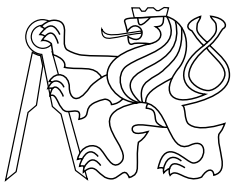
CENTER FOR
MACHINE PERCEPTION



CZECH TECHNICAL
UNIVERSITY IN PRAGUE

Habilitation thesis

Methods for Structural Pattern Recognition: Complexity and Applications

Daniel Průša

prusapa1@fel.cvut.cz

March 26, 2018

Center for Machine Perception, Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University
Technická 2, 166 27 Prague 6, Czech Republic
fax +420 2 2435 7385, phone +420 2 2435 7637, www: http://cmp.felk.cvut.cz

Contents

1 Introduction

The thesis comprises a collection of selected publications preceded by an introductory text. It studies methods for structural analysis, which is a stage of structural pattern recognition. Two formalisms suitable for structure modelling are examined: the min-sum labeling problem and two-dimensional grammar. The main focus is put on a contribution to the properties and complexity of these tools. The motivation for their study does not come purely from structural pattern recognition. The min-sum labeling problem has many applications in low-level computer vision where it needs to be solved for large-scale instances. The properties described in the thesis indicate where are the limits of some methods designed to tackle the problem in this setting. Two-dimensional grammars also have use in other fields, for example in computer-aided design.

Besides the theoretical results, the thesis presents two complete recognition systems, which have been implemented and tested on real data. Recognition of diagrammatic domains (flowcharts, finite-state automata) is cast as the binary min-sum labeling problem. Logical layout structure of PDF documents is defined and analysed via a novel type of a two-dimensional regular-like grammar.

The thesis is organized as follows. Section 2 gives a background on structural analysis, min-sum labeling problem and two-dimensional grammars. Section 3 overviews the contributions. Seven publications (5 journal and 2 conference papers) presenting the described results are included in Appendix A.

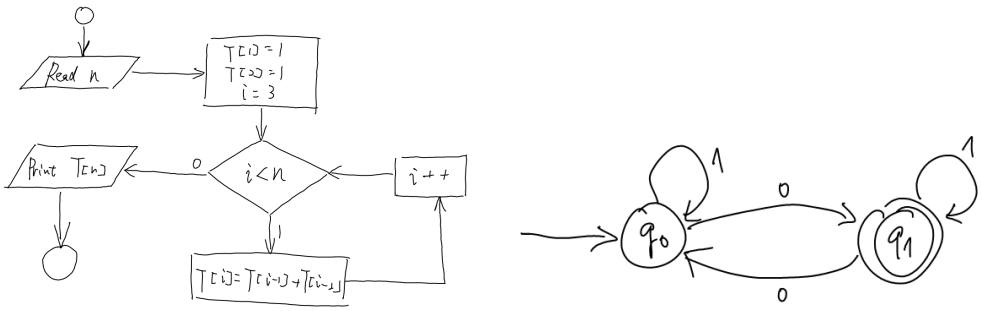
2 Background

This section reviews structural analysis, which is an important stage within the structural pattern recognition process, and two formalisms that are useful to implement it: the min-sum labeling problem and two-dimensional context-free grammars.

2.1 Structural analysis

The task of *structural pattern recognition* is to identify elementary units and interrelationships among them in input data. In contrast to *statistical pattern recognition*, it is has good use when data exhibit rich structure, which can not be handled by flat, numerical feature vectors of fixed dimensionality.

Several domains with very rich structure can be found in the field of document analysis and recognition (see Figure 1 for examples): mathematical formulas, chemical formulas, flowcharts, electric circuits, music notes or document layouts. There are also rich structural patterns e.g. in images of building facades. All these domains were considered by researchers as a subject for recognition in several past decades (Fahmy and Blostein, 1993; Kiyko, 1995; Savchynsky et al., 2003; Lemaitre et al., 2011; Sadawi et al., 2012; Álvaro et al., 2014, 2015; Liu and Liu, 2014).



(a) (b)

Figure 1: Examples of hand-written documents with rich structure: (a) flowchart, (b) finite-state automaton.

Two types of inputs are distinguished in the case of electronic documents – raster images (we speak about *off-line recognition*) and temporally ordered lists of strokes, captured by an electronic device like a tablet (we speak about *on-line recognition*).

The existing recognition methods are usually composed of several subtasks. Classifiers are required to identify elementary symbols, while *struc-*

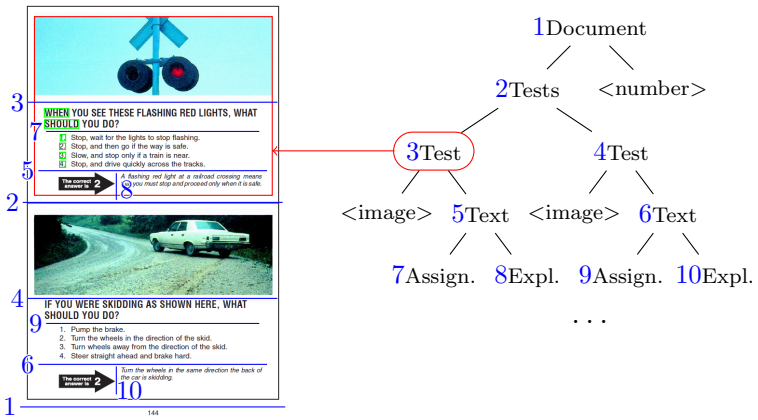


Figure 2: PDF document layout recognition: Structural analysis result represented as a derivation tree. Each inner node of the tree corresponds to a logical section in the document. Leaves of the tree correspond to terminal elements, which are words, numbers and images.

tural analysis takes care for establishing relations among the symbols and revealing thus a document structure. An example of a structural analysis result is depicted in Figure 2. More advanced methods do not treat symbols detection and structural analysis as two separate processes. An interaction is implemented to help to increase accuracy of both stages. This so called *structural construction paradigm* is explicated in (Schlesinger and Hlaváč, 2012).

Several formalisms have been used to model structure of documents. They include two-dimensional context-free grammars (Liang et al., 2005), graph grammars (Lavirotte and Pottier, 1998) and discrete optimization problems, e.g. finding a minimum-spanning tree (Eto and Suzuki, 2001). Moreover, attribute and stochastic grammars are grammar extensions that support combining structural and statistical approaches.

2.2 Min-sum labeling problem

The *min-sum (labeling) problem* is an NP-complete problem which arises in MAP inference in Markov random fields (Schlesinger, 1976; Wainwright and Jordan, 2008). It is also known as discrete energy minimization (Kappes et al., 2015) or valued constraint satisfaction (Thapper and Živný, 2012). The problem is quite flexible and can be applied to various structures. It has many applications in computer vision. As a tool for statistical-structural modelling, it has been applied in pattern recognition (Zeng and Liu, 2008).

And, as confirmed by the thesis contributions, it is also suitable for the structural analysis described in the previous subsection.

Given a set of discrete variables and a set of unary and binary (also called *pairwise*) functions, the task is to minimize the sum of the functions over all variables. Formally, let (V, E) be an undirected graph, where V is a finite set of *objects* and $E \subseteq \binom{V}{2}$ is a set of *object pairs*. Let K be a finite set of *labels*. Let $g_u\colon K \to \overline{\mathbb{R}}$ and $g_{uv}\colon K \times K \to \overline{\mathbb{R}}$ be unary and binary *cost functions*, where $\overline{\mathbb{R}} = \mathbb{R} \cup \{\infty\}$ and adopt that $g_{uv}(k, \ell) = g_{vu}(\ell, k)$. The min-sum problem is defined as

$$\min_{\mathbf{k} \in K^V} \left(\sum_{u \in V} g_u(k_u) + \sum_{\{u,v\} \in E} g_{uv}(k_u, k_v) \right). \tag{1}$$

The problem has a natural linear programming (LP) relaxation, proposed by a number of authors (Schlesinger (1976); Koster et al. (1998); Chekuri et al. (2001); Wainwright and Jordan (2008)): It reads

$$\min \left(\sum_{u \in V} \sum_{k \in K} g_u(k) x_{u;k} \; + \sum_{\{u,v\} \in E} \sum_{k \in K} \sum_{\ell \in K} g_{uv}(k, \ell) x_{uv;k\ell} \right) \tag{2}$$

$$\text{s.t.} \sum_{\ell \in K} x_{uv;k\ell} = x_{u;k}, \qquad u \in V,\ v \in N_u,\ k \in K \tag{3}$$

$$\sum_{k \in K} x_{u;k} = 1, \qquad u \in V \tag{4}$$

$$\mathbf{x} \geq \mathbf{0} \tag{5}$$

where $N_u = \{\, v \mid \{u, v\} \in E \,\}$ is the set of neighbours of u. We also assume $x_{uv;k\ell} = x_{vu;\ell k}$.

The relaxation is equivalent to the dual (Lagrangian) decomposition of the min-sum problem (Johnson et al., 2007; Komodakis et al., 2011). While the min-sum problem can be formulated as a linear optimization over the *marginal polytope*, the LP relaxation approximates this polytope by its outer bound, the *local marginal polytope* (Wainwright and Jordan, 2008). This polytope is the set of vectors fulfilling constraints (3), (4) and (5).

The relaxation is exact for a large class of min-sum instances and is a basis for constructing good approximations to many other instances (Thapper and Živný, 2012; Werner, 2007; Kappes et al., 2015). It is therefore of great practical interest to have efficient algorithms to solve the LP relaxation, however, the simplex and interior point methods are prohibitively inefficient for large-scale instances. They are neither suitable for real-time applications processing medium-sized instances.

The situation is much more favourable in the case of the min-sum problem with two labels (*binary min-sum problem*). It is known that the LP relaxation of this variant is half-integral, i.e., there is always an optimal solution whose all components are in $\{0, \frac{1}{2}, 1\}$. Moreover, the LP reduces in linear time to the *quadratic pseudo-boolean optimization*, whose LP relaxation reduces in linear time to the maximum flow (Boros and Hammer, 2002; Rother et al., 2007). A very efficient implementation solving the maximum flow problem is available (Kolmogorov and Rother, 2007). It outputs an exact optimal solution for submodular instances and plays an important role in methods dealing with the general problem as every multi-label min-sum reduces to the binary one (Ishikawa, 2003; Schlesinger and Flach, 2006), preserving the submodularity property. It is also utilized in methods seeking for approximate solutions (Boykov et al., 2001).

Regarding the min-sum problem with at least 3 labels, no really efficient algorithm (running e.g. in nearly linear time and space) is known to solve the LP relaxation. There are fast message passing algorithms (Kolmogorov, 2006; Werner, 2007; Globerson and Jaakkola, 2008) converging to a local optimum of the dual LP relaxation, characterized by arc consistency of the locally optimal tuples. This local optimum is nevertheless often good in practice.

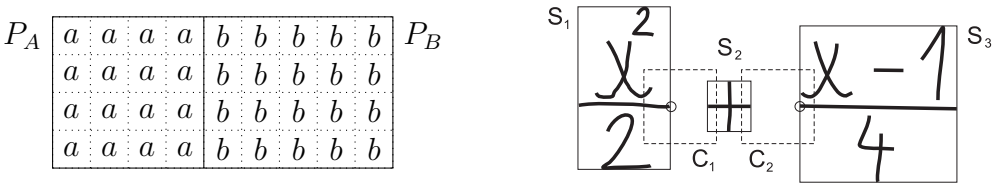
2.3 Two-dimensional grammars

Two-dimensional (2D) grammars are natural extensions of grammars generating strings over a finite alphabet. They have been studied from 60's of the past century, within the theory of two-dimensional languages as well as in the scope of document structure recognition. For example, Anderson considered 2D grammars in his seminal paper on printed mathematical formulas recognition (Anderson, 1967).

