Automatically Generating and Solving Eternity II Style Puzzles

Harris, Geoffrey; Vanstone, Bruce J; Gepp, Adrian

Published in:
Recent Trends and Future Technology in Applied Intelligence - 31st International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2018, Proceedings

DOI:
10.1007/978-3-319-92058-0_60

Licence:
Other

Automatically Generating and Solving Eternity II Style Puzzles

Geoff Harris[1][0000−0003−4284−8619], Bruce James Vanstone[1][0000−0002−3977−2468], and Adrian Gepp[1][0000−0003−1666−5501]

Bond Business School, Bond University, Gold Coast, Australia.
gharris@bond.edu.au, bvanston@bond.edu.au, adgepp@bond.edu.au

Abstract. The Eternity II puzzle is an NP-complete problem. Prior researchers have generated data sets that are similar to the Eternity II problem. These data sets can be created in linear time, but this comes at the cost of easing the problem by introducing exploitable statistical features. The first contribution of this paper is a new method to generate data sets that are truly of Eternity II style. The second contribution is an Eternity II specific implementation of a constraint-satisfaction-problem style algorithm. Unlike most other published algorithms, this one has no form of look-ahead, filtering, forward checking, back jumping or k-consistency checks. Instead, it uses knowledge about the structure of the puzzle and the uniform distribution of edge colours. This approach is up to three orders of magnitude faster than previously published attempts.

Keywords: Eternity II, edge matching puzzle, constraint satisfaction problem, NP-complete

1 Introduction

Edge Matching Puzzles (EMPs) belong to the NP-complete (NP-C) problem set for their worst-case complexity [1] and, depending on the objective function chosen, optimization of those solutions may result in an NP-hard problem. With the release of two high prize money puzzles, labelled Eternity and Eternity II, EMPs have attracted the attention of academics from a range of disciplines. The first Eternity puzzle was solved by exploiting statistical features (non-uniformities) in the distribution of piece colours. This weakness was corrected in Eternity II, which remains an open problem.

The contribution of this paper is two-fold. First, we develop an efficient algorithm to generate data sets that are truly of Eternity II style puzzle. Second, we present a naïve algorithm for solving such puzzles, which has no look-ahead, no forward checking, no back jumping and no k-consistency checks. Empirical tests reveal that our algorithm outperforms all previously published results.

2 Literature Review

Ansótegui et al. [2] provided a theoretical framework of the generalized EMP, established some terminology with mathematical definitions and provided both

elementary and advanced SAT and CSP solution algorithms. However, to ensure runtimes were feasible the authors only considered puzzles of edge size $n \in \{6, 7, 8\}$. Due to the combinatorial explosion with increasing n, they could not attempt the actual commercial puzzle of size $n = 16$.

The first published result of the commercial n=16 problem used a hybrid approach of Tabu search and local neighbourhood techniques to claim a score of 458/480 [3]. (The standard score is calculated from the number of edges that match.) However, the researchers noted that with their implementation the largest puzzle that they could solve was $n = 8$.

In a more practical CSP approach to the Eternity II puzzle, researchers have considered the constrained enumeration to all solutions of the problem [4]. Their reinforced filtering of the data structures enabled them to efficiently solve some EMP instances of size $n = 10$. However, these EMP puzzles were substantially easier to solve than the Eternity II style $n = 10$ puzzles.

An evolutionary algorithm approach only achieved 396/480 on the $n = 16$ commercial Eternity II puzzle [5]. Despite the poor score, the significance of this work is that three separate evolutionary algorithms were evaluated.

In another attempt of the general problem, a Tabu search algorithm was used as a two-phase divide and conquer technique [6]. They achieved a best score of 418/480 on the $n = 16$ puzzle at the 2010 International Conference on Metaheuristics and Nature Inspired Computing (META'10). Using a conceptually similar guided hyper-heuristic, the winners of the META'10 achieved a substantially higher best score of 461/480. However, both were unable to solve any puzzle of size $n > 8$.

The above quoted results need to be viewed in light of the academically unpublished, publicly available results of Louis Verhaard [7] who obtained a score of 467/480 for the commercial Eternity II puzzle.

3 Generation of Eternity II Style Puzzle Datasets

The frequency distribution of edge colours of the commercial Eternity II puzzle is shown in the left panel of Fig. 1. The distributions of inner and frame colours are uniform (to maximize entropy) In addition, (i) tiles are not permitted to be rotated and (ii) globally symmetric patterns of tiles are not allowed. These conditions ensure unique (non-degenerate) solutions that are a necessary requirement for a puzzle to be classed as an Eternity II style puzzle.

