Finding Shortest Reconfigurations Sequences of Independent Sets

Volker Turau and Christoph Weyer

Institute of Telematics, Hamburg University of Technology
Hamburg, Germany
turau@tuhh.de

1 Algorithm

The problem at hand is a classical problem in graph theory: find a path between two given vertices in an undirected graph such that the sum of the weights of its constituent edges is minimized. Since in our case all edge weights are equal to 1 this is equivalent to finding the path with fewest edges. There is a myriad of algorithms available for this problem. The challenge in this case is that the graph (a.k.a. the reconfiguration graph) as is only defined implicitly, i.e., the vertices and edges are not explicitly given but rather determined algorithmicly from the input. In addition this implicit graph is rather large, too large to fit into the main memory of current computers. This situation is very common in many search problems. Hence, search algorithms such as A* are good candidates to solve the given problem. The disadvantage of these algorithms is that they keep all generated vertices in memory. For example A* keeps two lists: *open* and *closed* vertices and all vertices touched before finding a solution end up in one of the two lists. Thus, for graphs with more than 10^{10} vertices this requires at 100 GB to 1 T GB of RAM. This outstrips current off-the-shelf hardware. Therefore memory-bounded heuristic searches such as SMA* or IDA* are a better choice. We use Iterative Deepening A* (IDA*) and adapt it to our needs.

IDA* roughly works as follows: In each iteration it performs a limited horizon depth-first search. The horizon is defined using a heuristic estimate of the length of the path from the current vertex to the target vertex. The horizon is the sum of the current distance and the heuristic estimate. The horizon is increased at the end of each iteration by the smallest value that exceed the last horizon. There is a trade of between the quality of a heuristic and the complexity of evaluating the heuristic. Since for large search problems the heuristic is evaluated extremely often it is necessary to choose a simple heuristic that can be preferably computed in constant time. Since in our case the vertices of the implicit graph correspond to independent sets of the input graph, we used the following simple heuristic to estimate the distance from the current independent set C to the target independent set T: $h(C) := |C \setminus T|$, i.e., the node of vertices that are not yet in the target set. This heuristic is easy to compute. Also it is *consistent*, this guarantees that IDA* finds the shortest path.

The downside of informed search algorithms is that if there is no solution – i.e., start and target vertex are in different connected components – they have to access every vertex of the connected component of the graph that contains the start vertex at least once. This problem is aggravated by the fact, that IDA* increases its horizon gradually. So in case no solution exists the number of operations roughly grows exponentially with the diameter of the graph. Thus, in this case a simpler approach is required. We have used a variant of breadth-first search (BFS) for this purpose, that does not require to explicitly store the visited vertices and evaluates no heuristic.

Since it is not known upfront whether a solution exits we have decided to use both approaches concurrently. After initializing the data structures from the input files we launch two threads. The first one executes our version of the breadth-first search while the second executes our variant of IDA*. Whichever thread first finds a shortest path between start and target vertex or finds out that these vertices lie in in different connected components of the implicit graph ends the program and outputs a solution.

The main challenge in solving shortest path problems for large graphs is to use a data structure that allows to quickly determine the neighbors of a vertex in the implicit graph and to quickly evaluate the heuristic. Since vertices in the implicit graph correspond to independent sets we used an adjacency matrix to represent the input graph. Each row of this matrix consists of n bits (n is the number of vertices of the input graph). Thus, a row is represent as $\lceil n/64 \rceil$ words of size 64. The reason for this choice is that current computers have a word size of 64 bits. A vertex of the implicit graph – an independent set of the input graph – is thus also represented by a bit field consisting of $\lceil n/64 \rceil$ words. Thus, given that X is an independent set, to check whether $X \setminus \{x\} \cup \{y\}$ is also independent can be decided by at most $\lceil n/64 \rceil$ XOR operations. We can update independent sets in constant time. Similarly, we can for a given independent set compute its neighboring independent sets in time $O(n)$ time. Moreover, very efficient low level functions for this type of representation are available, e.g., *__builtin_ffsll* for computing the index of the first bit set. At no time we have to iterate over the neighborhood of a vertex.