Early models of 2D context-free grammars generating 2D arrays of symbols (so called *pictures*) include the matrix and Kolam type grammars (Siromoney et al., 1972, 1973). The Kolam grammar forms a natural extension of the context-free grammar in the Chomsky normal form. This explains why it has attracted attention of several researchers and was proposed independently more times (Schlesinger, 1989; Matz, 1997). Beside that, extensions of the grammar have been studied. They include the two-dimensional context-free grammar with general right-hand sides of productions (Průša, 2004) and regional tile grammar (Pradella et al., 2011).

Let as give a definition of the Kolam grammar.

Definition 2.1. A *2D Kolam grammar* is a tuple $\mathcal{G} = (V_N, V_T, \mathcal{P}, S_0)$, where



(a) (b)

Figure 3: (a) 2D Kolam grammar: Applying production $N \to A\,B$ to concatenate pictures P_A and P_B by columns to derive a picture of size 4×9. It is assumed that P_A is generated from A and P_B is generated from B. (b) Mathematical formula recognition (Stria and Průša, 2011): Applying a production with a spatial constraint to derive a sum of two expressions. S_1, S_2 and S_3 are bounding boxes of sets (regions) of elementary symbols. Constraining rectangles C_1 and C_2 are computed with respect to symbol $+$. The circled point in S_1 and S_3 is required to be in C_1 and C_2, respectively. If the spatial constraint holds, S_1, S_2 and S_3 can be united into a new region representing a sum.

V_N is a finite set of non-terminals, V_T is a finite set of terminals, $S_0 \in V_N$ is the initial non-terminal and $\mathcal{P}$ is a finite set of productions in one of the following forms:

$$N \to a \qquad (6) \qquad\qquad S_0 \to \Lambda \qquad (7)$$

$$N \to A\,B \quad (8) \qquad\qquad N \to \begin{matrix} A \\ B \end{matrix} \quad (9)$$

where $N, A, B \in V_N$, $a \in V_T$ and Λ denotes the empty picture.

The grammar generates pictures over the set of terminals V_T. For each non-terminal $N \in V_N$, let $L(\mathcal{G}, N)$ be the set of pictures generated by $\mathcal{G}$ from N. Each production $N \to a$ of type (6) states that the picture P of size 1×1 consisting of the single symbol a is in $L(\mathcal{G}, N)$. Moreover, assuming there are pictures $P_1 \in L(\mathcal{G}, A_1)$ and $P_2 \in L(\mathcal{G}, A_2)$, both having the same number of rows, and there is a production $N \to A_1\,A_2$ of type (8), then the column concatenation of pictures P_1 and P_2 is in $L(\mathcal{G}, N)$. This case is illustrated in Figure 3(a). The same rule applies to productions of type (9) and row concatenations of pictures. The picture language generated by $\mathcal{G}$ is defined as $L(\mathcal{G}) = L(\mathcal{G}, S_0)$ where $L(\mathcal{G}, S_0)$ contains also the empty picture Λ if and only if the set of productions $\mathcal{P}$ contains production (7).

Classical parsing algorithms for string grammars can be extended to work with 2D context-free grammars. A 2D version of CKY algorithm (Younger,

1967) has been presented for the Kolam grammar (Reghizzi and Pradella, 2008; Schlesinger and Hlaváč, 2012). If an input picture is of size $m \times n$, it is parsed in time $\mathcal{O}\left(m^2n^2(m+n)\right)$. However, the complexity increases for more general 2D context-free grammars where the right-hand sides of productions consist of more variables. An extension of Earley parser (Earley, 1970) has also been proposed (Tomita, 1991).

2D arrays of symbols are not sufficient to model relationships occurring among elementary symbols (terminals) of real documents. These symbols have to be described by bounding boxes located at some coordinates. The grammar productions have to combine sets (regions) of elementary symbols into larger sets (regions). Moreover, spatial constraints are assigned to productions to specify when a particular production is applicable (see Figure 3(b)). Stochastic productions are used to deal with uncertainty. All these extensions make using the grammar more difficult, including more demanding parsing. In theory, the 2D topology and grammar ambiguity might result in parsing time exponential in the number of terminals. From this reason, techniques limiting the set of hypotheses examined and tested are required for acceleration (Liang et al., 2005). Nevertheless, 2D grammars are considered a powerful tool for document structure analysis and they are still popular nowadays and serve as a base for recognition in many domains (Lemaitre et al., 2011; Álvaro et al., 2014; Le et al., 2014; Álvaro et al., 2015).

3 Contributions

This section surveys my contributions to methods of structural analysis, complexity of solving the min-sum labeling problem and properties of two-dimensional grammars. The contributions are organized into four subsections below. Publications included in the thesis appendix relevant to each of the topics are indicated at the end of each subsection.

3.1 Universality of min-sum problem LP relaxation

In a joint work with my colleague Tomáš Werner, we answer the question whether it is possible to propose a very efficient algorithm solving the linear programming relaxation of the min-sum problem with 3 and more labels. Our results are negative as we showed that solving this special LP is not easier than solving any linear program. Precisely, we proved the following theorems.

Theorem 3.1. *Every linear program can be reduced in linear time to a linear optimization over a local marginal polytope with* 3 *labels.*

Theorem 3.2. *Every polytope is (up to scale and translation) a coordinate-erasing projection of a face of a local marginal polytope with* 3 *labels, whose description can be computed from the input polytope in linear time.*

An important consequence of these results is that finding a very fast algorithm to solve the LP relaxation would imply improving the best-known complexity of general LP, which is unlikely.

Having these negative results, one may ask whether the LP relaxation can be solved efficiently for some useful subclasses of the min-sum problem. One such subclass is planar min-sum problems or min-sum problems with *Potts* costs. Both these subclasses frequently occur in practical applications. We show that even in these cases, there is an efficient reduction of a general LP to the LP relaxation.

Theorem 3.2 is the reason why we speak about universality of the local marginal polytope. Similar universality results are also known for few other polytopes, e.g., the three-way transportation polytope (De Loera and Onn, 2006) and the travelling salesman polytope (Billera and Sarangarajan, 1996).

In (Průša and Werner, 2017a) we further extended the results on universality of LP relaxations. We proved that LP relaxations of more classical combinatorial NP-hard problems (set cover, maximum independent set, facility location, maximum satisfiability, etc.) are also as hard as any LP.

Publications included in the thesis: Průša and Werner (2015, 2017b).

3.2 Binary min-sum problem

As the next contribution, I showed how the simplex algorithm can be tailored to the linear programming relaxation of the binary min-sum problem. A special structure formed by basic and non-basic variables in each stage of the algorithm is identified and utilized to perform the whole iterative process combinatorially over the input graph rather than algebraically over the simplex tableau. This leads to a new efficient solver. It is demonstrated that for some artificial as well as computer vision instances it is faster than methods reducing the binary min-sum problem to finding maximum flow in a network.

This result illustrates that to solve the LP relaxation of the min-sum problem with 2 labels is really easier task when compared to the LP relaxation of the min-sum with at least 3 labels. Special versions of the simplex method with similar properties have already been proposed for *transportation*, *assignment* and *minimum cost-flow* problems (Dantzig and Thapa, 1997). They are known as *network simplex* algorithms.

There two open problems for future research. The first problem is how the proposed algorithm changes when some additional linear constraints are added to the LP relaxation. For example, one can incorporate a cardinality constraint giving a range for the sum of LP variables representing usages of one of the labels. We can conjecture that a constant number of added constraints would not change the basis structure too much, hence the algorithm could adapt.

The second open problem is whether the approach can be efficiently generalized to some multi-label instances since a similar, although a bit richer, graph structure of basic variables is observable. The universality results from Section 3.1 imply that such an approach will not be efficient in general, but there is a chance that the method might work well for some practical instances.

The second contribution to the binary min-sum problem is a joint work with Ph.D. student Martin Bresler, supervised by Václav Hlaváč and co-supervised by me. We introduced a new, on-line, stroke-based recognition system for hand-drawn diagrams which belong to a group of documents with an explicit structure obvious to humans but only loosely defined from the machine point of view. Examples of such diagrams are flowcharts or diagrams of finite-state automata.

The proposed method is a pipeline consisting of several stages: text/non-text separation, symbol candidate segmentation, symbol classification, arrow detection and structural analysis. Structural analysis is formulated as a binary max-sum problem, which is the maximization counterpart to the

binary min-sum problem. Each symbol candidate (a group of strokes which can potentially represent an input symbol) acts like a vertex of a graph. Assigning label 1 to a vertex means that the corresponding symbol candidate is recognized as a symbol in the document (assigning label 0 means it is not recognized as a symbol). Graph edges are added to represent relations between pairs of symbol candidates. Three types of relations are defined: 1) Conflict – two candidates share one or more strokes, or two arrows are connected to the same symbols; 2) Overlap – two symbol candidates have overlapping bounding boxes; 3) Endpoint – each arrow requires existence of both symbols it connects. Costs (scores) are assigned to symbol candidates (based on the classifier output) and relations (they are either positive or negative, depending on the relation type). Then we search for a 0-1 labeling of the graph with maximum cumulative score. Such an optimal labeling is found by writing the max-sum problem as an integer linear program (the formulation is similar to the LP relaxation, the only difference is that the variables attain value 0 or 1) and IBM ILOG CPLEX is used to solve it.

A thorough evaluation on benchmark databases shows that the accuracy of the system reaches the state-of-the-art.

Publications included in the thesis: Průša (2015a); Bresler et al. (2016).

3.3 Properties of two-dimensional context-free grammars

I have studied some theoretical properties of 2D context-free grammars. It is a well known phenomenon that the properties of picture languages recognized/generated by two-dimensional automata/grammars differ a lot when compared to their one-dimensional counterparts (Giammarresi and Restivo, 1997). The results I obtained for the two-dimensional context-free grammars are in line with this fact.

The emptiness problem is decidable for the one-dimensional context-free grammar (Hopcroft et al., 2006). This means there is an algorithm taking a context-free grammar as an input and computing whether the grammar generates at least one string or not. On the other hand, the problem is not decidable for the Kolam grammar. I first proved the undecidability for the more general 2D context-free grammar (Průša, 2014) and later for the 3D Kolam grammar (Průša, 2015b). The final proof for the 2D Kolam grammar has been established in a joint work with Klaus Reinhardt (Halle University, Germany) (Průša and Reinhardt, 2017). As an interesting consequence, there is no generalization of the pumping lemma for the Kolam grammar. This

suggests that the grammar is much more difficult regarding its automatic analysis, checking, etc. On the other hand, we have proved that the emptiness problem is decidable at least for the limited variant of the Kolam grammar – the matrix grammar.

The next result addresses succinctness of 2D grammars and automata. A *description size* of a grammar/automaton is measured in the number of its states, instructions or productions. An important question is how the size scales when an automaton/grammar of one type is replaced by an equivalent automaton/grammar of another type. We speak about so called *trade-offs* between the models. This *descriptional complexity* of systems has been studied heavily in the theory of one-dimensional languages. For example, it is well known that, for an n-state non-deterministic automaton, there is always an equivalent deterministic automaton with $\mathcal{O}(2^n)$ states (Hopcroft et al., 2006). The trade-off between these one-dimensional models is exponential.

Despite the extensive studies in the theory of formal languages, no comparison was done with respect to the descriptional complexity of 2D models. This gap has been filled in (Průša, 2016). I have shown that trade-offs between pairs of basic models of 2D context-free grammars and 2D finite-state automata are non-recursive (the size of the description of an equivalent automaton/grammar can not be bounded from above by any recursive function). Again, this result illustrates that two-dimensional arrays of symbols are quite complex objects.

Publications included in the thesis: Průša and Reinhardt (2017); Průša (2016).

3.4 Regular-like two-dimensional grammars

In a joint work with Akio Fujiyoshi (Ibaraki University, Japan), we contributed to the portfolio of 2D grammars by defining a subclass of the 2D Kolam grammar. Our goal was to propose a simpler version, which can be more easily parsed, but which is still powerful enough to have a practical significance. We restricted the 2D Kolam grammar by introducing productions with a *non-terminal rank-reducing property*.