In the puzzle generation algorithm described by Ansótegui et al. [2], a random number generator is used to create the colour distribution on the tile edges. Consequently, their results are for a combination of Eternity II style puzzles and other EMP problems, which they refer to as GEMP (General Edge Matching Puzzles). In stark contrast to the left panel of Figure 1, consider the colour distribution shown in the right panel for one of the easier $n = 10$ puzzles [4], which has many statistical weaknesses that can be exploited by algorithms with domain trimming heuristics. That is, it is reasonable to expect the use of look-ahead, forward checking, back jumping and k-consistency checks should result

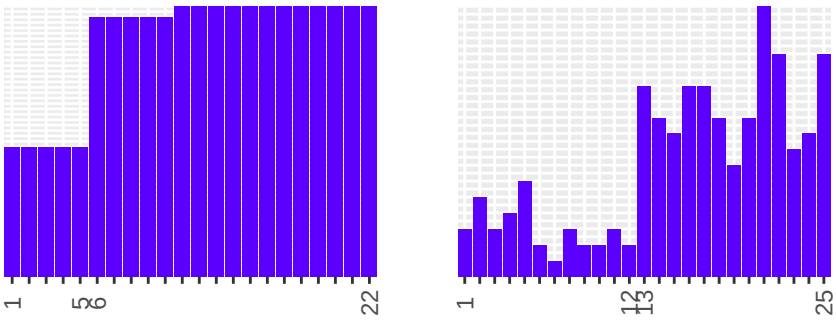

Fig. 1. The distribution of frame and inner-edge colours in the commercial Eternity II puzzle (left) contrasted with a typical 10×10 EMP [4] (right). For Eternity II, the frame colours are numbered 1 to 5 and the inner colours are numbered 6 to 22. For the typical EMP, the frame colours are numbered 1 to 12 and the inner colours are numbered 13 to 25.

in exploitation by those algorithms to dramatically reduce the size of the search space. Unlike the methods above that can result in EMP problems that do not meet the requirements of an Eternity II style puzzle, we propose an algorithm that always generates an $n \times n$ Eternity II style puzzle with $N = n^2$ tiles consisting of i unique frame colours and j unique inner colours where $f(i) \leq f(j) \div 2$:

```
Generate a list of 4(n-1) colors from a uniform range of [1..i]
           colors
Generate a list of 2(n-1)(n-2) colors from a uniform range of
           [1..j] colors
Generate N blank tiles on an n by n grid
Repeat
  Randomly assign the frame colors to adjacent edges of the
           frame pieces
  Randomly assign the inner colors to adjacent edges of the
           inner pieces
  Validate tiles for rotation and symmetry
Until All N tiles are validated.
```

Any puzzle generated by this algorithm will fulfil the necessary and sufficient conditions to be classed an Eternity II style hard EMP.

4 Our Zero Look-ahead Algorithm (ZLA)

A brute force algorithm that does not employ look-ahead simply attempts to instantiate the next variable available. However, given the assumption $P \neq NP$

[8], we need a method of variable instantiation which will reasonably trim the branching factor of the implemented search tree that incorporates the specifics of Eternity II style puzzles. The smallest domains from the root node will be those associated with the corners as there are only four elements possible for each domain. Thus, our ZLA begins by instantiating a single corner piece, which is arbitrarily chosen as the top left corner of the puzzle. It is self-evident that all solutions (after removing those with rotational redundancy) to the entire puzzle necessarily have the same corner piece in that position. Ipso facto, our first domain can be created as a single corner piece.

For reasonable sized puzzles, as a potential constituent if this were a look-ahead approach, the next smallest domain will be a tie between the other three corner variables. However, our ZLA works on the application of instantiating the most logically constrained of the neighbouring variables. The next variable to instantiate is thus one of the two frame positions adjacent to the top left corner. As the Eternity II style puzzles have uniform colour distributions, there will be a tie between these two variables. Thus, without any computational loss, we can choose to instantiate the variable to the right of the top left corner. Once that variable has been instantiated, we again utilise knowledge of the puzzles properties to realize that the shortest path going forward is to work down the rows, using a zigzag pattern across and back pairs of rows.

The elements of each domain are accessed via a global indexing structure which connects domains to rotations of pieces by the top edge of each rotated piece with one of the sides. There are no look-ahead, forward check, back jumping or k-consistency checks performed by the algorithm. Other than using an index to ensure it only considers pieces that could be instantiated into a variable, the only runtime check performed is whether that piece is already in use or not.

5 Results and Discussion

Our ZLA algorithm was implemented in Pascal using bit sets to speed up comparison operations. The source code (and all datasets used for the results) have been uploaded to Bitbucket [9]. The implementation was tested against published benchmarks [4] for correctness and efficacy. The results of running our implementation on the full set of 48 benchmarks is shown in Table 1. Note that we have adjusted our runtimes (by multiplying them by 3.31) to enable a more objective comparison with the prior published results. Specifically, we sourced an i7 laptop from 2014 and determined from a respected online site (CPUBoss) that the difference in single threaded CPU speed of the Athlon used by Ansótegui et al. [2] and the early i7 in the laptop was a factor of 3.31.

