How Hard is a Commercial Puzzle: the Eternity II Challenge[1]

Carlos ANSÓTEGUI, Ramon BÉJAR, Cèsar FERNÁNDEZ, and Carles MATEU
{carlos,ramon,cesar,carlesm}@diei.udl.cat
Dept. of Computer Science, Universitat de Lleida, SPAIN

Abstract. Recently, edge matching puzzles, an NP-complete problem, have received, thanks to money-prized contests, considerable attention from wide audiences. We consider these competitions not only a challenge for SAT/CSP solving techniques but also as an opportunity to showcase the advances in the SAT/CSP community to a general audience. This paper studies the NP-complete problem of edge matching puzzles focusing on providing generation models of problem instances of variable hardness and on its resolution through the application of SAT and CSP techniques. From the generation side, we also identify the phase transition phenomena for each model. As solving methods, we employ both; SAT solvers through the translation to a SAT formula, and two ad-hoc CSP solvers we have developed, with different levels of consistency, employing several generic and specialized heuristics. Finally, we conducted an extensive experimental investigation to identify the hardest generation models and the best performing solving techniques.

1. Introduction

The purpose of this paper is to introduce a new set of problems, edge matching puzzles, a problem that has been shown to be NP-complete [8], modelling them as SAT/CSP problems. Edge matching puzzles have been known for more than a century (E.L. Thurston was granted US Patents 487797 and 487798 in 1892) and there is a number of child toys based on edge matching puzzles. These puzzles have recently received world wide attention with the publication of an edge matching puzzle with a money prize of 2 million dollars if resolved (*Eternity II*). This kind of competitions is both, a challenge to develop more competitive SAT/CSP solvers, and a real showcase to show recent advances in hard problem solving attained by the SAT/CSP community.

Our contribution is threefold. First, we provide an algorithm for generating edge matching puzzles. The proposed algorithm is simpler and faster than other generators of hard SAT/CSP instances. Second, to our best knowledge, we provide the first detailed analysis of the phase transition phenomenon for edge matching puzzles in order to locate hard/easy puzzles. Third, we provide a collection of solving methods and a wide experimental evaluation. This collection includes SAT and CSP solving techniques. The overall solving process is to encode the edge matching puzzle as a SAT instance or CSP, and then to apply a SAT or CSP solver in order to obtain a solution for the puzzle. For SAT, we provide different SAT encodings and we apply state-of-the-art preprocessors and SAT solvers. For CSP, we encode the puzzles as CSPs with both binary and higher arity constraints and solve them with state-of-the-art CSP solvers. We have also developed two

[1]Research partially supported by projects TIN2006-15662-C02-02, TIN2007-68005-C04-02 and José Castillejo 2007 program funded by the *Ministerio de Educación y Ciencia*

ad-hoc solvers based on Partial Look-ahead (PLA) [7] and Maintaining Arc-Consistency (MAC) [6] algorithms, respectively. These ad-hoc solvers are enhanced with specialized heuristics and filtering algorithms to increase performance and efficiency. Another reason for using ad-hoc CSP solvers instead of standard solvers is that this way we can use an implicit encoding of the problem that is more compact than using explicit encodings as in standard solvers, as Minion [12].

2. Preliminary Definitions

Roughly described, an edge matching puzzle is a puzzle where we must place a set of tokens in a board following a simple rule. Tokens have four sides (called also half-edges), in our case for simplicity we assume square tokens, each of a different color or pattern. The rule to follow when placing tokens is that two tokens can be placed side by side iff adjacent half-edges are of the same color (or pattern), such that when placed side by side they will form an edge with an unique color. A more formal definition is as follows,

Definition 1 (Generic Edge Matching Puzzle (GEMP)) *A Generic Edge Matching Puzzle (GEMP) , $P(n \times m, c)$ of size $n \times m$ and c colors, is a tuple (V, S), where V is the set of variables representing cell positions on the plane, of the form, $V = \{v_{i,j}, 1 \leq i \leq n, 1 \leq j \leq m\}$. Variables in V take values from the domain S, with $S = \{(t, r)|t \in \{T\}, r \in \{R\}\}$ being R the set of possible rotations $(0°, 90°, 180°, 270°)$, T the token subset of the form $T \subset \{(x_1, x_2, x_3, x_4)|x_i \in C\}$ and C is the set of colors, $C = \{c_i, 1 \leq i \leq c\}$.*

One possible variant on GEMPs is that where token rotations are not allowed, that is, all tokens must be placed **exactly** in the same orientation as they are in the puzzle specification. Actually, this last variant coincides with the Tetravex puzzle, that has been shown also to be NP-complete [22].