To cope with the large number of vertices we used different techniques. In BFS we represented the vertices of every layer in a separate AVL tree. We maintain at any time only the top two layers of the BFS tree. The predecessor relation is maintained with reference counters. At the completion of each layer, lower layers are thinned out by recursively removing vertices with no successor, i.e., reference counter 0. Vertices in the AVL trees correspond to independent sets. In order to speed up the search in such trees we additionally stored a 64 Bit hash value with each independent set. Thus, first we compared the hash values and only if these coincided we needed to explicitly compare the independent sets. Therefore, in most cases a comparison only required one operation.

We implemented a variant of IDA*. To speed up this algorithm we maintained a compact history between individual iterations. This history includes information at what distance from the start vertex a vertex appeared. If it reappears at the same or a larger distance it can be skipped. At the beginning of a new iteration the history was *lifted*, i.e., the distance of each vertex is incremented by 1, so that we search beyond the previous horizon. This helps to considerably reduce the number of vertices that had to be revisited. Also, since this history is not required for the algorithm to work correctly, we can at any time close the history to limit the usage of RAM. The history is implemented as hash table using open addressing, in particular we used linear probing, the hash function *MurmurHash2* yielding hash values of size 64 bit, and a load factor of 5/8. We also designed the data structures such as AVL trees in a way that allows to efficiently allocate and deallocate the required memory.

The algorithm is to a large degree a general search algorithm. It can use the TS (token sliding) or the TJ (token jumping) model. Some of the problem specific aspects are as follows.

- The estimate function $h(C)$, it can be computed in constant time given the value of the previous state.
- The successors of a vertex in the search graph can be found very efficiently by maintaining for each node the number of neighbors that are in the current independent set. There are only two cases. If a node has no such neighbor, then every token can be placed on it. If it has a single such neighbor then the token of this neighbor can be placed on it.
- Representing the graph as a bitset allowed to implement many basic graph operations with logical operators such as *and, or*, and *xor* on words of length 64 bit.

A simple extension of the algorithm would be to run two more threads that start to search from the target independent set. For the graphs in the benchmark that did not bring any benefit, because they seem to be *symmetric* with respect to start and target. We do not know whether this property holds in general, properly not.

2 Implementation Details

The algorithm is implemented in the programming language C, using c99. As a compiler we used *clang* with options *-Ofast -march=native -mtune=native*. The source code consists of seven modules, in total about 50 KiB of code. The threads are implemented using *pthread*. We do not use any third party software.

3 Computation Environment

We run our benchmarks on the following configuration. We used an Ubuntu 20.04 virtual machine with 4 logical CPUs and 60 GiB RAM. Our program uses

Dat File	n	m	Solution	Time
grid004x004_01.dat	16	24	No	00:00:00.07
grid004x004_02.dat	16	24	No	00:00:00.05
grid004x004_03.dat	16	24	7	00:00:00.05
grid004x004_04.dat	16	24	8	00:00:00.05
grid004x004_05.dat	16	24	No	00:00:00.07
hc-power-11_01.dat	21	28	21	00:00:00.06
hc-power-12_01.dat	36	51	69	00:00:00.05
hc-square-01_01.dat	14	18	12	00:00:00.05
hc-square-02_01.dat	24	33	30	00:00:00.05
hc-toyno-01_01.dat	6	9	No	00:00:00.05
hc-toyyes-01_01.dat	7	7	3	00:00:00.05

Table 1: Solutions for first benchmark

only 2 CPUs. The server uses Proxmox as virtualization environment. The server has an AMD Ryzen9 3900X CPU with 12 cores at 3.80 GHz. For storage we use NVMe SSDs.

4 Solutions

In the following tables we summarize the outcome of our program. The corresponding output files can be found in die `solutions` directory. The column solution specifies if no path can be found or the number of steps for the shortest path. The time column is the measured elapsed time from the Unix tool `time` as `HH:MM:SS.mm`. If a program runs longer than one hour no milliseconds are given.

When we found no solution for the given problem due to time constraints a — is shown.