In 40 of the 48 benchmarks, the ZLA outperformed all other implementations. This is an interesting result as the benchmarks are for EMP puzzles that have distinctly non-uniform colour distributions that should favour the other implementations that use a various mix of forward checking, look-ahead, back jumping and k-consistency checks. ZLA totally ignores any exploitable structure to the colours or tiles. Nevertheless, overall as judged by the cumulative geomet-

Table 1. Comparison of runtimes (seconds) published by Ansótegui et al. [2] with our ZLA implementation on the datasets provided by Bourreau and Benoist [4]. Boldface indicates the fastest runtime on that particular puzzle.

Puzzle ID	PLA-ONION	Benoist	MAC	SAT(PD)	ZLA
E_7_1.b6i6	30	50	1968	3040	**17**
E_7_2.b6i6	45	**33**	96	602	115
E_7_3.b6i6	184	179	10322	17713	**4.3**
E_7_4.b6i6	**9**	20	1517	250	42
E_8_1.b8i8	**33**	188	13036	4281	145
E_9_1.b9i9	4866	20000	20000	5937	**1540**
E_9_2.b9i9	337	2140	20000	3240	**127**
E_9_3.b11i12	16	5	277	28.5	**0.06**
E_9_4.b9i12	3	3	491	3.7	**0.07**
E_9_5.b10i11	51	160	549	174	**0.5**
E_9_6.b9i10	2657	931	20000	9854	**404**
E_9_7.b9i11	4	69	1251	57	**0.1**
E_9_8.b8i9	20000	15299	20000	6950	**4136**
E_9_9.b9i10	5193	16526	20000	6869	**57**
E_9_10.b10i10	**16**	119	8404	62	73
E_9_11.b9i11	35	21	2849	3	**2**
E_9_12.b10i10	15	70	7278	303	**3**
E_9_13.b10i10	27	14	7568	29	**3**
E_9_14.b9i10	916	1961	15231	3645	**29**
E_9_15.b10i10	592	2298	2436	569	**14**
E_9_16.b9i10	12	2710	15280	876	**4**
E_9_17.b9i10	392	2317	20000	11323	**9**
E_10_1.b11i11	7372	12106	20000	**5904**	14704
E_10_2.b11i11	7641	20000	20000	**3257**	6804
E_10_3.b11i11	8480	11399	20000	6012	**721**
E_10_4.b11i11	1849	4544	20000	6075	**1486**
E_10_5.b14i14	16	8	34	0.7	**0.05**
E_10_6.b11i12	2077	2552	20000	1098	**88**
E_10_7.b12i15	2	18	3	6.7	**0.01**
E_10_8.b13i14	13	34	88	14	**0.08**
E_10_9.b12i14	24	130	70	63.5	**0.1**
E_10_10.b12i12	444	196	20000	611	**34**
E_10_11.b13i13	14	54	321	65	**0.1**
E_10_12.b12i13	9	53	173	36	**0.1**
E_10_13.b12i12	**67**	2222	20000	260	106
E_10_14.b12i13	97	187	6553	253	**0.5**
E_10_15.b11i12	179	224	20000	688	**25**
E_10_16.b13i13	14	29	179	4.5	**0.4**
E_10_17.b13i13	0	18	684	6.6	**0.6**
E_10_18.b12i13	24	87	2647	2	**0.4**
E_10_19.b12i13	58	348	4886	3.5	**1**
E_10_20.b11i14	282	1573	9175	45.6	**3**
E_10_21.b12i13	60	261	5364	21	**2**
E_10_22.b12i13	**1**	38	3986	7	2
E_10_23.b12i12	233	1055	17019	223	**6**
E_10_24.b12i12	414	8913	20000	1094	**18**
E_10_25.b12i12	107	123	20000	202	**8**
E_10_26.b12i12	2000	9950	20000	3526	**80**
Geometric Mean	94.79	310.04	3419.49	234.26	**7.76**

ric mean (as suggested in [2]), the ZLA implementation clearly outperforms the other implementations by a substantial margin.

A more meaningful comparison is with harder EMP puzzles, which is shown in Table 2 using datasets provided by Ansótegui et al. [2]. In-line with the previous research, each cell represents the median time, over 100 instances of these puzzles, for a particular implementation. It is interesting to note that ZLA is the fastest implementation across all puzzles. This is in spite of there still being a proportion of harder puzzles that do not meet the conditions for being Eternity II style. That is, some of the puzzles still have exploitable statistical features that should favour the non-ZLA implementations.