Definition 2 (Generic Edge Matching Puzzle Solution) *A valid solution for a GEMP, $P = (V, S)$ is an assignment of values from S to all the variables in V such that for each pair of neighboring variables, the color value assigned to the adjacent half-edges between those two variables is the same.*

Definition 3 (Framed GEMP (GEMP-F)) *A Framed Generic Edge Matching Puzzle (GEMP-F), $P(n \times m, c)$ is a Edge Matching Puzzle that includes a special color, we represent it in figure 1 as 'gray'(0), that, in all valid solutions can only appear in variables located at the frame of the puzzle, i.e., those variables in $\{v_{1,j}, v_{n,j}|1 \leq j \leq m\} \bigcup \{v_{i,1}, v_{i,m}|1 \leq i \leq n\}$ and only on the outside half-edges of those variables.*

One could think on several variants of framed puzzles attending to the sets of colors employed on distinct areas of the puzzle. In this paper we deal with two types, that have a profound impact on hardness, **one-set GEMP-F** when colors can be used at any edge of the puzzle, and **two-set GEMP-F** when two disjoint sets of colors are used; one set for edges joining frame pieces and another set for any other edge. As an example take Figure 1. One can observe that colors joining frame pieces are different from the rest. As real-world puzzles (as in *Eternity II*[2]) are usually framed puzzles and due to the interesting effect that the frame has on hardness this work deals with GEMP-F.

During this work we study square $n \times n$ GEMP-F problems. Rectangular puzzles will probably not be harder, similarly to what happens in problems like Sudoku [2].

[2]In fact, *Eternity II* is a two-set GEMP-F.

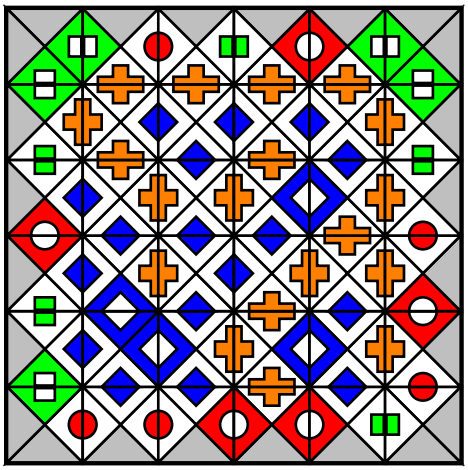

Figure 1. 6x6 size two-set GEMP-F example with 4 frame colors and 3 inner colors

Input: n, c
Output: an Edge Puzzle of size n with c colors
for $i = 1$ **to** n
  for $j = 1$ **to** n
    for $side = 1$ **to** 4
      if $v_{i,j}^{side}$ *is empty*
        $v_{i,j}^{side}$= random(c)
        $i', j' = \mathrm{N}_{\mathrm{side}}(i, j)$
        if $i', j' \neq 0, 0$
          $v_{i',j'}^{front(side)} = v_{i,j}^{side}$

Algorithm 1. Algorithm for generating GEMP(n, c) puzzles

3. Generation Models

The general method for a solvable puzzle generator is detailed in Algorithm 1. Roughly explained, the method assigns colors to edges of puzzle pieces (assigning a color to both half-edges). When all edges are colored, tokens are built from the existing color assignment. In the algorithm, $v_{i,j}^{side}$ refers to one of the four half-edges of the variable at position i, j, $N(v_{i,j})$ is the set of up to four neighbors of variable $v_{i,j}$ (at sides up, right, down and left), and $N_s(v_{i,j})$ gives position of neighbor at side s of $v_{i,j}$, if exists, and $0, 0$ otherwise. Finally, $front(s)$ gives the opposite side to s, that is, given two adjacent positions, s and $front(s)$ represent the two adjacent sides that join the two positions.