Dat File	n	m	Solution	Time
grid010x010_01_6141.dat	100	180	No	00:00:00.05
grid010x010_02_0495.dat	100	180	No	00:00:00.05
grid010x010_03_6844.dat	100	180	No	00:00:00.05
grid010x010_04_4957.dat	100	180	No	00:00:00.08
grid020x020_01_9275.dat	400	760	No	00:00:01.40
grid020x020_02_7550.dat	400	760	No	00:00:00.07
grid020x020_03_3902.dat	400	760	No	00:00:01.44
grid020x020_04_3283.dat	400	760	No	00:00:00.05
grid030x030_01_6931.dat	900	1740	No	00:00:00.05
grid030x030_02_7112.dat	900	1740	No	00:00:21.81
grid030x030_03_4248.dat	900	1740	No	00:00:00.05
grid030x030_04_2988.dat	900	1740	No	00:00:22.17
grid040x040_01_6703.dat	1600	3120	No	00:00:00.13
grid040x040_02_3552.dat	1600	3120	No	00:00:00.09
grid040x040_03_0746.dat	1600	3120	No	00:02:29.34
grid040x040_04_3680.dat	1600	3120	No	00:02:35.90
grid050x050_01_8271.dat	2500	4900	No	00:00:00.31
grid050x050_02_8314.dat	2500	4900	No	00:14:18.34
grid050x050_03_4298.dat	2500	4900	No	00:00:00.29
grid050x050_04_8232.dat	2500	4900	No	00:14:16.52
grid060x060_01_5912.dat	3600	7080	No	00:00:00.62
grid060x060_02_4571.dat	3600	7080	No	00:57:59.71
grid060x060_03_2183.dat	3600	7080	No	00:57:30.17
grid060x060_04_3263.dat	3600	7080	No	00:00:00.63
grid070x070_01_6558.dat	4900	9660	No	00:00:01.41
grid070x070_02_7903.dat	4900	9660	No	02:43:26
grid070x070_03_4857.dat	4900	9660	No	02:45:27
grid070x070_04_3525.dat	4900	9660	No	00:00:01.35
grid080x080_01_9321.dat	6400	12640	No	00:00:02.76
grid080x080_02_8623.dat	6400	12640	No	06:50:57
grid080x080_03_8653.dat	6400	12640	No	00:00:02.77
grid080x080_04_8592.dat	6400	12640	No	06:55:10
grid090x090_01_0710.dat	8100	16020	No	18:55:31
grid090x090_02_9631.dat	8100	16020	No	00:00:07.58
grid090x090_03_8778.dat	8100	16020	No	19:37:18
grid090x090_04_4969.dat	8100	16020	No	00:00:07.56
grid100x100_01_9072.dat	10000	19800	No	38:47:16
grid100x100_02_7191.dat	10000	19800	No	00:00:11.73
grid100x100_03_4887.dat	10000	19800	No	39:08:22
grid100x100_04_0664.dat	10000	19800	No	00:00:11.96
grid200x200_01_8265.dat	40000	79600	—	—
grid200x200_02_2458.dat	40000	79600	No	00:11:13.24
grid200x200_03_0743.dat	40000	79600	No	00:11:09.19
grid200x200_04_2816.dat	40000	79600	—	—