Definition 3.1. Let $\mathcal{G} = (V_N, V_T, \mathcal{P}, S_0)$ be a 2D Kolam grammar. A function $\sigma : V_N \cup V_T \to \mathbb{N}$ is called a *rank function* of $\mathcal{G}$ if it fulfils $\forall a \in V_T : \sigma(a) = 0$ and $\forall N \in V_N : \sigma(N) > 0$. For $V \in V_N \cup V_T$, $\sigma(V)$ is called a *rank* of V.

Definition 3.2. We say that a $\mathsf{2CFG}$ $\mathcal{G} = (V_N, V_T, \mathcal{P}, S_0)$ is *rank-reducing* iff there is a rank function $\sigma\colon V_N \cup V_T \to \mathbb{N}$ such that

1. $\sigma(N) \geq \sigma(A)$ for each production $N \to A \in \mathcal{P}$,
2. $\sigma(N) \geq \sigma(B) > \sigma(A)$ for each production $N \to A\,B \in \mathcal{P}$ or $N \to \begin{matrix} A \\ B \end{matrix} \in \mathcal{P}$.

If a rank-reducing grammar does not have any production of type (9), it generates a set of one-row pictures (i.e. strings) and this set is a regular language. The same holds for a rank-reducing grammar without any production of type (8). The language of one-column pictures generated by the grammar is again regular.

Next, we proposed a top-down parsing algorithm strengthened by strongly discriminative rules for finding branching points. These rules are based on one-dimensional grammars describing sequences of terminal elements that can form borders of 2D arrays generated from non-terminals. As mentioned above, these one-dimensional grammars are regular. They can be extracted fully automatically from the 2D rank-reducing grammar, together with equivalent non-deterministic finite-state automata used to recognize the sequences of terminals. We show that with this method the number of backtracks during the parsing is minimal, hence the proposed algorithm is very efficient.

With some effort, the grammar and introduced technique of parsing can be extended to inputs where terminal symbols are freely located in a plane. We showed that this grammar extension is suitable for modelling document layouts that consist of logical sections like titles, paragraphs, lists of items, footnotes, etc. The method was tested on PDF documents since we are motivated by a development of digital document access methods for people with disabilities in which a retrieval of structural information plays an important role.

Publications included in the thesis: Průša and Fujiyoshi (2017).

References

Álvaro, F., Sánchez, J.-A., and Benedí, J.-M. Recognition of on-line handwritten mathematical expressions using 2D stochastic context-free grammars and hidden Markov models. *Pattern Recognition Letters*, 35(0):58 – 67, 2014. ISSN 0167-8655. Frontiers in Handwriting Processing.

Álvaro, F., Cruz, F., Sánchez, J.-A., Ramos Terrades, O., and Benedí, J.-M. Structure detection and segmentation of documents using 2D stochastic context-free grammars. *Neurocomputing*, 150(PA):147–154, February 2015. ISSN 0925-2312.

Anderson, R. H. Syntax-directed recognition of hand-printed two-dimensional mathematics. In *Symposium on Interactive Systems for Experimental Applied Mathematics: Proceedings of the Association for Computing Machinery Inc. Symposium*, pages 436–459, New York, NY, USA, 1967. ACM.

Billera, L. J. and Sarangarajan, A. All 0-1 polytopes are traveling salesman polytopes. *Combinatorica*, 16(2):175–188, 1996.

Boros, E. and Hammer, P. L. Pseudo-Boolean optimization. *Discrete Applied Mathematics*, 123(1-3):155–225, 2002. ISSN 0166-218X.

Boykov, Y., Veksler, O., and Zabih, R. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, November 2001. ISSN 0162-8828.

Bresler, M., Průša, D., and Hlaváč, V. Online recognition of sketched arrow-connected diagrams. *IJDAR*, 19(3):253–267, 2016.

Chekuri, C., Khanna, S., Naor, J., and Zosin, L. Approximation algorithms for the metric labeling problem via a new linear programming formulation. In *Symposium on Discrete Algorithms*, pages 109–118, 2001.

Dantzig, G. and Thapa, M. *Linear Programming 1: Introduction*. Springer, 1997.

De Loera, J. A. and Onn, S. All linear and integer programs are slim 3-way transportation programs. *SIAM Journal on Optimization*, 17(3):806–821, 2006.

Earley, J. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, February 1970. ISSN 0001-0782.

Eto, Y. and Suzuki, M. Mathematical formula recognition using virtual link network. In *Proceedings of Sixth International Conference on Document Analysis and Recognition*, pages 762–767, 2001.

Fahmy, H. and Blostein, D. A graph grammar programming style for recognition of music notation. *Machine Vision and Applications*, pages 83–99, 1993. ISSN 0932-8092.

Giammarresi, D. and Restivo, A. Two-dimensional languages. In Rozenberg, G. and Salomaa, A., editors, *Handbook of Formal Languages, Vol. 3*, pages 215–267. Springer, New York, 1997.

Globerson, A. and Jaakkola, T. Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations. In *Conference on Neural Information Processing Systems*, pages 553–560, 2008.

Hopcroft, J. E., Motwani, R., and Ullman, J. D. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006. ISBN 0321455363.

Ishikawa, H. Exact optimization for Markov random fields with convex priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(10): 1333–1336, October 2003. ISSN 0162-8828.

Johnson, J. K., Malioutov, D. M., and Willsky, A. S. Lagrangian relaxation for MAP estimation in graphical models. In *Allerton Conference on Communication, Control and Computing*, 2007.

Kappes, J. H., Andres, B., Hamprecht, F. A., Schnörr, C., Nowozin, S., Batra, D., Kim, S., Kausler, B. X., Kröger, T., Lellmann, J., Komodakis, N., Savchynskyy, B., and Rother, C. A comparative study of modern inference techniques for structured discrete energy minimization problems. *Int. J. Comput. Vision*, 115(2):155–184, November 2015. ISSN 0920-5691.

Kiyko, V. Recognition of objects in images of paper based line drawings. In *Third International Conference on Document Analysis and Recognition*, pages 970–973, Montreal, 1995.

Kolmogorov, V. Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1568–1583, 2006.

Kolmogorov, V. and Rother, C. Minimizing nonsubmodular functions with graph cuts-a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(7):1274–1279, 2007. ISSN 0162-8828.

Komodakis, N., Paragios, N., and Tziritas, G. MRF energy minimization and beyond via dual decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(3):531–552, 2011.

Koster, A., van Hoesel, S. P., and Kolen, A. W. The partial constraint satisfaction problem: Facets and lifting theorems. *Operations Research Letters*, 23(3–5):89–97, 1998.

Lavirotte, S. and Pottier, L. Mathematical formula recognition using graph grammar. In *Proceedings of the SPIE 1998*, volume 3305, pages 44–52, San Jose, CA, 1998.

Le, A. D., Van Phan, T., and Nakagawa, M. A system for recognizing online handwritten mathematical expressions and improvement of structure analysis. In *11th IAPR International Workshop on Document Analysis Systems (DAS), 2014*, pages 51–55, April 2014.

Lemaitre, A., Mouchère, H., Camillerapp, J., and Coüasnon, B. Interest of syntactic knowledge for on-line flowchart recognition. In *9th IAPR International Workshop on Graphics Recognition, 2011. GREC 2011*, pages 85–88, 2011.

Liang, P., Narasimhan, M., Shilman, M., and Viola, P. Efficient geometric algorithms for parsing in two dimensions. *International Conference on Document Analysis and Recognition*, pages 1172–1177, 2005. ISSN 1520-5263.

Liu, J. and Liu, Y. Local regularity-driven city-scale facade detection from aerial images. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pages 3778–3785. IEEE Computer Society, 2014. ISBN 978-1-4799-5118-5.

Matz, O. Regular expressions and context-free grammars for picture languages. In *In 14th Annual Symposium on Theoretical Aspects of Computer Science*, pages 283–294. Springer-Verlag, 1997.

Pradella, M., Cherubini, A., and Reghizzi, S. C. A unifying approach to picture grammars. *Information and Computation*, 209(9):1246 – 1267, 2011. ISSN 0890-5401.

Průša, D. Non-recursive trade-offs between two-dimensional automata and grammars. In Jürgensen, H., Karhumäki, J., and Okhotin, A., editors, *Descriptional Complexity of Formal Systems - 16th International Workshop, DCFS 2014, Turku, Finland, August 5-8, 2014. Proceedings*, volume 8614 of *Lecture Notes in Computer Science*, pages 352–363. Springer, 2014. ISBN 978-3-319-09703-9.

Průša, D. Graph-based simplex method for pairwise energy minimization with binary variables. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 475–483. IEEE Computer Society, 2015a. ISBN 978-1-4673-6964-0.

Průša, D. (Un)decidability of the emptiness problem for multi-dimensional context-free grammars. In Drewes, F., editor, *Implementation and Application of Automata - 20th International Conference, CIAA 2015, Umeå, Sweden, August 18-21, 2015, Proceedings*, volume 9223 of *Lecture Notes in Computer Science*, pages 250–262. Springer, 2015b. ISBN 978-3-319-22359-9.

Průša, D. Non-recursive trade-offs between two-dimensional automata and grammars. *Theoretical Computer Science*, 610:121–132, 2016.

Průša, D. and Fujiyoshi, A. Rank-reducing two-dimensional grammars for document layout analysis. In *14th International Conference on Document Analysis and Recognition (ICDAR 2017), 9-15 November, Kyoto, Japan*, pages 1120–1125, 2017.

Průša, D. and Werner, T. Universality of the local marginal polytope. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(4):898–904, 2015.

Průša, D. and Werner, T. LP relaxations of some np-hard problems are as hard as any LP. In Klein, P. N., editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, January 16-19*, pages 1372–1382. SIAM, 2017a. ISBN 978-1-61197-478-2.

Průša, D. and Werner, T. LP relaxation of the Potts labeling problem is as hard as any linear program. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(7):1469–1475, 2017b.

Průša, D. *Two-dimensional Languages*. PhD thesis, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic, 2004.

Průša, D. and Reinhardt, K. Undecidability of the emptiness problem for context-free picture languages. *Theoretical Computer Science*, 679 (Supplement C):118 – 125, 2017. ISSN 0304-3975. Implementation and Application of Automata.

Reghizzi, S. C. and Pradella, M. A CKY parser for picture grammars. *Information Processing Letters*, 105(6):213 – 217, 2008. ISSN 0020-0190.

Rother, C., Kolmogorov, V., Lempitsky, V. S., and Szummer, M. Optimizing binary MRFs via extended roof duality. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

Sadawi, N. M., Sexton, A. P., and Sorge, V. Chemical structure recognition: a rule-based approach. In Viard-Gaudin, C. and Zanibbi, R., editors, *Document Recognition and Retrieval XIX, part of the IS&T-SPIE Electronic Imaging Symposium, Burlingame, California, USA, January 25-26, 2012, Proceedings*, volume 8297 of *SPIE Proceedings*, page 82970E. SPIE, 2012. ISBN 978-0-819-48944-9.

Savchynsky, B., Schlesinger, M., and Anochina, M. Parsing and recognition of printed notes. In *Proceedings of the conference Control Systems and Computers*, pages 30–38, Kiev, Ukraine, 2003. in Russian, preprint in English available.

Schlesinger, D. and Flach, B. Transforming an arbitrary MinSum problem into a binary one. Technical Report TUD-FI06-01, Dresden University of Technology, Germany, April 2006.

Schlesinger, M. I. *Matematiceskie sredstva obrabotki izobrazenij, in Russian, (Mathematic tools for image processing).* Naukova Dumka, Kiev, 1989.

Schlesinger, M. I. and Hlaváč, V. *Ten Lectures on Statistical and Structural Pattern Recognition (Computational Imaging and Vision).* Springer, 1 edition, May 2012. ISBN 9048160278.

Schlesinger, M. I. Syntactic analysis of two-dimensional visual signals in noisy conditions. *Cybernetics and Systems Analysis*, 12(4):612–628, 1976. Translation from Russian.

Siromoney, G., Siromoney, R., and Krithivasan, K. Abstract families of matrices and picture languages. *Computer Graphics and Image Processing*, 1(3):284 – 307, 1972. ISSN 0146-664X.