Special care must be taken on implementing this algorithm because this method does not prevent having repeated tokens or symmetric tokens (tokens with rotations that leave the token invariant), but for higher enough values of c (as those around the Phase Transition values), repetitions or symmetric tokens are low enough to do not suppose an impact on problem hardness.

Extending this algorithm to generate framed puzzles is easy. First the inner part of the puzzle is generated (tokens without gray color), without taking into account the frame. Then colors are assigned to the half-edges of the frame adjacent to inner tokens, that

are already determined by the inner tokens, and then half-edges that join tokens of the frame are filled with colors, randomly choosing either from the same set of colors used for the inner tokens (**one-set GEMP-F**) or from a second set of colors with no colors in common with the first set (**two-set GEMP-F**).

As it can be seen in the experimental results, this generation algorithm generates extremely hard solvable instances. The fact that the generation algorithm is so simple, in contrast with previous generation models for only-solvable structured problems like for example the one for quasigroups [1], or Sudoku [2], makes this generation model very interesting for a more detailed analysis. The most simple model for hard Satisfiable instances that we are aware of is the regular k-XORSAT [15,13] , but the instances generated are not inherently hard, as even if they are hard for $k-$consistency based algorithms [5], they can be solved in polynomial time due to their structure based on systems of linear equations. By contrast, we do not have any guaranteed particular structure in our instances that make them easy. So, as a first step, we present in this paper an analysis of the phase transition (PT) phenomenon for the not SAT-forced version of the model, and show that the PT point coincides remarkably well with the hardest instances point of our SAT-forced model. We also show in the experimental results that similarly to what happens in [1], here the hardest instances seem to be concentrated around the point where a sudden change in the backbone variables fraction occurs.

4. Solving approaches

The following section details the methods used for solving edge matching puzzles used in this paper. We use two different approaches to the problem, solving it as a SAT formula and as a CSP. For both methods, state of the art solvers or ad-hoc solvers have been used, choosing the most efficient ones for our experimental results in the following sections.

4.1. SAT solving

The objective is to solve the edge matching puzzles through its compilation to a SAT formula and the application of a SAT solver. The immediate advantage of this approach is the availability of a wide variety of competitive SAT solvers that can be applied to our SAT encoding. However, although there has been a significant advance in the engineering of efficient SAT solvers it is still a more immature question how to design *good* encodings for a given problem.

In the following we assume, $1 \leq i \leq n$, $1 \leq j \leq n$, $t \in T$, $r \in R$, $d \in D$ and $D = \{up, right, down, left\}$.

The first SAT encoding we introduce is the *primal* encoding. The *primal* Boolean variables $p_{t,r,i,j} \in P_b$, have the following meaning: $p_{t,r,i,j}$ is true iff the token t with rotation r is *placed* at cell (i, j). The primal constraints are the following:

- P1. A cell has exactly one token *placed* on it.
  $\bigwedge_{i,j} \left(\sum_{t,r} p_{t,r,i,j} = 1 \right)$
- P2. A token is exactly *placed* on one cell.
  $\bigwedge_{t} \left(\sum_{i,j,r} p_{t,r,i,j} = 1 \right)$
- P3. A piece matches its neighbours.
  $\bigwedge_{t,r,i,j,d} (p_{t,r,i,j} \rightarrow \bigvee_{p \in P'_b} p)$ such that P'_b is the set of variables that represent the *placed* tokens at the cell at direction d from cell (i, j) that match the color of $p_{t,r,i,j}$.
- P4. Only the pieces at the frame can have the gray color. We write a set of unit clauses $\neg p_{t,r,i,j}$. For the pieces at the frame: t, r, i, j corresponds to a piece placed at the frame

which has not the gray color at the border. For the internal pieces: t, r, i, j corresponds to any gray colored piece placed internally.