Table 2. Comparison of runtimes (seconds) published by Ansótegui et al. [2] for their GEMP problems with our ZLA algorithm for corresponding hard Eternity II style problems. Times for ZLA have been adjusted as specified above to allow direct comparison with the earlier results. Boldface indicates the fastest runtime on that particular puzzle.

Solver	Median Time (s)					
Puzzle Size ($n \times n$)	6×6		7×7			8×8
Colours inner:frame	6 : 2	6 : 4	7 : 2	7 : 3	7 : 4	8 : 2
PLA-DOM	15	520	18193	9464	581	8387
PLA-CHESS	0.5	5249	137	4181	6906	510
PLA-ONION	82	382	$> 2 \times 10^4$	$> 2 \times 10^4$	429	$> 2 \times 10^4$
MAC+GAColor	0.94	328	96	646	348	208
MAC+GAColor+Ctadiff	0.73	377	94	727	395	516
SAT(P)	7.45	$> 2 \times 10^4$	4418	$> 2 \times 10^4$	7960	6465
SAT(PD)	0.55	777	125	1785	682	359
MAC_b dom/deg	19	2415	$> 2 \times 10^4$	$> 2 \times 10^4$	3307	$> 2 \times 10^4$
Minion	125	3463	$> 2 \times 10^4$	$> 2 \times 10^4$	4675	$> 2 \times 10^4$
Benoist	133	681	$> 2 \times 10^4$	8535	124	$> 2 \times 10^4$
ZLA-E II style puzzles	**0.026**	**0.017**	**1.152**	**12.111**	**0.801**	**88.069**

6 Conclusion and Future Work

The first contribution of this paper is determining the criteria required for an Edge Matching Puzzle (EMP) to be of Eternity II style, and providing an algorithm to generate puzzles guaranteed to meet these criteria. This provides a valuable way to test algorithms to solve hard Eternity II style puzzles and is an improvement over methods in the existing literature that generate datasets that sometimes do not meet the Eternity II style criteria.

The second main finding from this work is that for the range of Eternity II style puzzles examined, a ZLA implementation was significantly faster (up to

many orders of magnitude) than the CSP or SAT implementations published to date. This is an interesting result that reinforces the assertion in prior literature that a good variable ordering heuristic will outperform computationally expensive domain reduction heuristics and back jumping [10]. That is, the computational expense of invoking those heuristics will almost always slow down the search to result in slow implementation runtimes relative to a well-constructed (instance-specific) variable ordering heuristic.

An unexpected result was the complete dominance of the ZLA implementation on the relatively easy EMP puzzles. It was anticipated here that any CSP or SAT implmentation would be able to exploit the statistical features in easy puzzle instances and thus outperform the ZLA implementation.

There are future research opportunities in testing which domain minimization heuristics, when added to the ZLA implementation, result in better runtimes than the raw ZLA implementation presented in this paper. The knowledge gained in the comparison of the trade-off between computational overhead against the search space size reduction should be of great value to the CSP and SAT community.

References

1. Demaine, E.D., Demaine, M.L.: Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity. Graphs and Combinatorics **23** (2007) 195–208
2. Ansótegui, C., Béjar, R., Fernández, C., Mateu, C.: On the hardness of solving edge matching puzzles as SAT or CSP problems. Constraints (2013) 1–31
3. Schaus, P., Deville, Y.: Hybridization of CP and VLNS for eternity II. Journées Francophones de Programmation par Contraintes (JFPC'08) (2008)
4. Bourreau, E., Benoist, T.: Fast global filtering for Eternity II. Constraint Programming Letters (CPL) **3** (2008) 036–049
5. Munoz, J., Gutierrez, G., Sanchis, A.: Evolutionary techniques in a constraint satisfaction problem: Puzzle Eternity II. In: IEEE Congress on Evolutionary Computation (CEC'09), IEEE (2009) 2985–2991
6. Wang, W.S., Chiang, T.C.: Solving Eternity-II puzzles with a tabu search algorithm. In: Proceedings of the 3rd International Conference on Metaheuristics and Nature Inspired Computing (META). Volume 10. (2010)
7. Verhaard, L.: Details of eternity II solver eii. `http://www.shortestpath.se/eii/eii_details.html` (2009)
8. Aaronson, S.: Guest column: NP-complete problems and physical reality. ACM Sigact News **36**(1) (2005) 30–52
9. Harris, G., Vanstone, B., Gepp, A.: Code and data repository: Automatically generating and solving eternity II style puzzles. `https://bvanston@bitbucket.org/bvanston/bag-canadian-conference-2018.git` (2018)
10. Bessiere, C., Régin, J.C.: MAC and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems. In: Principles and Practice of Constraint Programming-CP96, Springer (1996) 61–75