Siromoney, G., Siromoney, R., and Krithivasan, K. Picture languages with array rewriting rules. *Information and Control*, 22(5):447 – 470, 1973. ISSN 0019-9958.

Stria, J. and Průša, D. Web application for recognition of mathematical formulas. In Lopatková, M., editor, *Proceedings of the Conference on Theory and Practice of Information Technologies, Vrátna Dolina, Slovak Republic, September 23-27, 2011*, volume 788 of *CEUR Workshop Proceedings*, pages 47–54. CEUR-WS.org, 2011.

Thapper, J. and Živný, S. The power of linear programming for valued CSPs. In *Symposium on Foundations of Computer Science (FOCS)*, pages 669–678. IEEE, 2012.

Tomita, M. Parsing 2-dimensional language. In Tomita, M., editor, *Current Issues in Parsing Technology*, pages 277–289. Springer US, Boston, MA, 1991. ISBN 978-1-4615-3986-5.

Wainwright, M. J. and Jordan, M. I. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.

Werner, T. A linear programming approach to max-sum problem: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(7): 1165–1179, 2007.

Younger, D. Recognition of context-free languages in time n^3. *Information and Control*, 10:189–208, 1967.

Zeng, J. and Liu, Z. Q. Markov random field-based statistical character structure modeling for handwritten Chinese character recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(5):767–780, May 2008. ISSN 0162-8828.

A Included Publications

As the rest of the thesis, the following publications are included (listed in the order they appear):

- Universality of the local marginal polytope (Průša and Werner, 2015),
- LP relaxation of the Potts labeling problem is as hard as any linear program (Průša and Werner, 2017b),
- Graph-based simplex method for pairwise energy minimization with binary variables (Průša, 2015a),
- On-line recognition of sketched arrow-connected diagrams (Bresler et al., 2016),
- Undecidability of the emptiness problem for context-free picture languages (Průša and Reinhardt, 2017),
- Non-recursive trade-offs between two-dimensional automata and grammars (Průša, 2016),
- Rank-reducing two-dimensional grammars for document layout analysis (Průša and Fujiyoshi, 2017).

Universality of the Local Marginal Polytope

Daniel Průša and Tomáš Werner

Abstract—We show that solving the LP relaxation of the min-sum labeling problem (also known as MAP inference problem in graphical models, discrete energy minimization, or valued constraint satisfaction) is not easier than solving any linear program. Precisely, every polytope is linear-time representable by a local marginal polytope and every LP can be reduced in linear time to a linear optimization (allowing infinite costs) over a local marginal polytope. The reduction can be done (though with a higher time complexity) even if the local marginal polytope is restricted to have a planar structure.

Index Terms—Graphical model, Markov random field, discrete energy minimization, valued constraint satisfaction, linear programming relaxation, local marginal polytope

1 Introduction

The *min-sum (labeling) problem* is defined as follows: given a set of discrete variables and a set of functions depending on one or two variables, minimize the sum of the functions over all variables. This problem arises in MAP inference in graphical models [22] and it is also known as discrete energy minimization [9] or valued constraint satisfaction [21].

This NP-complete problem has a natural linear programming (LP) relaxation, proposed by a number of authors [4], [13], [18], [22]. This relaxation is equivalent to the dual (Lagrangian) decomposition of the min-sum problem [8], [12], [19]. While the min-sum problem can be formulated as a linear optimization over the *marginal polytope*, the LP relaxation approximates this polytope by its outer bound, the *local marginal polytope* [22].

The relaxation is exact for a large class of min-sum instances and it is a basis for constructing good approximations for many other instances [9], [20], [23]. It is therefore of great practical interest to have efficient algorithms to solve the LP relaxation.

To solve the LP relaxation, the simplex and interior point methods are prohibitively inefficient for large-scale instances (which often occur, e.g., in computer vision). For min-sum problems with two labels, the LP relaxation can be solved efficiently because it reduces in linear time to max-flow [3], [17]. For more general problems, no really efficient algorithm is known to solve the LP.

In this paper we show that the quest for efficient algorithms to solve the LP relaxation of the general min-sum problem has a fundamental limitation, because this task is not easier than solving any linear program. Precisely, we prove the following theorems.

Theorem 1. *Every polytope is (up to scale) a coordinate-erasing projection of a face of a local marginal polytope with three labels, whose description can be computed from the input polytope in linear time.*

The input polytope is described by a set of linear inequalities with integer coefficients. By coordinate-erasing projection, we mean a projection that copies a subset of coordinates and erases the remaining ones.

Theorem 2. *Every linear program can be reduced in linear time to a linear optimization (allowing infinite costs) over a local marginal polytope with three labels.*

While Theorem 2 immediately follows from Theorem 1, the situation is more complex when infinite costs are not allowed. In this case, the reduction time and the output size are quadratic (see Theorem 9).

Given these negative results, one may ask whether the LP relaxation can be solved efficiently for some useful subclasses of the min-sum problem. One such subclass is the planar min-sum problem, which frequently occurs in computer vision. We show (in Theorem 11) that even in this case, the reduction can be done (with infinite costs allowed), in better than quadratic time.

Similar universality results are known also for other polytopes, e.g., the three-way transportation polytope [6] and the traveling salesman polytope [2].

2 The Local Marginal Polytope

Let (V, E) be an undirected graph, where V is a finite set of *objects* and $E \subseteq \binom{V}{2}$ is a set of object pairs. Let K be a finite set of *labels*. Let $g_u\colon K \to \overline{\mathbb{R}}$ and $g_{uv}\colon K \times K \to \overline{\mathbb{R}}$ be unary and binary *cost functions*, where $\overline{\mathbb{R}} = \mathbb{R} \cup \{\infty\}$ and we adopt that $g_{uv}(k,\ell) = g_{vu}(\ell,k)$. The *min-sum problem* is defined as

$$\min_{\mathbf{k}\in K^V} \left(\sum_{u\in V} g_u(k_u) + \sum_{\{u,v\}\in E} g_{uv}(k_u,k_v) \right). \tag{1}$$

All the costs $g_u(k), g_{uv}(k,\ell)$ form a vector $\mathbf{g} \in \overline{\mathbb{R}}^I$ where $I = (V \times K) \cup \{\{(u,k),(v,\ell)\} \,|\, \{u,v\} \in E;\ k,\ell \in K\}$. The problem instance is given by a tuple $(V, E, K, \mathbf{g})$.

The *local marginal polytope* [22] is the set Λ of vectors $\boldsymbol{\mu} \in \mathbb{R}_+^I$ satisfying

$$\sum_{\ell\in K} \mu_{uv}(k,\ell) = \mu_u(k), \quad u \in V,\ v \in N_u,\ k \in K, \tag{2a}$$

$$\sum_{k\in K} \mu_u(k) = 1, \quad u \in V, \tag{2b}$$

where $N_u = \{\, v \,|\, \{u,v\} \in E \,\}$ are the neighbors of u and we assume $\mu_{uv}(k,\ell) = \mu_{vu}(\ell,k)$. The numbers $\mu_u(k), \mu_{uv}(k,\ell)$ are known as *pseudomarginals* [22]. The local marginal polytope is given by a triplet (V, E, K).

The LP relaxation of the min-sum problem reads

$$\Lambda^*(\mathbf{g}) = \operatorname*{argmin}_{\boldsymbol{\mu}\in\Lambda} \langle \mathbf{g}, \boldsymbol{\mu} \rangle, \tag{3}$$

where in the scalar product $\langle \mathbf{g}, \boldsymbol{\mu} \rangle$ we define $0 \cdot \infty = 0$. The set (3) contains all vectors $\boldsymbol{\mu}$ for which $\langle \mathbf{g}, \boldsymbol{\mu} \rangle$ attains minimum over Λ. It is itself a polytope, a face of Λ.

We will depict min-sum problems by diagrams, as in Fig. 1. Objects $u \in V$ are depicted as boxes, labels $(u,k) \in I$ as nodes, label pairs $\{(u,k),(v,\ell)\} \in I$ as edges. Each node is assigned a unary pseudomarginal $\mu_u(k)$ and cost $g_u(k)$. Each edge is assigned a binary pseudomarginal $\mu_{uv}(k,\ell)$ and cost $g_{uv}(k,\ell)$.

Note the meaning of constraints (2) in Fig. 1. Constraint (2b) imposes for unary pseudomarginals a, b, c that $a + b + c = 1$. Constraint (2a) imposes for binary pseudomarginals p, q, r that $a = p + q + r$.

3 Input Polyhedron

We consider the input polyhedron in the form

$$P = \{\, \mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n \,|\, \mathbf{A}\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq \mathbf{0} \,\}, \tag{4}$$

where[1] $\mathbf{A} = [a_{ij}] \in \mathbb{Z}^{m\times n}$, $\mathbf{b} = (b_1, \ldots, b_m) \in \mathbb{Z}^m$, $m \leq n$. We assume there is at least one non-zero entry in each row and column of $\mathbf{A}$.

1. The assumption that $(\mathbf{A}, \mathbf{b})$ are integer-valued is common, see e.g., [10]. In the more general case of rational-valued $(\mathbf{A}, \mathbf{b})$, Lemma 4 would not hold. Linear complexity of the reduction could probably be maintained under some additional assumptions, such as prior bounds on the sizes of coordinates of the vertices of P.

• *The authors are with the Department of Cybernetics, Czech Technical University, Karlovo náměstí 13,12135 Praha, Czech Republic. E-mail: {prusapa1, werner}@cmp.felk.cvut.cz.*

Manuscript received 16 Aug. 2013; revised 15 July 2014; accepted 28 July 2014. Date of publication 28 Aug. 2014; date of current version 3 Mar. 2015.
Recommended for acceptance by C. H. Lampert.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPAMI.2014.2353626

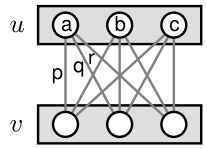


Fig. 1. A pair of objects $\{u, v\} \in E$ with $|K| = 3$ labels.

The instance of polyhedron (4) is given by $(\mathbf{A}, \mathbf{b})$ or, in short, by the extended matrix

$$\bar{\mathbf{A}} = [\overline{a}_{ij}] = [\mathbf{A} \,|\, \mathbf{b}] \in \mathbb{Z}^{m\times(n+1)}. \tag{5}$$

It will be convenient to rewrite the system $\mathbf{Ax} = \mathbf{b}$ as follows. In the ith equation

$$a_{i1}x_1 + \cdots + a_{in}x_n = b_i, \tag{6}$$

it is assumed that $b_i \geq 0$ (if not, multiply the equation by -1). Further, the terms with negative coefficients are moved to the right-hand side, such that both sides have only non-negative terms. Thus, (6) is rewritten as

$$a^+_{i1}x_1 + \cdots + a^+_{in}x_n = a^-_{i1}x_1 + \cdots + a^-_{in}x_n + b_i, \tag{7}$$

where $a^+_{ij} \geq 0$, $a^-_{ij} \geq 0$, $a_{ij} = a^+_{ij} - a^-_{ij}$. We assume w.l.o.g. that $a^+_{i1} + \cdots + a^+_{in} \neq 0$ and $a^-_{i1} + \cdots + a^-_{in} + b_i \neq 0$.

The following lemmas give some bounds that will be needed in the encoding algorithm.