Similarly, we could think on an encoding just working on a set of dual variables, where the dual variables represent how the edges of the puzzle are colored. The *dual* Boolean variables $e_{c,d,i,j} \in E_b$ have the following meaning: $e_{c,d,i,j}$ is true iff the edge located at cell (i, j) at direction d is *colored* with color c. Since internal edges belong to two cells, we can just use one Boolean variable to represent that an edge takes a certain color. For the sake of space, we skip the dual encoding and we present the constraints for what we call the *primal-dual* encoding:

- PD1. P1 $\wedge$ P2.
- PD2. An edge is exactly *colored* with one color.
  $\bigwedge_{i,j,d} (\sum_c e_{c,d,i,j} = 1)$ Since the internal edges belong to two cells, we avoid repeating the same constraint.
- PD3. There are exactly $k_c/2$ internal edges *colored* with color c.
  $\bigwedge_c \left(\sum_{d,i,j} e_{c,d,i,j} = k_c/2\right)$
  k_c is the number of times the color c appears at the tokens. We do not take here into account the gray color.
- PD4. If a token is *placed* on a cell then the edges have to *match*.
  $\bigwedge_{t,r,i,j} \bigwedge_{e \in E'_b} (p_{t,r,i,j} \rightarrow e)$, such that E'_b is the set of variables that represent the edges at cell (i, j) with a direction an a color that *match* the token t with rotation r at cell (i, j).
- PD5. If an edge is *colored*, then the tokens *placed* on the cells the edge belongs to have to *match*.
  $\bigwedge_{c,d,i,j} (e_{c,d,i,j} \rightarrow \left(\bigvee_{p \in P'_b} p\right))$ such that P'_b is the set of variables that represent the tokens at cell (i, j), with a rotation that has the color c at direction d.
- PD6. Only the edges at the frame are gray colored.
  $\bigwedge_{d,i,j} e_{gray,d,i,j}$, such that the values of d, i, j correspond to an external edge .
  $\bigwedge_{d,i,j} \neg e_{gray,d,i,j}$, such that the values of d, i, j correspond to an internal edge.

PD3 is actually a set of redundant constraints which contribute to increase the propagation power of the *complete* SAT solvers.

The above encoding channels the primal and dual encodings. Constraints PD5 and PD6 interconnect the primal and dual variables. On the one hand, they help to reduce the size of the encoding. On the other hand they increase the propagation power of SAT solvers. The level of inference we try to achieve is the one achieved by Arc Consistency in the CSP solvers, see [3].

The presented constraints have to be transformed into a conjunction of clauses. The following transformations are applied: (i) $A \rightarrow B \equiv \neg A \vee B$ were A and B are Boolean formulas and (ii) $\sum_{b \in B} b = k$ is a cardinality constraint that has to be efficiently transformed into clauses in order to keep the size of the formula as low as possible. When $k = 1$, the naive encoding has a quadratic size complexity while if we apply the transformation described in [4] we get a linear one. Similarly, when $k > 1$ we apply the default transformations applied by the pseudo-Boolean solver MiniSat+(v1.13) described in [11], which achieves a good tradeoff between the pruning power and the complexity of the encoding. Then, in order to simplify the resulting SAT formula we apply the preprocessor SatELite(v1.0) [9] with the default options. The transformation process with MiniSat+(v1.13) and the simplification with SatELite(v1.0) take less than five seconds for the hardest instances we have considered in our experimental investigation.

4.2. CSP Solving

Edge matching puzzles are easy modeled as CSP problems, with two basic sets of constraints, one set of constraints for neighboring relations, modelling the relation between half-edges and a set of global constraints modelling the fact that every token must be assigned to one variable. We have used two base algorithms for CSP solving, PLA (Partial Look-ahead) [14,7] and MAC (Maintaining Arc-Consistency), and we have added specific improvements for increasing constraint propagation. Both algorithms have been tested with two variable selection heuristics, DOM (minimum domain) and CHESS. CHESS is a static variable heuristic that considers all the variables of the problem as if placed in a Chess board, and proceeds by choosing all 'black' variables following a spiral shaped order from the center towards the frame, and then repeats the same procedure with 'white' variables. That causes unitary variables (singletons) appear earlier.