Table 2: Solutions for grid benchmarks

Dat File	n	m	Solution	Time
queen008x008_01_4136.dat	64	728	9	00:00:00.13
queen008x008_02_8804.dat	64	728	11	00:00:00.05
queen008x008_03_8723.dat	64	728	11	00:00:00.05
queen008x008_04_4441.dat	64	728	12	00:00:00.05
queen010x010_01_0612.dat	100	1470	10	00:00:00.05
queen010x010_02_5832.dat	100	1470	11	00:00:00.05
queen010x010_03_7635.dat	100	1470	13	00:00:00.05
queen010x010_04_2943.dat	100	1470	14	00:00:00.05
queen020x020_01_7075.dat	400	12540	24	00:00:00.57
queen020x020_02_4815.dat	400	12540	22	00:00:00.31
queen020x020_03_8667.dat	400	12540	21	00:00:00.38
queen020x020_04_0961.dat	400	12540	18	00:00:00.25
queen030x030_01_3761.dat	900	43210	15	00:00:00.13
queen030x030_02_4963.dat	900	43210	26	00:00:03.61
queen030x030_03_7394.dat	900	43210	26	00:00:02.41
queen030x030_04_9144.dat	900	43210	24	00:00:00.66
queen040x040_01_6291.dat	1600	103480	13	00:00:00.08
queen040x040_02_9487.dat	1600	103480	14	00:00:00.14
queen040x040_03_7037.dat	1600	103480	14	00:00:00.12
queen040x040_04_8369.dat	1600	103480	17	00:00:00.14
queen050x050_01_6006.dat	2500	203350	13	00:00:00.12
queen050x050_02_6166.dat	2500	203350	16	00:00:00.12
queen050x050_03_0562.dat	2500	203350	6	00:00:00.08
queen050x050_04_9047.dat	2500	203350	12	00:00:00.10
queen060x060_01_8336.dat	3600	352820	5	00:00:00.12
queen060x060_02_6710.dat	3600	352820	9	00:00:00.12
queen060x060_03_1394.dat	3600	352820	9	00:00:00.10
queen060x060_04_3799.dat	3600	352820	9	00:00:00.11
queen070x070_01_1828.dat	4900	561890	38	04:21:08
queen070x070_02_8683.dat	4900	561890	40	00:38:16.40
queen070x070_03_1545.dat	4900	561890	34	00:01:23.21
queen070x070_04_7426.dat	4900	561890	33	00:01:06.70
queen080x080_01_0297.dat	6400	840560	25	00:00:10.44
queen080x080_02_2368.dat	6400	840560	31	00:05:34.91
queen080x080_03_0355.dat	6400	840560	—	—
queen080x080_04_4954.dat	6400	840560	36	00:10:46.40
queen090x090_01_8978.dat	8100	1198830	—	—
queen090x090_02_0808.dat	8100	1198830	41	02:06:41
queen090x090_03_9570.dat	8100	1198830	—	—
queen090x090_04_1188.dat	8100	1198830	38	00:05:02.33
queen100x100_01_3955.dat	10000	1646700	—	—
queen100x100_02_1699.dat	10000	1646700	6	00:00:00.38
queen100x100_03_6670.dat	10000	1646700	—	—
queen100x100_04_0618.dat	10000	1646700	—	—
queen200x200_01_0761.dat	40000	13253400	4	00:00:02.09
queen200x200_02_3057.dat	40000	13253400	29	00:00:10.10
queen200x200_03_0327.dat	40000	13253400	29	00:00:10.08
queen200x200_04_4796.dat	40000	13253400	27	00:00:03.06

Table 3: Solutions for queen benchmarks

Dat File	n	m	Solution	Time
hc-square-004-002_01.dat	44	63	90	00:00:00.05
hc-square-005-002_01.dat	54	78	132	00:00:00.05
hc-square-006-002_01.dat	64	93	182	00:00:00.07
hc-square-007-002_01.dat	74	108	240	00:00:00.49
hc-square-008-002_01.dat	84	123	306	00:00:03.39
hc-square-009-002_01.dat	94	138	380	00:00:25.28
hc-square-010-002_01.dat	104	153	462	00:03:04.50
hc-square-011-002_01.dat	114	168	552	00:24:20.14
hc-square-012-002_01.dat	124	183	650	03:06:18
hc-square-013-002_01.dat	134	198	756	23:08:27
hc-square-014-002_01.dat	144	213	—	—
hc-square-015-002_01.dat	154	228	—	—
hc-square-016-002_01.dat	164	243	—	—
hc-square-017-002_01.dat	174	258	—	—
hc-square-018-002_01.dat	184	273	—	—
hc-square-019-002_01.dat	194	288	—	—
hc-square-020-002_01.dat	204	303	—	—