Lemma 3. *For every matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ with columns $\mathbf{A}_j$,*

$$|\det \mathbf{A}| \leq \prod_{j=1}^{n} \|\mathbf{a}_j\|_2 \leq \prod_{j=1}^{n} \|\mathbf{a}_j\|_1.$$

Proof. The first inequality is well-known as Hadamard's inequality. The second inequality holds because $\|\mathbf{a}\|_2 \leq \|\mathbf{a}\|_1$ for every $\mathbf{a} \in \mathbb{R}^n$. □

Lemma 4. *Let $\mathbf{b} \neq \mathbf{0}$. Let $(x_1, \ldots, x_n)$ be a vertex of P. Then for each j we have $x_j = 0$ or $M^{-1} \leq x_j \leq M$ where*

$$M = \prod_{j=1}^{n+1} \sum_{i=1}^{m} |\overline{a}_{ij}|. \tag{8}$$

Proof. It is well-known from the theory of linear programming that every vertex $\mathbf{x}$ of P is a solution of a system $\mathbf{A}'\mathbf{x}' = \mathbf{b}'$, where $\mathbf{x}' = (x'_1, x'_2, \ldots)$ are the non-zero components of $\mathbf{x}$, $\mathbf{A}'$ is a non-singular submatrix of $\mathbf{A}$, and $\mathbf{b}'$ is a subvector of $\mathbf{b}$. By Cramer's rule,

$$x'_j = \frac{\det \mathbf{A}'_j}{\det \mathbf{A}'}, \tag{9}$$

where $\mathbf{A}'_j$ denotes $\mathbf{A}'$ with the jth column replaced by $\mathbf{b}'$. Lemma 3 implies $|\det \mathbf{A}'_j|, |\det \mathbf{A}'| \leq M$. □

Lemma 5. *Let P be bounded. Then for every $\mathbf{x} \in P$, each side of equation (7) is not greater than*

$$N = M \max_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|. \tag{10}$$

Proof. Since every point $(x_1, \ldots, x_n)$ of P is a convex combination of vertices of P, we have $x_j \leq M$ for each j. Hence, $a^+_{i1}x_1 + \cdots + a^+_{in}x_n \leq M(|a_{i1}| + \cdots + |a_{in}|) \leq N$ for each i. □

4 Encoding a Polytope

In this section, we prove Theorem 1 by constructing, in linear time, a min-sum problem $(V, E, K, \mathbf{g})$ with costs $\mathbf{g} \in \{0, 1\}^I$ such that the input polyhedron P is a scaled coordinate-erasing projection of $\Lambda^*(\mathbf{g})$. We assume that P is bounded, i.e., a polytope.[2]

4.1 Elementary Constructions

The output min-sum problem will be constructed from small building blocks, which implement certain simple operations on unary pseudomarginals. We call these blocks *elementary constructions*. An elementary construction is a min-sum problem with $|K| = 3$ labels, zero unary costs $g_u(k) = 0$, binary costs $g_{uv}(k, \ell) \in \{0, 1\}$, and optimal value $\min_{\mu\in\Lambda}\langle \mathbf{g}, \mu\rangle = 0$. It follows that $\mu \in \Lambda$ is optimal to the LP relaxation if and only if

$$g_{uv}(k, \ell)\, \mu_{uv}(k, \ell) = 0, \quad \{u, v\} \in E;\ k, \ell \in K. \tag{11}$$

We will define elementary constructions by diagrams such as in Fig. 1, in which we draw only edges with costs $g_{uv}(k, \ell) = 1$. Edges with costs $g_{uv}(k, \ell) = 0$ are not drawn. We will use the following elementary constructions (see Fig. 2):

Copy enforces equality of two unary pseudomarginals a, d in two objects while imposing no other constraints on b, c, e, f. Precisely, given any feasible unary pseudomarginals a, b, c, d, e, f, there exist feasible binary pseudomarginals satisfying (11) if and only if $a = d$.

Addition adds two unary pseudomarginals a, b in one object and represents the result as a unary pseudomarginal $c = a + b$ in another object. No other constraints are imposed on the remaining unary pseudomarginals.

Equality enforces equality of two unary pseudomarginals a, b in a single object, introducing two auxiliary objects. No other constraints are imposed on the remaining unary pseudomarginals. In the sequel, this construction will be abbreviated by omitting the two auxiliary objects and writing the equality sign between the two nodes, as shown in Fig. 2d.

Powers creates the sequence of unary pseudomarginals with values $2^i a$ for $i = 0, \ldots, d$, each in a separate object. We call d the *depth* of the pyramid.

NegPowers is similar to Powers but constructs values 2^{-i} for $i = 0, \ldots, d$.

Fig. 3 shows an example of how the elementary constructions can be combined. The edge colors distinguish different elementary constructions. By summing selected bits from NegPowers, the number $\frac{5}{8}$ is constructed. The example can be easily generalized to construct the value $2^{-d}k$ for any $d, k \in \mathbb{N}$ such that $2^{-d}k \leq 1$.

4.2 The Algorithm

Now we are ready to describe the encoding algorithm. The input of the algorithm is a set of equalities (7). Its output will be a min-sum problem $(V, E, K, \mathbf{g})$ with $|K| = 3$ labels and costs $g_u(k) = 0$, $g_{uv}(k, \ell) \in \{0, 1\}$. We will number labels and objects by integers, $K = \{1, 2, 3\}$ and $V = \{1, \ldots, |V|\}$.

The algorithm is initialized as follows:

1.1. For each variable x_j in (4), introduce a new object j into V. The variable x_j will be represented (up to scale) by pseudomarginal $\mu_j(1)$.

1.2. For each such object j, build Powers to the depth $d_j = \lfloor \log_2 \max_{i=1}^{m} |a_{ij}| \rfloor$ based on label 1. This yields the sequence of numbers $2^i \mu_j(1)$ for $i = 0, \ldots, d_j$.

1.3. Build NegPowers to the depth $d = \lceil \log_2 N \rceil$.

2. If the input polytope is in the general form $\{\, \mathbf{x} \in \mathbb{R}^n \,|\, \mathbf{Ax} \leq \mathbf{b} \,\}$, it can be transformed to the form (4) by adding slack variables and translating.

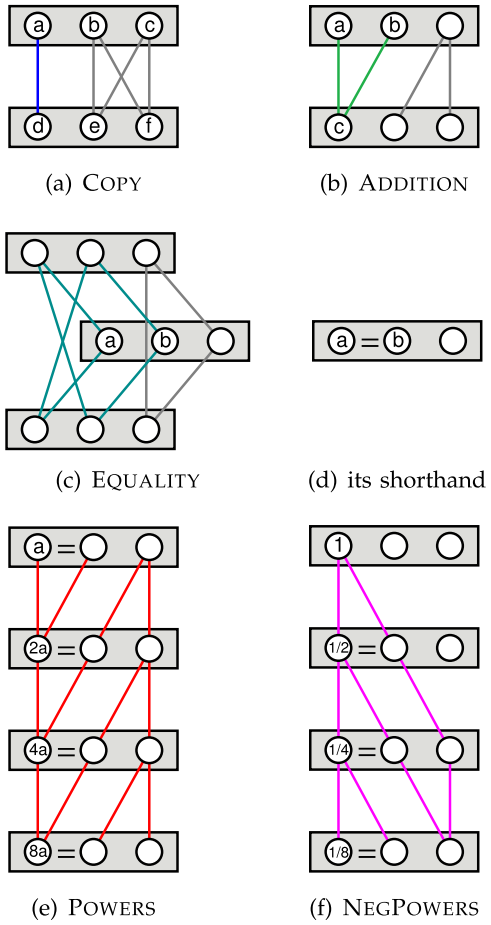


Fig. 2. Elementary constructions.

Then the algorithm proceeds by encoding each equation (7). The ith equation is encoded as follows:

2.1. Construct pseudomarginals with non-zero values $|a_{ij}|x_j$, $j = 1, \dots, n$, by summing selected values from POWERS built in Step 1.2, similarly as in Fig. 3. Note that the depths d_j are large enough to make this possible.

2.2. Construct a pseudomarginal with value $2^{-d}b_i$ by summing selected bits from NEGPOWERS built in Step 1.3, similarly as in Fig. 3. The value $2^{-d}b_i$ represents b_i, which sets the scale (mentioned in Theorem 1) between the input and output polytope to 2^{-d}. Note, the depth d is large enough to ensure that all pseudomarginals are bounded by 1.

2.3. Sum all the terms on each side of the equation by repetitively applying ADDITION and COPY.

2.4. Apply COPY to enforce equality of the two sides of the equation.

Fig. 4 shows the output min-sum problem for an example polytope P. By construction, the resulting min-sum problem encodes the input polytope as follows:

- If $P = \emptyset$ then $\min_{\boldsymbol{\mu} \in \Lambda} \langle \mathbf{g}, \boldsymbol{\mu} \rangle > 0$.
- If $P \neq \emptyset$ then $\min_{\boldsymbol{\mu} \in \Lambda} \langle \mathbf{g}, \boldsymbol{\mu} \rangle = 0$ and

$$P = \pi(\Lambda^*(\mathbf{g})), \tag{12}$$

where $\pi\colon \mathbb{R}^I \to \mathbb{R}^n$ is the scaled coordinate-erasing projection given by

$$(x_1, \dots, x_n) = \pi(\boldsymbol{\mu}) = 2^d(\mu_1(1), \dots, \mu_n(1)). \tag{13}$$

Let us make some remarks on this construction. The output min-sum problem has costs $\mathbf{g} \in \{0, 1\}^I$ but we could also use $\mathbf{g} \in \{0, \infty\}^I$ without affecting the result. The min-sum problem

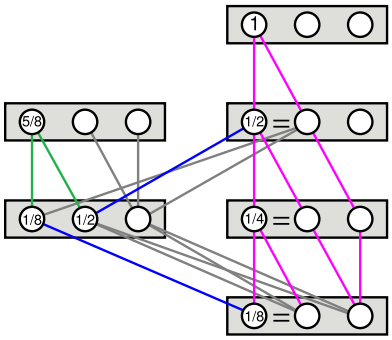


Fig. 3. Construction of the number $\frac{5}{8}$.

with costs in $\{0, \infty\}$ is well-known as the *constraint satisfaction problem* (CSP). An instance of CSP is *arc consistent* [1] if

$$\min_{\ell \in K} g_{uv}(k, \ell) = g_u(k), \qquad u \in V,\ v \in N_u,\ k \in K. \tag{14}$$

Our constructed min-sum problem is arc consistent.

Solving the LP relaxation of the problem $(V, E, K, \mathbf{g})$ decides whether $P \neq \emptyset$ and if so, it finds $\mathbf{x} \in P$. But this in fact means it solves the system $\{\, \mathbf{A}\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq \mathbf{0} \,\}$. Thus, we have the following side-result.

Theorem 6. *Solving any system of linear inequalities reduces in linear time to the LP relaxation of an arc consistent min-sum problem with three labels and costs in* $\{0, \infty\}$.

4.3 The Complexity of Encoding

Let us show that the running time of the algorithm in Section 4.2 is linear in the size of P, i.e., in the size of the matrix (5). It is usual (see e.g. [10]) to define the description size of a matrix as the number of bits needed to encode all its entries in binary. Since an integer $a \in \mathbb{Z}$ needs at least $\log_2(|a| + 1)$ bits to encode, the number

$$L_1 = \sum_{j=1}^{n+1} \sum_{i=1}^{m} \log_2(|\overline{a}_{ij}| + 1) \tag{15}$$

is a lower bound on the size of $\overline{\mathbf{A}}$. Now it suffices to show that the running time is $\mathcal{O}(L_1)$ because then it will clearly be linear also in the true size of P.

Note that zero entries $\overline{a}_{ij} = 0$ do not contribute to L_1. Thus L_1 is a lower bound on a *sparse* representation of $\overline{\mathbf{A}}$, in which only non-zero entries are stored.

The running time of the algorithm is obviously[3] linear in $|E|$. Object pairs are created only when an object is created and the number of object pairs added with one object is bounded by a constant, hence $|E| = \mathcal{O}(|V|)$. So it suffices to show that $|V| = \mathcal{O}(L_1)$.

On initialization, the algorithm creates $\sum_{j=1}^{n}(d_j + 1)$ objects in Step 1.2 and $d + 1$ objects in Step 1.3. It is easy to verify that both these numbers are $\mathcal{O}(L_1)$. To show that $d + 1 = \mathcal{O}(L_1)$, one needs to show (referring to (10)) that $\log_2 M = \mathcal{O}(L_1)$ and $\log_2 \max_i \sum_j |a_{ij}| = \mathcal{O}(L_1)$.