With the MAC algorithm we have considered the inclusion of global (n-ary) constraints with powerful filtering algorithms for maintaining generalized arc-consistency (GAC). The most important n-ary constraint we have identified is the exactly-k constraint between the set of $2k$ half-edges with a same color. That is, in any solution this set of $2k$ half edges must be arranged in a set of k disjoint pairs of half-edges. With this aim, we use the symmetric alldiff constraint (that is formally equivalent to our exactly-k constraint), and its specialized filtering algorithm [20], that achieves GAC over this constraint in polynomial time. So, we define a symmetric alldiff constraint for each exactly-k constraint we have (one for each color). More specifically, we have a color graph that represents either already matched half-edges (that become disconnected from the rest of the graph) or half-edges that could be matched given the current domains of the unassigned variables. Observe that in order to be able to extend the current partial solution to a complete solution, a necessary condition is that any color graph must contain at least one perfect matching. If this is not the case for some color, we can backtrack. Moreover, using the filtering algorithm of Regin we can eliminate any edge that will not appear in any perfect matching of the graph (i.e. to maintain GAC) and discover forced partial matchings.

We have also considered maintaining GAC for the alldiff constraint over the set of position variables using the filtering algorithm of [19].

5. Experimental Results

We present experimental results and an analytical approach for the location of the Phase Transition on one-set and two-set GEMP-F models, as well as a solver performance comparison on the instances on the peak of hardness. The hardness of GEMP-F problems is evident from the median times and from the fact that to obtain the experimental data for this section the total CPU time has been 5.5 CPU/years on a single Opteron 1.8 Ghz 64bit.

5.1. Model Hardness and Phase Transition

One-set and two-set GEMP-F present a hardness characterization depending on their constituent number of colors. As shown in Figure 2, an accurate selection of the number of colors increases the puzzle hardness by several orders of magnitude. While comparing results for one-set and two-set GEMP-F, it is worth to note that for the same puzzle size, two-set GEMP-F are harder.

As detailed in [1], one can link this hardness characterization, on only satisfiable problems, with a phase transition effect when the backbone is considered, i.e. the number of variables that take the same value on all the solutions [17]. Figure 3 shows this phase transition plotting the fraction of the backbone as a function of the number of inner colors (c_m) for two-set GEMP-F with 3 frame colors ($c_f = 3$).

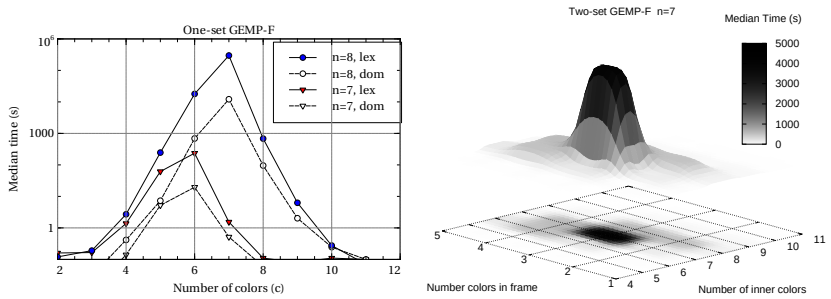

Figure 2. Hardness characteristic for a one-set GEMP-F as a function of the number of colors (left). Hardness characteristic for a 7x7 two-set GEMP-F as a function of the number of colors. Minimum median time of PLA-CHESS and PLA-DOM (right)

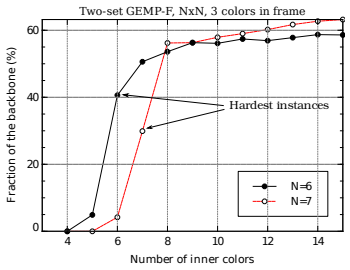

Figure 3. Phase transition of the percentage of backbone variables for a two-set GEMP-F

From an analytical point of view, we can derive some expressions that predict the phase transition location. For the sake of tractability, we consider tokens generated randomly, unregarding adjacency constraints that give only SAT puzzles. Of course, this is only an approach, but experimental results and numerical evaluations agree for both models. As usual in SAT/UNSAT models, the point where the expected number of solutions ($E[X]$) is small, but not negligible, marks the phase transition [21,18] for random CSP problems, being proved by [23] that such a transition occurs for $E[X] = 1$ on Model RB. Of course, we have not the same level of granularity on GEMP problems than in Random CSP models, and we are not able to tune our parameters to lead $E[X]$ to a desired point, but we can observe in Table 1 how the point where $E[X]$ changes from many to few solutions predicts where the harder instances are. Appendix 1 shows in detail the computation for the first moment of the number of solutions for one-set and two-set GEMP-F. It is worth to note that for $n = 16$ and $c_f = 5$ the predicted phase transition occurs at $c_m = 17$ that is exactly the number of inner colors of the two-set GEMP-F puzzle used in *E*ternity II contest.