Table 4: Solutions for square benchmarks

Dat File	n	m	Solution	Time
hc-power-004-002_01.dat	64	95	359	00:00:00.07
hc-power-005-002_01.dat	79	118	779	00:00:00.21
hc-power-006-002_01.dat	94	141	1631	00:00:02.24
hc-power-007-002_01.dat	109	164	3347	00:00:25.03
hc-power-008-002_01.dat	124	187	6791	00:04:39.50
hc-power-009-002_01.dat	139	210	13691	00:58:42.48
hc-power-010-002_01.dat	154	233	27503	11:00:38
hc-power-011-002_01.dat	169	256	—	—
hc-power-012-002_01.dat	184	279	—	—
hc-power-013-002_01.dat	199	302	—	—
hc-power-014-002_01.dat	214	325	—	—
hc-power-015-002_01.dat	229	348	—	—
hc-power-016-002_01.dat	244	371	—	—
hc-power-017-002_01.dat	259	394	—	—
hc-power-018-002_01.dat	274	417	—	—
hc-power-019-002_01.dat	289	440	—	—
hc-power-020-002_01.dat	304	463	—	—

Table 5: Solutions for power benchmarks

Dat File	n	m	Solution	Time
sp001_01.dat	13	66	11	00:00:00.05
sp002_01.dat	26	150	33	00:00:00.05
sp003_01.dat	39	234	77	00:00:00.05
sp004_01.dat	52	318	165	00:00:00.05
sp005_01.dat	65	402	341	00:00:00.05
sp006_01.dat	78	486	693	00:00:00.20
sp007_01.dat	91	570	1397	00:00:01.77
sp008_01.dat	104	654	2805	00:00:15.77
sp009_01.dat	117	738	5621	00:02:27.31
sp010_01.dat	130	822	11253	00:24:02.35
sp011_01.dat	143	906	22517	04:01:44
sp012_01.dat	156	990	—	—
sp013_01.dat	169	1074	—	—
sp014_01.dat	182	1158	—	—
sp015_01.dat	195	1242	—	—
sp016_01.dat	208	1326	—	—
sp017_01.dat	221	1410	—	—
sp018_01.dat	234	1494	—	—
sp019_01.dat	247	1578	—	—
sp020_01.dat	260	1662	—	—
sp021_01.dat	273	1746	—	—
sp022_01.dat	286	1830	—	—
sp023_01.dat	299	1914	—	—
sp024_01.dat	312	1998	—	—
sp025_01.dat	325	2082	—	—
sp026_01.dat	338	2166	—	—
sp027_01.dat	351	2250	—	—
sp028_01.dat	364	2334	—	—
sp029_01.dat	377	2418	—	—
sp030_01.dat	390	2502	—	—