For illustration, we only prove $\log_2 M = \mathcal{O}(L_1)$ and leave the rest up to the reader. For every j, we have

$$\sum_{i=1}^{m} |\overline{a}_{ij}| \leq \prod_{i=1}^{m} (|\overline{a}_{ij}| + 1)$$

because multiplying out the left-hand side yields the right-hand side plus additional non-negative terms. Taking logarithm and

3. The only thing that may not be obvious is how to multiply large integers a, b in linear time. But this issue can be avoided by instead computing $p(a, b) = 2^{\lceil \log_2 a \rceil + \lceil \log_2 b \rceil}$, which can be done in linear time using bitwise operations. Since $ab \leq p(a, b) \leq (2a)(2b)$, the bounds like M become larger but this does not affect the overall complexity.

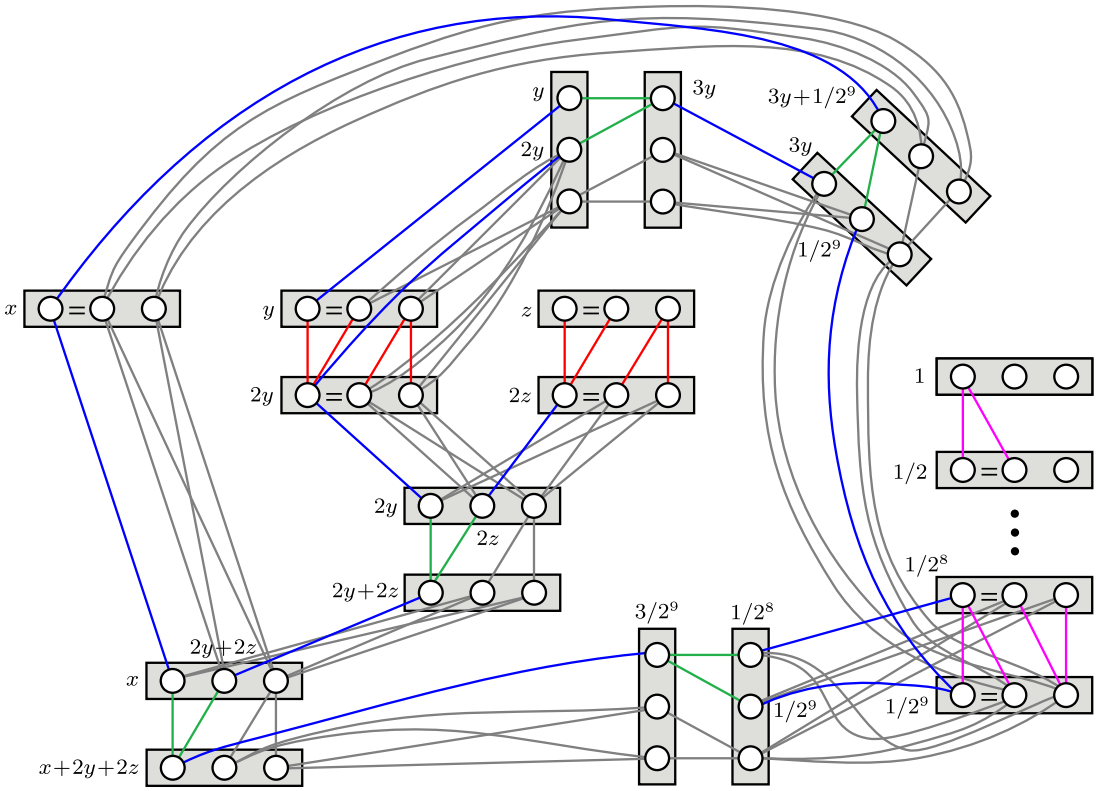


Fig. 4. The output min-sum problem for the polytope $P = \{\,(x, y, z) \,|\, x + 2y + 2z = 3;\ -x + 3y = -1;\ x, y, z \geq 0\,\}$.

summing over j yields

$$\log_2 M = \sum_{j=1}^{n+1} \log_2 \sum_{i=1}^{m} |\overline{a}_{ij}| \leq \sum_{j=1}^{n+1} \sum_{i=1}^{m} \log_2(|\overline{a}_{ij}| + 1) = L_1.$$

Finally, encoding one equality (7) adds at most as many objects as there are bits in the binary representation of all its coefficients. Thus, the number of objects added to encode all equalities (7) is $\mathcal{O}(L_1)$.

5 Encoding a Linear Program

Here we show how to reduce any linear program to linear optimization over a local marginal polytope. By saying that problem A reduces to problem B we mean there is an algorithm to solve problem A that can repeatedly[4] call an oracle for problem B (this is known as Turing reduction [15]). The complexity of the reduction is the complexity of this algorithm, assuming that the oracle for B takes constant time and space. If B is a linear program, we assume the oracle returns not only the optimal value but also an optimal argument.

We assume the input linear program in the form

$$P^*(\mathbf{c}) = \operatorname*{argmin}_{\mathbf{x} \in P} \langle \mathbf{c}, \mathbf{x} \rangle, \tag{16}$$

where $\mathbf{c} = (c_1, \ldots, c_n) \in \mathbb{Z}^n$. Since the encoding in Section 4 can be applied only to a bounded polyhedron but the LP (16) can be unbounded, we first need a lemma.

Lemma 7. *Every linear program can be reduced in linear time to a linear program over a bounded polyhedron.*

Proof. Denote $H(\alpha) = \{\,\mathbf{x} \in \mathbb{R}^n \,|\, \langle \mathbf{1}, \mathbf{x} \rangle \leq \alpha\,\}$. By Lemma 4, all vertices of P are contained in the halfspace $H(nM)$. Clearly,

$$\min_{\mathbf{x} \in P \cap H(nM)} \langle \mathbf{c}, \mathbf{x} \rangle \ \geq \ \min_{\mathbf{x} \in P \cap H(2nM)} \langle \mathbf{c}, \mathbf{x} \rangle. \tag{17}$$

Each side of (17) is a linear program over a bounded polyhedron. Inequality (17) is tight if and only if (16) is bounded, in which case (17) has the same optimum as (16). The linear programs (17) are infeasible if and only if (16) is infeasible.

The description size of numbers nM and $2nM$ is $\mathcal{O}(L_1)$, thus the reduction is done in linear time. □

By Lemma 7, we further assume that P is bounded. We also assume that $P \neq \emptyset$ because $P = \emptyset$ is indicated by $\min_{\boldsymbol{\mu} \in \Lambda} \langle \mathbf{g}', \boldsymbol{\mu} \rangle > 0$.

By Theorem 1, optimizing a linear function over P can be reduced in linear time to optimizing a linear function over a face of Λ. Given an oracle to optimize a linear function over Λ, it may seem unclear how to optimize a linear function over a *face* of Λ. This can be done by setting non-zero binary costs to a large constant.

Precisely, let $(V, E, K, \mathbf{g}')$ be the min-sum problem that encodes P, constructed in Section 4. Define $\mathbf{g} \in \overline{\mathbb{R}}^I$ by

$$g_i(k) = \begin{cases} c_i, & \text{if } k = 1 \text{ and } i \leq n, \\ 0, & \text{if } k > 1 \text{ or } i > n, \end{cases} \tag{18a}$$

$$g_{ij}(k, \ell) = \begin{cases} 0, & \text{if } g'_{ij}(k, \ell) = 0, \\ g_\infty, & \text{if } g'_{ij}(k, \ell) = 1, \end{cases} \tag{18b}$$

where the constant $g_\infty \geq 0$ is large enough to ensure that every $\boldsymbol{\mu} \in \Lambda^*(\mathbf{g})$ satisfies (11). It follows that

$$P^*(\mathbf{c}) = \pi(\Lambda^*(\mathbf{g})). \tag{19}$$

It remains to choose g_∞. The situation is different depending on whether or not we are allowed to use infinite costs. If infinite costs are allowed, we simply set $g_\infty = \infty$. This proves Theorem 2.

4. In our case, the oracle is called only twice, as given by Lemma 7.

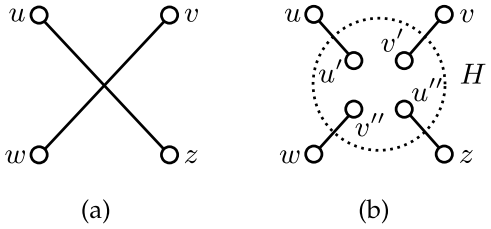


Fig. 5. Eliminating an edge crossing.

If infinite costs are not allowed, g_∞ must be large enough but finite. Unfortunately, manipulation with these large numbers increases the complexity of the reduction. This is given by Theorem 9. To prove it, we first need a lemma, which refines Lemma 4 for the special case of the local marginal polytope.

Lemma 8. *Let $\boldsymbol{\mu} \in \mathbb{R}^I$ be a vertex of the local marginal polytope defined by (V, E, K) with $|K| = 3$. Then each component μ of $\boldsymbol{\mu}$ satisfies $\mu = 0$ or $\mu \geq M_\Lambda^{-1}$ where*

$$M_\Lambda = 2^{|V|+6|E|}. \tag{20}$$

Proof. Write the local marginal polytope in the form (4), i.e., constraints (2) read $\mathbf{A}\mathbf{x} = \mathbf{b}$. Matrix $\mathbf{A}$ has $|V| + 6|E|$ rows and $3|V| + 9|E|$ columns. Each row of matrix $[\mathbf{A} \mid \mathbf{b}]$ has exactly 4 non-zeros, each of them in $\{-1, 1\}$. By Hadamard's inequality, in (9) we have $|\det \mathbf{A}'_j|, |\det \mathbf{A}'| \leq M_\Lambda$. □

Theorem 9. *Every linear program (16) can be reduced to a linear optimization (allowing only finite costs) over a local marginal polytope with three labels. The size of the output and the reduction time are $\mathcal{O}(L_1(L_1 + L_2))$ where L_2 is the description size of $\mathbf{c}$.*

Proof. Choose $g_\infty = 1 + M_\Lambda(C_2 - C_1)$ where

$$C_1 = \sum_{i=1}^{n} \min\{0, c_i\}, \qquad C_2 = \sum_{i=1}^{n} \max\{0, c_i\}.$$

We show that now every $\boldsymbol{\mu} \in \Lambda^*(\mathbf{g})$ satisfies (11). It suffices to show this only for vertices of $\Lambda^*(\mathbf{g})$ because taking convex combinations of vertices preserves (11).

Since $\boldsymbol{\mu} \in [0, 1]^I$, the contribution of the unary terms to $\langle \mathbf{g}, \boldsymbol{\mu} \rangle$ is in the interval $[C_1, C_2]$. Since $P \neq \emptyset$, we have $\min_{\boldsymbol{\mu} \in \Lambda} \langle \mathbf{g}', \boldsymbol{\mu} \rangle = 0$ and therefore $\min_{\boldsymbol{\mu} \in \Lambda} \langle \mathbf{g}, \boldsymbol{\mu} \rangle \leq C_2$.

Suppose there is a vertex $\boldsymbol{\mu}$ of $\Lambda^*(\mathbf{g})$ and a label pair $\{(u, k), (v, \ell)\}$ such that $g_{uv}(k, \ell) = g_\infty$ and $\mu_{uv}(k, \ell) > 0$. By Lemma 8, we have $\mu_{uv}(k, \ell) \geq M_\Lambda^{-1}$. Thus

$$\min_{\boldsymbol{\mu} \in \Lambda} \langle \mathbf{g}, \boldsymbol{\mu} \rangle \geq g_\infty M_\Lambda^{-1} + C_1 > C_2,$$

which is a contradiction.

Let us prove the claimed complexity. The binary length of g_∞ is $\mathcal{O}(L_1 + L_2)$. It occurs in $\mathbf{g}$ at $\mathcal{O}(L_1)$ positions, thus the binary length of $\mathbf{g}$ is $\mathcal{O}(L_1(L_1 + L_2))$. □

6 Reduction to Planar Min-Sum

In this section, we show that the reduction can be done even if we require the graph (V, E) of the output min-sum problem to be planar. For that, it suffices to modify the construction in Section 4.2 to ensure that (V, E) is planar.