Table 1 shows that hard instances may be found for one or two contiguous values of c_m, meaning that their respective median times to solve are equivalent. That is usual for small orders, tending to disappear for larger n and therefore concentrating their hard problems for a given value of c_m. Actually, using Markov inequality that gives an upper bound to the probability of having a satisfiable instance, $P(Sat) \leq E[X]$, it can be shown that $\lim_{n\to\infty} P(Sat) = 0$ beyond a critical value of $c_m > c_{m_{cr}}$. From Equations 1 we obtain that $c_{m_{cr}} = \frac{2n}{\sqrt{e}}$. Similar result apply for one-set GEMP-F.

Table 1. Round of $\log_{10}(E[X])$ according to Eq. 1 for two-set GEMP-F. Shadowed cells shows where the hardest problems have been experimentally found

	$\mathbf{n=6}$				$\mathbf{n=7}$					$\mathbf{n=8}$		
$\mathbf{c_f \setminus c_m}$	4	5	6	7	4	5	6	7	8	6	7	8
3	4	0	-3	-6	12	7	2	-2	-6	10	4	-1
4	2	-2	-6	-8	9	4	-1	-5	-9	6	1	-4
5					7	1	-3	-7	-11	3	-2	-7
6					5	-1	-5	-9	-13	1	-4	-9

Table 2. Comparison of solving approaches for the one-set and two-set models. 100 instances per point.

	One-set GEMP-F		**Two-set GEMP-F**					
Size ($\mathbf{n \times n}$)	7×7	8×8	6×6	7×7				
Colors inner[:frame]	6	7	6:2	6:4	7:2	7:3	7:4	8:2
PLA-LEX	235	$>2{\cdot}10^5$	-	-	-	-	-	-
PLA-DOM	**20**	**12,125**	15	520	18,193	9,464	581	8,387
PLA-CHESS	42	52,814	**0.5**	5,249	137	4,181	6,906	510
MAC+GAColor	90	23,210	0.94	**328**	96	**646**	**348**	**208**
MAC+GAColor+CTadiff	92	22,442	0.73	377	**94**	727	395	216
SAT(P)	1,341	$>2{\cdot}10^5$	7.45	$>2{\cdot}10^4$	4,418	$>2{\cdot}10^4$	7,960	6,465
SAT(PD)	34	117,823	0.55	777	125	1,785	682	359
$\mathrm{MAC_b}$ dom/deg	154	39,742	19	2,415	$>2{\cdot}10^4$	$>2{\cdot}10^4$	3,307	$>2{\cdot}10^4$
Minion	413	$>2{\cdot}10^5$	125	3,463	$>2{\cdot}10^4$	$>2{\cdot}10^4$	4,675	$>2{\cdot}10^4$
Solver	**Median Time (seconds)**							

5.2. SAT and CSP Solving Methods

We generated the SAT instances according to the two previously described SAT encodings. The complete SAT solvers we experimented with were: Minisat2 (v.061208-simp) [10], siege(v.4.0), picosat(v.535) and satz [16]. Minisat2 was the best performing SAT solver, and required to activate the option *polarity-mode=true*. For the state-of-the-art CSP solvers, Minion and $\mathrm{MAC_b}$ dom/deg[6], we adapted the primal and primal-dual encodings taking into account variables with a domain greater than or equal to two (we only report results for the best encoding).

Table 2 shows median time results for one-set and two-set GEMP-F with distinct sizes and number of colors, solved with several techniques. These techniques are: (i) PLA CSP solvers with variable selection heuristics LEX, DOM and CHESS, explained above; (ii) MAC with filtering algorithm for Generalized Arc-Consistency for color graphs (GAColor), using CHESS heuristic, with and without GAC filtering for the alldiff over position variables (CTadiff); (iii) the Minisat2 (v.061208-simp) [10] on the primal encoding SAT(P), and on the primal-dual encoding SAT(PD), and (iv) the state-of-the-art CSP solvers Minion and $\mathrm{MAC_b}$ dom/deg.