Table 6: Solutions for SP benchmarks

Dat File	n	m	Solution	Time
1-FullIns_3_01.dat	30	100	1	00:00:00.06
1-FullIns_3_02.dat	30	100	3	00:00:00.06
1-FullIns_4_01.dat	93	593	1	00:00:00.05
1-FullIns_4_02.dat	93	593	1	00:00:00.05
1-FullIns_5_01.dat	282	3247	1	00:00:00.07
1-FullIns_5_02.dat	282	3247	1	00:00:00.06
1-Insertions_4_01.dat	67	232	3	00:00:00.05
1-Insertions_4_02.dat	67	232	2	00:00:00.05
1-Insertions_5_01.dat	202	1227	2	00:00:00.06
1-Insertions_5_02.dat	202	1227	2	00:00:00.05
1-Insertions_6_01.dat	607	6337	1	00:00:00.07
1-Insertions_6_02.dat	607	6337	3	00:00:00.05
2-FullIns_3_01.dat	52	201	2	00:00:00.05
2-FullIns_3_02.dat	52	201	2	00:00:00.05
2-FullIns_4_01.dat	212	1621	1	00:00:00.05
2-FullIns_4_02.dat	212	1621	1	00:00:00.05
2-FullIns_5_01.dat	852	12201	1	00:00:00.06
2-FullIns_5_02.dat	852	12201	1	00:00:00.05
2-Insertions_3_01.dat	37	72	3	00:00:00.06
2-Insertions_3_02.dat	37	72	2	00:00:00.05
2-Insertions_4_01.dat	149	541	1	00:00:00.05
2-Insertions_4_02.dat	149	541	1	00:00:00.05
2-Insertions_5_01.dat	597	3936	1	00:00:00.06
2-Insertions_5_02.dat	597	3936	1	00:00:00.05
3-FullIns_3_01.dat	80	346	2	00:00:00.05
3-FullIns_3_02.dat	80	346	4	00:00:00.06
3-FullIns_4_01.dat	405	3524	2	00:00:00.05
3-FullIns_4_02.dat	405	3524	3	00:00:00.05
3-FullIns_5_01.dat	2030	33751	1	00:00:00.06
3-FullIns_5_02.dat	2030	33751	5	00:00:00.06
3-Insertions_3_01.dat	56	110	4	00:00:00.05
3-Insertions_3_02.dat	56	110	2	00:00:00.05
3-Insertions_4_01.dat	281	1046	2	00:00:00.05
3-Insertions_4_02.dat	281	1046	5	00:00:00.05
3-Insertions_5_01.dat	1406	9695	1	00:00:00.08
3-Insertions_5_02.dat	1406	9695	1	00:00:00.05
4-FullIns_3_01.dat	114	541	1	00:00:00.05
4-FullIns_3_02.dat	114	541	1	00:00:00.05
4-FullIns_4_01.dat	690	6650	1	00:00:00.05
4-FullIns_4_02.dat	690	6650	2	00:00:00.05
4-FullIns_5_01.dat	4146	77305	1	00:00:00.07
4-FullIns_5_02.dat	4146	77305	1	00:00:00.07
4-Insertions_3_01.dat	79	156	1	00:00:00.05
4-Insertions_3_02.dat	79	156	3	00:00:00.05
4-Insertions_4_01.dat	475	1795	1	00:00:00.05
4-Insertions_4_02.dat	475	1795	1	00:00:00.05
5-FullIns_3_01.dat	154	792	1	00:00:00.05
5-FullIns_3_02.dat	154	792	5	00:00:00.05
5-FullIns_4_01.dat	1085	11395	1	00:00:00.07
5-FullIns_4_02.dat	1085	11395	6	00:00:00.05

Table 7: Solutions for color benchmarks (part 1)

Dat File	n	m	Solution	Time
DSJC1000.1_01.dat	1000	49629	1	00:00:00.06
DSJC1000.1_02.dat	1000	49629	3	00:00:00.06
DSJC1000.5_01.dat	1000	249826	6	00:00:00.10
DSJC1000.5_02.dat	1000	249826	2	00:00:00.08
DSJC1000.9_01.dat	1000	449449	2	00:00:00.11
DSJC1000.9_02.dat	1000	449449	2	00:00:00.11
DSJC125.1_01.dat	125	736	3	00:00:00.06
DSJC125.1_02.dat	125	736	5	00:00:00.05
DSJC125.5_01.dat	125	3891	4	00:00:00.06
DSJC125.5_02.dat	125	3891	2	00:00:00.05
DSJC125.9_01.dat	125	6961	1	00:00:00.05
DSJC125.9_02.dat	125	6961	8	00:00:00.05
DSJC250.1_01.dat	250	3218	1	00:00:00.05
DSJC250.1_02.dat	250	3218	1	00:00:00.05
DSJC250.5_01.dat	250	15668	1	00:00:00.05
DSJC250.5_02.dat	250	15668	5	00:00:00.05
DSJC250.9_01.dat	250	27897	5	00:00:00.22
DSJC250.9_02.dat	250	27897	5	00:00:00.05
DSJC500.1_01.dat	500	12458	3	00:00:00.05
DSJC500.1_02.dat	500	12458	6	00:00:00.05
DSJC500.5_01.dat	500	62624	1	00:00:00.07
DSJC500.5_02.dat	500	62624	1	00:00:00.06
DSJC500.9_01.dat	500	112437	6	00:00:00.08
DSJC500.9_02.dat	500	112437	9	00:00:00.06
DSJR500.1_01.dat	500	3555	8	00:00:00.05
DSJR500.1_02.dat	500	3555	13	00:00:00.05
DSJR500.1c_01.dat	500	121275	1	00:00:00.10
DSJR500.1c_02.dat	500	121275	4	00:00:00.06
DSJR500.5_01.dat	500	58862	2	00:00:00.06
DSJR500.5_02.dat	500	58862	6	00:00:00.07
anna_01.dat	138	493	5	00:00:00.05
anna_02.dat	138	493	13	00:00:00.05
ash331GPIA_01.dat	662	4181	1	00:00:00.05
ash331GPIA_02.dat	662	4181	1	00:00:00.06
ash608GPIA_01.dat	1216	7844	1	00:00:00.05
ash608GPIA_02.dat	1216	7844	1	00:00:00.05
ash958GPIA_01.dat	1916	12506	1	00:00:00.08
ash958GPIA_02.dat	1916	12506	1	00:00:00.06
david_01.dat	87	406	6	00:00:00.05
david_02.dat	87	406	9	00:00:00.05
games120_01.dat	120	638	7	00:00:00.05
games120_02.dat	120	638	13	00:00:00.05
huck_01.dat	74	301	7	00:00:00.05
huck_02.dat	74	301	7	00:00:00.05
latin_square_10_01.dat	900	307350	1	00:00:00.28
latin_square_10_02.dat	900	307350	1	00:00:00.09