Consider a drawing of the graph (V, E) in the plane, in which vertices are distinct points and edges are straight line segments connecting the vertices. We assume w.l.o.g. that no three edges intersect at a common point, except at graph vertices.

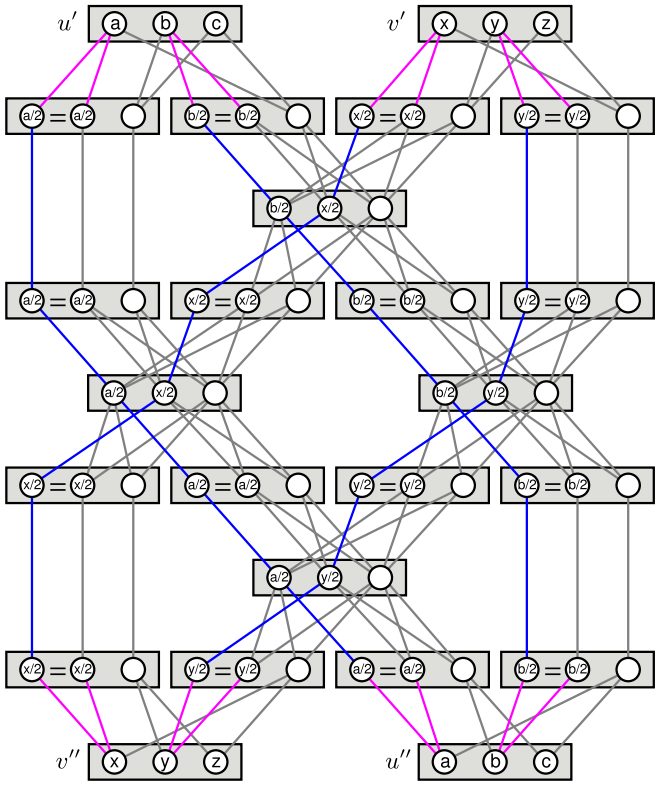


Fig. 6. Planar edge crossing using three labels.

The main idea is to replace every edge crossing with an equivalent planar min-sum problem. Consider a pair $\{u, z\}, \{v, w\} \in E$ of crossing edges, as shown in Fig. 5a. This pair is replaced by a construction in Fig. 5b. The cost functions $g_{uu'} = g_{vv'}$ copy unary pseudomarginals, i.e., they enforce $\mu_u = \mu_{u'}$ and $\mu_v = \mu_{v'}$. The other cost functions are set as $g_{u''z} = g_{uz}$ and $g_{v''w} = g_{vw}$. Problem H is a planar min-sum problem that enforces unary pseudomarginals in objects u', u'' and v', v'' to be equal, $\mu_{u'} = \mu_{u''}$ and $\mu_{v'} = \mu_{v''}$. This problem can be drawn arbitrarily small so that it is not intersected by any other edges.

Fig. 6 shows how the planar min-sum problem H can be designed. We work with halves of unary pseudomarginals, the first two from each object. The order of unary pseudomarginals is changed by swapping neighbors, imitating bubble sort on four elements.

Recall that the (non-planar) min-sum problem constructed in Section 4.2 has $E = \mathcal{O}(L_1)$ object pairs. Thus, there are $\mathcal{O}(L_1^2)$ edge crossings in this problem, which yields a reduction to a planar min-sum problem (allowing infinite costs) done in time $\mathcal{O}(L_1^2 + L_2)$.

It turns out that a more careful strategy of drawing the graph decreases the bound on edge crossings to $\mathcal{O}(mL_1)$. Before proving this in Theorem 11, we need a lemma.

Suppose we are given numbers $\alpha_1, \ldots, \alpha_p$ and sets $I_1, \ldots, I_q \subseteq \{1, \ldots, p\}$ and we want to compute numbers $\beta_j = \sum_{i \in I_j} \alpha_i$, $j = 1 \ldots, q$. The jth sum is constructed using a binary tree, T_j, in which every non-leaf vertex is the sum of its children (i.e., every non-leaf vertex with two children is Addition and every edge is Copy, as in Fig. 3). The leaves of T_j are α_i, $i \in I_j$, and its root is β_j. We refer to this construction as SumTrees.

Lemma 10. *Let SumTrees be drawn such that the leaves $\alpha_1, \ldots, \alpha_p$ lie on a common horizontal line and their positions on the line are given, and the roots $\beta_1, \ldots, \beta_q$ lie on a different horizontal line and their positions on the line can be arbitrary. Under this constraint, SumTrees can be drawn with $\mathcal{O}(q \sum_{j=1}^{q} |I_j|)$ edge crossings.*

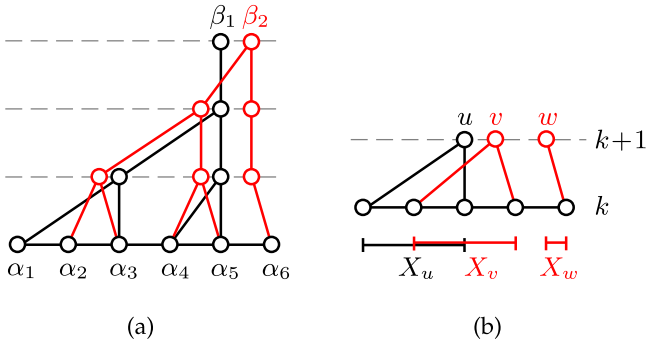


Fig. 7. (a) A drawing of SumTrees for $p=6$, $q=2$, $I_1=\{1,3,4,5\}$, $I_2=\{2,3,4,5,6\}$. (b) Crossing edges between two layers.

Proof. The construction is drawn as follows (see Fig. 7a). Each tree is drawn without edge crossings. In each tree T_j, all the leaves α_i, $i\in I_j$, have the same distance (i.e., the number of edges) to the root β_j. Let the *height* of a tree vertex be defined as its distance to the nearest leaf. The vertical coordinate of every non-root vertex is equal to its height. All the roots $\beta_1,\dots,\beta_q$ have the same vertical coordinate $h=\lceil\log_2\max_{j=1}^{q}|I_j|\rceil$.

Let us focus on tree T_1. It is built in the bottom-up manner. All non-leaf vertices with the same height have two children except the right-most one, which can have only one child. The horizontal coordinate of a vertex equals the horizontal coordinate of its second child; if there is only one child, it equals the horizontal coordinate of this child. When a layer containing only one vertex has been drawn and its height is less than h, the vertex is linked by a single vertical edge with the layer of height h (thus, this edge can jump over several layers), where it forms the root β_1. Clearly, adding this vertical edge does not affect the overall complexity.

The trees $T_2,\dots,T_q$ are drawn similarly. The only difference is that all non-leaf vertices are shifted to the left by a small offset, to ensure that the non-leaf vertices of all the trees are distinct.

We will show that the number of edge crossings between two trees T_i and T_j is $\mathcal{O}(|I_i|+|I_j|)$. Consider all vertices with heights k and $k+1$ (see Fig. 7b). For a vertex u with height $k+1$, let $X_u\subset\mathbb{R}$ denote the smallest interval containing the horizontal coordinates of u and its children. Edges going down from u and v to height k can cross each other only if the intervals X_u and X_v intersect. Note that if u and v belong to the same tree, then X_u and X_v are disjoint.

Let $q_{i,k}$ and $q_{j,k}$ be the number of vertices with height k of T_i and T_j, respectively. The number of pairs of intersecting intervals is $\mathcal{O}(q_{i,k}+q_{j,k})$. To see this, observe that if an interval is included in another, then it appears only in one intersecting pair. If all such included intervals are discarded, each interval intersects at most two others. Thus the number of intersections is $\mathcal{O}(q_{i,k}+q_{j,k})$.

It follows that the number of edge crossings between T_i and T_j is $\mathcal{O}(|T_i|+|T_j|)$, where $|T|$ denotes the number of vertices of tree T. But we have $|T_j|=\mathcal{O}(|I_j|)$, because $q_{j,k+1}=\lceil q_{j,k}/2\rceil$ for every j,k (recall, in every tree the highest non-root layer with a single node is linked with the root layer by a *single* edge).

The total number of crossings in the whole SumTrees graph is $\sum_{1\le i\ne j\le q}\mathcal{O}(|I_i|+|I_j|)=\mathcal{O}(q\sum_{j=1}^{q}|I_j|)$. □

Theorem 11. *Every linear program can be reduced in $\mathcal{O}(mL_1+L_2)$ time to a linear optimization (allowing infinite costs) over a local marginal polytope with three labels over a planar graph.*

Proof. It suffices to show how to draw, in the algorithm from Section 4.2, the graph (V,E) with $\mathcal{O}(mL_1)$ edge crossings. We show this in the rest of the proof.

We start by drawing Powers for variable x_1 horizontally. Then we draw SumTrees over the objects of Powers, with roots being non-zero numbers $|a_{i1}|x_1$, $i=1,\dots,m$. The ith tree has $\mathcal{O}(\log_2(|a_{i1}|+1))$ leaves, therefore, by Lemma 10, this SumTrees construction has $\mathcal{O}(m\sum_{i=1}^{m}\log_2(|a_{i1}|+1))$ edge crossings.

This is repeated for the remaining variables $x_2,\dots x_n$, resulting in n independent SumTrees constructions. The numbers $2^{-d}b_i$, $i=1,\dots,m$, are constructed similarly, by drawing SumTrees over NegPowers. The total number of edge crossings is

$$\mathcal{O}\Big(\sum_{j=1}^{n} m\sum_{i=1}^{m}\log_2(|a_{ij}|+1)+m\sum_{i=1}^{m}\log_2(|b_i|+1)\Big)=\mathcal{O}(mL_1).$$

At this stage, we have objects representing all non-zero numbers $|a_{ij}|x_j$ and $2^{-d}b_i$. We assume that the vertical positions of all SumTrees were such that all these objects lie on a single horizontal line. Now we proceed to sum the terms of each side of each equality (7). This is done by drawing SumTrees over these objects, with $2m$ roots being the left-hand and right-hand sides of all equalities (7). The tree associated with any side of the ith equality (7) has $\mathcal{O}(n_i)$ leaves, where n_i is the number of non-zeros in the ith row of $\mathbf{A}$. Therefore, the number of edge crossings is $\mathcal{O}(m\sum_{i=1}^{m}n_i)=\mathcal{O}(mL_1)$.

At this stage, all objects representing both sides of all equalities (7) lie on a common horizontal line. It remains to join corresponding left- and right-hand sides using Copy. This creates $\mathcal{O}(m^2)\subseteq\mathcal{O}(mL_1)$ edge crossings. □

7 Consequences

Let us discuss some consequences of our results.

Most importantly, our results show that solving the LP relaxation of the min-sum problem is comparably hard as solving any LP. This is straightforward if infinite costs are allowed. Then, by Theorem 2, the reduction is done in time $\mathcal{O}(L)$ where $L=L_1+L_2$, while the best known algorithm [10] for general LP has time complexity[5] $\mathcal{O}(n^{3.5}L^2\log L\log\log L)$. Finding a very fast algorithm, such as $\mathcal{O}(L^2\log L)$, to solve the LP relaxation would imply improving the best-known complexity of LP, which is unlikely.

The cases in which the reduction time is polynomial but higher than linear (Theorems 11 and 9) still impose a restriction on possible search for an efficient algorithm to solve the LP relaxation. There are not many principles how to solve the general LP in polynomial time (one is the ellipsoid algorithm), and finding a new such principle is expected to be difficult. Therefore, we should restrict our search to modifying these known principles rather than to discovering a new principle.

