, 5:327–333, 1991.

[2] T. N. Bui and B.-R. Moon. On multi-dimensional encoding/crossover. In *Sixth International Conference on Genetic Algorithms*, pages 49–56, 1995.

[3] J. P. Cohoon and W. Paris. Genetic placement. In *IEEE International Conference on Computer-Aided Design*, pages 422–425, 1986.

[4] C.-H. Im, H.-K. Jung, and Y.-J. Kim. Hybrid genetic algorithm for electromagnetic topology optimization. *IEEE Transactions on Magnetics*, 39(5):2163–2169, 2003.

[5] A. B. Kahng and B.-R. Moon. Toward more powerful recombinations. In *Sixth International Conference on Genetic Algorithms*, pages 96–103, 1995.

[6] B.-R. Moon, Y.-S. Lee, and C.-K. Kim. GEORG: VLSI circuit partitioner with a new genetic algorithm framework. *Journal of Intelligent Manufacturing*, 9(5):401–412, 1998.

[7] E.-J. Park, Y.-H. Kim, and B.-R. Moon. Genetic search for fixed channel assignment problem with limited bandwidth. In *Genetic and Evolutionary Computation Conference*, pages 1172–1179, 2002.

[8] G. Syswerda. Uniform crossover in genetic algorithms. In *Third International Conference on Genetic Algorithms*, pages 2–9, 1989.