Table 8: Solutions for color benchmarks (part 2)

Dat File	n	m	Solution	Time
le450_15a_01.dat	450	8168	3	00:00:00.06
le450_15a_02.dat	450	8168	3	00:00:00.05
le450_15b_01.dat	450	8169	2	00:00:00.05
le450_15b_02.dat	450	8169	5	00:00:00.05
le450_15c_01.dat	450	16680	2	00:00:00.05
le450_15c_02.dat	450	16680	3	00:00:00.05
le450_15d_01.dat	450	16750	2	00:00:00.05
le450_15d_02.dat	450	16750	6	00:00:00.05
le450_25a_01.dat	450	8260	3	00:00:00.07
le450_25a_02.dat	450	8260	12	00:00:00.05
le450_25b_01.dat	450	8263	6	00:00:00.05
le450_25b_02.dat	450	8263	10	00:00:00.05
le450_25c_01.dat	450	17343	3	00:00:00.07
le450_25c_02.dat	450	17343	5	00:00:00.05
le450_25d_01.dat	450	17425	3	00:00:00.05
le450_25d_02.dat	450	17425	2	00:00:00.05
le450_5a_01.dat	450	5714	1	00:00:00.05
le450_5a_02.dat	450	5714	1	00:00:00.05
le450_5b_01.dat	450	5734	1	00:00:00.06
le450_5b_02.dat	450	5734	2	00:00:00.05
le450_5c_01.dat	450	9803	1	00:00:00.05
le450_5c_02.dat	450	9803	1	00:00:00.05
le450_5d_01.dat	450	9757	1	00:00:00.05
le450_5d_02.dat	450	9757	1	00:00:00.05
miles1000_01.dat	128	3216	3	00:00:00.05
miles1000_02.dat	128	3216	5	00:00:00.05
miles1500_01.dat	128	5198	2	00:00:00.05
miles1500_02.dat	128	5198	3	00:00:00.05
miles500_01.dat	128	1170	7	00:00:00.05
miles500_02.dat	128	1170	10	00:00:00.05
miles750_01.dat	128	2113	4	00:00:00.05
miles750_02.dat	128	2113	6	00:00:00.05
mug100_1_01.dat	100	166	11	00:00:00.05
mug100_1_02.dat	100	166	18	00:00:00.05
mug100_25_01.dat	100	166	15	00:00:00.05
mug100_25_02.dat	100	166	21	00:00:00.05
mug88_1_01.dat	88	146	9	00:00:00.05
mug88_1_02.dat	88	146	20	00:00:00.05
mug88_25_01.dat	88	146	7	00:00:00.05
mug88_25_02.dat	88	146	19	00:00:00.05
myciel3_01.dat	11	20	2	00:00:00.05
myciel3_02.dat	11	20	3	00:00:00.05
myciel4_01.dat	23	71	1	00:00:00.05
myciel4_02.dat	23	71	1	00:00:00.05
myciel5_01.dat	47	236	1	00:00:00.05
myciel5_02.dat	47	236	1	00:00:00.05
myciel6_01.dat	95	755	1	00:00:00.05
myciel6_02.dat	95	755	1	00:00:00.05
myciel7_01.dat	191	2360	1	00:00:00.05
myciel7_02.dat	191	2360	1	00:00:00.05