On one hand, the best performer for one-set GEMP-F is PLA-DOM meanwhile for two-set puzzles MAC+GAColor is the best one. It seems that the additional pruning effect of the GAColor filtering is powerful enough to pay off the additional time needed by such filtering in the two-set GEMP-F.

On the other hand, on PLA solvers for two-set GEMP-F, CHESS heuristic performs better than DOM when the number of frame colors is lower, and this could be because CHESS instantiates frame variables at the end of the search, and in those cases, the probability of finding a consistent frame is higher than when the number of frame colors is

higher. About SAT solvers, the best performing encoding is the primal-dual encoding being quite competitive with the CSP approaches, but still with a poor scaling behaviour.

6. Conclusions

This work clearly shows that edge matching puzzles are a very hard problem with a reduced and simple definition and a very easy and fast generation process. State of the art solvers (SAT or CSP) cannot solve problems bigger than a meager 8×8. Even using sophisticated specialised filtering algorithms, solvers are unable to keep pace with the problem hardness scaling. This makes GEMP-F a really challenging problem.

Appendix 1

In this appendix, we derive exact expressions to the number of solutions of one-set and two-set GEMP-F, when tokens are generated at random, unregarding adjacency constraints that give only SAT puzzles.

For a two-set GEMP-F, according to Definition 1, one can think on set T as $T = T_c \cup T_f \cup T_m$, being T_c, T_f and T_m the set of tokens corresponding to the corners, rest of the frame and mid of the board respectively.

Lets denote as $\mathcal{S} = \mathcal{S}_c \times \mathcal{S}_f \times \mathcal{S}_m$ the set of possible locations on the board for T_c, T_f and T_m jointly, and $\mathcal{C}$ the subset of $\mathcal{S}$ that satisfies 2-set GEMP-F rules. Clearly, considering a $n \times n$ board, and that only elements of the set T_m can be rotated:

$$|T_c| = 4, \quad |T_f| = 4(n-2), \quad |T_m| = (n-2)^2$$
$$|\mathcal{S}_c| = 4!, \quad |\mathcal{S}_f| = (4(n-2))!, \quad |\mathcal{S}_m| = 4^{(n-2)^2} \cdot ((n-2)^2)!$$

We define X as the random variable that denotes the number of satisfying locations according to the rules of 2-set GEMP-F puzzles, (i.e. the elements of $\mathcal{C}$). So, its expectation can be expressed as

$$\begin{aligned} E[X] &= E\left[\sum_{\sigma \in \mathcal{S}} \mathbf{1}_{\mathcal{C}}(\sigma)\right] = \sum_{\sigma \in \mathcal{S}} E[\mathbf{1}_{\mathcal{C}}(\sigma)] \\ &= \sum_{\sigma_c \in \mathcal{S}_c} \sum_{\sigma_f \in \mathcal{S}_f} \sum_{\sigma_m \in \mathcal{S}_m} E[\mathbf{1}_{\mathcal{C}}(\sigma_c \times \sigma_f \times \sigma_m)] \\ &= 4! \cdot (4(n-2))! \cdot 4^{(n-2)^2} \cdot (n-2)^2! \cdot E[\mathbf{1}_{\mathcal{C}}(\sigma_c \times \sigma_f \times \sigma_m)], \end{aligned}$$

being $\mathbf{1}_A(x)$ the indicator function, i.e., takes value 1 if $x \in A$ and 0 if $x \notin A$. We claim that $E[\mathbf{1}_{\mathcal{C}}(\sigma_c \times \sigma_f \times \sigma_m)]$ is the probability that a given arrangement of tokens satisfies a 2-set GEMP-F puzzle. If tokens are build randomly, such a probability is

$$E[\mathbf{1}_{\mathcal{C}}(\sigma_c \times \sigma_f \times \sigma_m)] = \left(\frac{1}{c_f}\right)^{4(n-1)} \cdot \left(\frac{1}{c_m}\right)^{2(n-1)(n-2)},$$

being c_f and c_m the number of colors in frame and mid, respectively, and giving

$$E[X] = 4! \cdot (4(n-2))! \cdot 4^{(n-2)^2} \cdot (n-2)^2! \cdot \left(\frac{1}{c_f}\right)^{4(n-1)} \cdot \left(\frac{1}{c_m}\right)^{2(n-1)(n-2)} \quad (1)$$

Analogously, one can derive an exact expression for one-set GEMP-F, resulting

$$E[X] = 4! \cdot (4(n-2))! \cdot 4^{(n-2)^2} \cdot (n-2)^2! \cdot \left(\frac{1}{c}\right)^{2n(n-1)}, \quad (2)$$

where c is the number of colors.

References

[1] Dimitris Achlioptas, Carla Gomes, Henry Kautz, and Bart Selman. Generating satisfiable problem instances. In *Proceedings of the AAAI 2000*, pages 256–261. AAAI Press / The MIT Press, 2000.

[2] C. Ansótegui, R. Béjar, C. Fernández, C. Gomes, and C. Mateu. The impact of balance in a highly structured problem domain. In *Proceedings of the AAAI 2006*, pages 438–443. AAAI Press / The MIT Press, 2006.

[3] Carlos Ansótegui, Alvaro del Val, Iván Dotú, Cèsar Fernández, and Felip Manyà. Modelling choices in quasigroup completion: SAT vs CSP. In *Proceedings of the AAAI 2004*. AAAI Press / The MIT Press, 2004.

[4] Carlos Ansótegui and Felip Manyà. Mapping many-valued CNF formulas to boolean CNF formulas. In *Proceedings of the International Symposia on Multiple-Valued Logic*, pages 290–295, 2005.

[5] Albert Atserias, Andrei A. Bulatov, and Víctor Dalmau. On the power of *k*-consistency. In *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007*, volume 4596 of *Lecture Notes in Computer Science*, pages 279–290. Springer, 2007.

[6] Christian Bessière and Jean-Charles Régin. MAC and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems. In *Principles and Practice of Constraint Programming*, pages 61–75, 1996.

[7] Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[8] Erik D. Demaine and Martin L. Demaine. Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity. *Graphs and Combinatorics*, 23(s1):195, 2007.

[9] Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In *Proceedings of SAT 2005*, pages 61–75, 2005.

[10] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Proceedings of SAT 2003*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[11] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *Journal of Satisfiability*, 2:1–26, 2006.

[12] Ian P. Gent, Christopher Jefferson, and Ian Miguel. Watched literals for constraint propagation in minion. In *Principles and Practice of Constraint Programming.*, pages 182–197, 2006.

[13] Harri Haanpää, Matti Järvisalo, Petteri Kaski, and Ilkka Niemelä. Hard satisfiable clause sets for benchmarking equivalence reasoning techniques. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):27–46, 2006.

[14] R. M. Haralick and G. L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *AI Journal*, 14:263–313, 1980.

[15] Matti Järvisalo. Further investigations into regular XORSAT. In *Proceedings of the AAAI 2006*. AAAI Press / The MIT Press, 2006.

[16] Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI 97.*, pages 366–371. Morgan Kaufmann, 1997.

[17] Rémi Monasson, Riccardo Zecchinna, Scott Kirkpatrick, Bart Selman, and Lidror Troyansky. Determining computational complexity from characteristic phase transitions. *Nature*, 400:133–137, 1999.

[18] P. Prosser. An empirical study of phase transitions in binary constraint satisfaction problems. *AI Journal*, 81:81–109, 1996.

[19] Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the AAAI 1994*, pages 362–367. AAAI Press / The MIT Press, 1994.

[20] Jean-Charles Régin. The symmetric alldiff constraint. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99*, pages 420–425. Morgan Kaufmann, 1999.

[21] B. Smith and M. Dyer. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81:155–181, 1996.

[22] Yasuhiko Takenaga and Toby Walsh. Tetravex is NP-complete. *Information Processing Letters*, 99(5):171–174, 2006.

[23] K. Xu and W. Li. Exact phase transition in random constraint satisfaction problems. *Journal of Artificial Intelligence Research*, 12:93–103, 2000.