Table 9: Solutions for color benchmarks (part 3)

Dat File	n	m	Solution	Time
qg.order100_01.dat	10000	990000	97	00:00:00.26
qg.order100_02.dat	10000	990000	98	00:00:00.32
qg.order30_01.dat	900	26100	30	00:00:00.08
qg.order30_02.dat	900	26100	29	00:00:00.06
qg.order40_01.dat	1600	62400	40	00:00:00.09
qg.order40_02.dat	1600	62400	39	00:00:00.06
qg.order60_01.dat	3600	212400	59	00:00:00.09
qg.order60_02.dat	3600	212400	58	00:00:00.08
queen10_10_01.dat	100	1470	8	00:00:00.06
queen10_10_02.dat	100	1470	11	00:00:00.05
queen11_11_01.dat	121	1980	8	00:00:00.05
queen11_11_02.dat	121	1980	8	00:00:00.05
queen12_12_01.dat	144	2596	6	00:00:00.05
queen12_12_02.dat	144	2596	9	00:00:00.05
queen13_13_01.dat	169	3328	12	00:00:00.08
queen13_13_02.dat	169	3328	4	00:00:00.05
queen14_14_01.dat	196	4186	9	00:00:00.05
queen14_14_02.dat	196	4186	8	00:00:00.05
queen15_15_01.dat	225	5180	5	00:00:00.05
queen15_15_02.dat	225	5180	4	00:00:00.05
queen16_16_01.dat	256	6320	5	00:00:00.05
queen16_16_02.dat	256	6320	7	00:00:00.05
queen5_5_01.dat	25	160	7	00:00:00.05
queen5_5_02.dat	25	160	7	00:00:00.05
queen6_6_01.dat	36	290	8	00:00:00.05
queen6_6_02.dat	36	290	9	00:00:00.05
queen7_7_01.dat	49	476	11	00:00:00.05
queen7_7_02.dat	49	476	8	00:00:00.05
queen8_12_01.dat	96	1368	10	00:00:00.08
queen8_12_02.dat	96	1368	6	00:00:00.05
queen8_8_01.dat	64	728	9	00:00:00.07
queen8_8_02.dat	64	728	9	00:00:00.05
queen9_9_01.dat	81	1056	7	00:00:00.05
queen9_9_02.dat	81	1056	5	00:00:00.05
school1_01.dat	385	19095	4	00:00:00.06
school1_02.dat	385	19095	11	00:00:00.05
school1_nsh_01.dat	352	14612	4	00:00:00.07
school1_nsh_02.dat	352	14612	8	00:00:00.05
wap01a_01.dat	2368	110871	22	00:00:00.07
wap01a_02.dat	2368	110871	76	00:00:00.07
wap02a_01.dat	2464	111742	25	00:00:00.08
wap02a_02.dat	2464	111742	97	00:00:00.08
wap03a_01.dat	4730	286722	10	00:00:00.10
wap03a_02.dat	4730	286722	96	00:00:00.11
wap04a_01.dat	5231	294902	21	00:00:00.10
wap04a_02.dat	5231	294902	112	00:00:00.11
wap05a_01.dat	905	43081	16	00:00:00.07
wap05a_02.dat	905	43081	26	00:00:00.06
wap06a_01.dat	947	43571	17	00:00:00.07
wap06a_02.dat	947	43571	36	00:00:00.06
wap07a_01.dat	1809	103368	9	00:00:00.08
wap07a_02.dat	1809	103368	36	00:00:00.07
wap08a_01.dat	1870	104176	13	00:00:00.07
wap08a_02.dat	1870	104176	43	00:00:00.07
will199GPIA_01.dat	701	6772	1	00:00:00.05
will199GPIA_02.dat	701	6772	3	00:00:00.05

Table 10: Solutions for color benchmarks (part 4)