Our results make more precise the known observation that the LP relaxation of the min-sum problem is easier for two labels than for the general case. It is known that for two labels the LP relaxation reduces in linear time to max-flow [3], [17] and the local marginal polytope has half-integral vertices [11], [23]. For three labels, the coordinates of the vertices of local marginal polytopes can have much more general values, as shown in Section 4.1. Moreover, there is not much difference in complexity between the LP relaxation for three labels and for more than three labels (allowing infinite costs) because, by Theorem 2, the latter can be reduced to the former in linear time.

Rather than solving directly the LP relaxation (3), it is often more desirable to solve its dual. The dual seeks to maximize a lower bound on (1) by reparameterizations. One class of algorithms to tackle this dual LP converges only to its local minimum,

5. Note, Karmarkar [10] assumes full encoding of the LP matrix but we allow sparse encoding (see Section 4.3). To the best of our knowledge, the complexity of solving sparse LPs is largely open [16].

characterized by arc consistency. This class includes popular message passing algorithms [23, Section 6], [11], [7] and the algorithms [14], [23, Section 7], [5]. Theorem 6 has an interesting consequence. Suppose we are given a fixed point of say min-sum diffusion [23, Section 6] and want to decide whether it is (globally) optimal to the dual LP relaxation and if so, find a corresponding optimal solution to the primal LP (3). This problem is equivalent to the LP relaxation of an arc consistent min-sum problem with costs in $\{0, \infty\}$, therefore it is as hard as solving the general system of linear inequalities.

Acknowledgments

The authors were supported by the Czech Science Foundation grant P202/12/2071. Besides, Tomáš Werner was supported by the European Commission grant FP7-ICT-270138.

References

[1] C. Bessiere, "Constraint propagation," in *Handbook of Constraint Programming*. Amsterdam, The Netherlands: Elsevier, 2006, ch. 3.
[2] L. J. Billera and A. Sarangarajan, "All 0-1 polytopes are traveling salesman polytopes," *Combinatorica*, vol. 16, no. 2, pp. 175–188, 1996.
[3] E. Boros and P. L. Hammer, "Pseudo-Boolean optimization," *Discrete Appl. Math.*, vol. 123, nos. 1–3, pp. 155–225, 2002.
[4] C. Chekuri, S. Khanna, J. Naor, and L. Zosin, "Approximation algorithms for the metric labeling problem via a new linear programming formulation," in *Proc. 12th Annu. Symp. Discrete Algorithms*, 2001, pp. 109–118.
[5] M. C. Cooper, S. de Givry, M. Sanchez, T. Schiex, M. Zytnicki, and T. Werner, "Soft arc consistency revisited," *Artif. Intell.*, vol. 174, nos. 7/8, pp. 449–478, 2010.
[6] J. A. De Loera and S. Onn, "All linear and integer programs are slim 3-way transportation programs," *SIAM J. Optim.*, vol. 17, no. 3, pp. 806–821, 2006.
[7] A. Globerson and T. Jaakkola, "Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations," in *Proc. 21st Annu. Conf. Neural Inf. Process. Syst.*, 2008, pp. 553–560.
[8] J. K. Johnson, D. M. Malioutov, and A. S. Willsky, "Lagrangian relaxation for MAP estimation in graphical models," in *Proc. Allerton Conf. Commun., Control Comput.*, 2007.
[9] J. H. Kappes, B. Andres, F. A. Hamprecht, C. Schnörr, S. Nowozin, D. Batra, S. Kim, B. X. Kausler, J. Lellmann, N. Komodakis, and C. Rother, "A comparative study of modern inference techniques for discrete energy minimization problem," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2013, pp. 1328–1335.
[10] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *Proc. 16th Anuu. ACM Symp. Theory Comput.*, 1984, pp. 302–311.
[11] V. Kolmogorov, "Convergent tree-reweighted message passing for energy minimization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 10, pp. 1568–1583, Oct. 2006.
[12] N. Komodakis, N. Paragios, and G. Tziritas, "MRF energy minimization and beyond via dual decomposition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 3, pp. 531–552, Mar. 2011.
[13] A. Koster, S. P. van Hoesel, and A. W. Kolen, " The partial constraint satisfaction problem: Facets and lifting theorems," *Oper. Res. Lett.*, vol. 23, nos. 3–5, pp. 89–97, 1998.
[14] V. K. Koval and M. I. Schlesinger, "Dvumernoe programmirovanie v zadachakh analiza izobrazheniy (Two-dimensional programming in image analysis problems)," *USSR Acad. Sci., Autom. Telemech.*, vol. 8, pp. 149–168, 1976, in Russian.
[15] C. M. Papadimitriou, *Computational Complexity*. Reading, MA, USA: Addison-Wesley, 1994.
[16] P. M. Pardalos and S. A. Vavasis, "Open questions in complexity theory for numerical optimization," *Math. Program.*, vol. 57, pp. 337–339, 1992.
[17] C. Rother, V. Kolmogorov, V. S. Lempitsky, and M. Szummer, "Optimizing binary MRFs via extended roof duality," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2007, pp. 1–8.
[18] M. I. Shlezinger, "Syntactic analysis of two-dimensional visual signals in noisy conditions," *Cybern. Syst. Anal.*, vol. 12, no. 4, pp. 612–628, 1976.
[19] D. Sontag, A. Globerson, and T. Jaakkola, "Introduction to dual decomposition for inference," in *Optimization for Machine Learning*, Cambridge, MA, USA: MIT Press, 2011.
[20] J. Thapper and S. Živný, "The power of linear programming for valued CSPs," in *Proc. Symp. Found. Comput. Sci.*, 2012, pp. 669–678.
[21] S. Živný, *The Complexity of Valued Constraint Satisfaction Problems*. New York, NY, USA: Springer, 2012.
[22] M. J. Wainwright and M. I. Jordan, "Graphical models, exponential families, and variational inference," *Found. Trends Mach. Learn.*, vol. 1, nos. 1/2, pp. 1–305, 2008.
[23] T. Werner, "A linear programming approach to max-sum problem: A review," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 7, pp. 1165–1179, Jul. 2007.

LP Relaxation of the Potts Labeling Problem Is as Hard as Any Linear Program

Daniel Průša and Tomáš Werner

Abstract—In our recent work, we showed that solving the LP relaxation of the pairwise min-sum labeling problem (also known as MAP inference in graphical models or discrete energy minimization) is not much easier than solving any linear program. Precisely, the general linear program reduces in linear time (assuming the Turing model of computation) to the LP relaxation of the min-sum labeling problem. The reduction is possible, though in quadratic time, even to the min-sum labeling problem with planar structure. Here we prove similar results for the pairwise min-sum labeling problem with attractive Potts interactions (also known as the uniform metric labeling problem).

Index Terms—Markov random field, graphical model, MAP inference, discrete energy minimization, valued constraint satisfaction, linear programming relaxation, uniform metric labeling problem, Potts model

1 Introduction

The *pairwise min-sum (labeling) problem* consists in minimizing a sum of unary and binary (also called pairwise) cost functions of discrete variables. It is also known as (pairwise) discrete energy minimization [1], [2], valued constraint satisfaction [3], or MAP inference in graphical models [4]. It has many applications in computer vision, machine learning, and other fields. This NP-hard problem has a natural linear programming (LP) relaxation [4], [5], [6], [7], [8], which underlies many successful algorithms to tackle the problem (see [2] and the references therein). Therefore it would have a great practical impact to have efficient algorithms to solve this LP relaxation. The popular simplex and interior point methods are prohibitively inefficient for large instances of the LP relaxation, often arising, e.g., in computer vision. Our recent work [9] showed that, unfortunately, solving the LP relaxation of the pairwise min-sum problem with three variable states (labels) is as hard as solving the general linear program. Precisely, the latter reduces to the former in linear time, assuming the Turing model of computation. Therefore it is unlikely that a very efficient algorithm for the LP relaxation exists.

This negative result suggests the question whether there are any interesting subclasses of the min-sum problem for which the LP relaxation is easier than the general LP and thus there is a hope for efficient algorithms. One such subclass is the pairwise min-sum problem with two labels, for which the LP relaxation has half-integral solutions and reduces in linear time to max-flow [10], [11]. Thus the LP relaxation can be solved very efficiently because the complexity of best known algorithms for max-flow is much better than for the general LP.

Another subclass is the *metric labeling problem* [12], [13], [14], a pairwise min-sum problem in which the pairwise cost functions satisfy the axioms of a metric. An important special case is the *uniform metric*, in statistical physics known as the *attractive Potts interaction*. We refer to the pairwise min-sum problem with attractive Potts interactions as the *Potts (labeling) problem*. The LP relaxation for this (still NP-hard) problem was proposed in [13] and later generalized to any metric in [14]. For the Potts problem, the LP relaxation [4], [5], [6], [7], [8] coincides with that in [13], [14].

The LP relaxation is the basis for approximation algorithms to the metric labeling problem with theoretical approximation guarantees, in particular for the uniform metric where the approximation ratio is most favorable [13], [14], [15]. There is another class of approximation algorithms for metric labeling problems, α-expansion algorithms [12], which call a max-flow solver a small number of times and thus they are very efficient. They achieve comparable worst-case approximation guarantees [16] but the algorithms based on LP relaxation are often more accurate in practice [1], [2]. Moreover, for the multiway cut problem, closely related to the Potts labeling problem, the LP relaxation is the only known way to achieve the best possible approximation [17].

In this article, we show that solving the LP relaxation is hard even for the Potts labeling problem. Precisely, the general linear program can be reduced in linear time to the LP relaxation of the Potts labeling problem with three labels (Theorem 4). Unlike in [9] where the input LP is directly encoded by a min-sum problem, we proceed in a different way. By duality, the LP problem is linear-time equivalent to the linear feasibility (LF) problem (Lemma 3), therefore it suffices to construct a reduction from LF. We do this in two steps: first LF with rational coefficients is reduced to LF with coefficients in $\{-1, 0, 1\}$ by algebraic manipulations (Section 3) and then this problem is reduced to the LP relaxation of the Potts problem (Section 4).

This construction allows us to strengthen the result from [9] for general min-sum problem because infinite costs are no longer needed to achieve linear time. It allows us to formulate several other results. As in [9], the reduction has a polyhedral formulation (Theorem 5): any polytope is linear-time representable as a face of the feasible set of the LP relaxation [13] of a Potts problem, which we call the *relaxed Potts polytope*. We show (Theorem 8) that the reduction to the LP relaxation of the Potts problem can be also understood as a reduction to the LP relaxation of the *multiway cut problem* [17]. Finally, again similarly to [9], the reduction can be modified such that the output Potts problem is planar, but this needs more than linear time (Theorem 9).

2 LP Relaxation of Min-Sum Problem

The pairwise min-sum (labeling) problem is defined as

$$\min_{k \in K^V} \left(\sum_{u \in V} g_u(k_u) + \sum_{\{u,v\} \in E} g_{uv}(k_u, k_v) \right), \tag{1}$$

where (V, E) is a graph with V a finite set of *objects* and $E \subseteq \binom{V}{2}$ a set of *object pairs*, K is a finite set of *labels*, and $g_u\colon K \to \mathbb{R}$ and $g_{uv}\colon K \times K \to \mathbb{R}$ are unary and pairwise *cost functions*, adopting that $g_{uv}(k, \ell) = g_{vu}(\ell, k)$.

The LP relaxation of this problem reads

$$\min_{\mu \in \Lambda} \langle \mathbf{g}, \boldsymbol{\mu} \rangle, \tag{2}$$

where $\mathbf{g} \in \mathbb{R}^I$ and $\boldsymbol{\mu} \in \mathbb{R}^I$ is the vector with components $g_u(k), g_{uv}(k, \ell)$ and $\mu_u(k), \mu_{uv}(k, \ell)$, respectively, and

$$I = (V \times K) \cup \{ \{(u, k), (v, \ell)\} \,|\, \{u, v\} \in E,\ k, \ell \in K \}.$$

The set $\Lambda \subseteq \mathbb{R}^I$ contains all vectors $\boldsymbol{\mu} \geq \mathbf{0}$ satisfying

$$\sum_{\ell \in K} \mu_{uv}(k, \ell) = \mu_u(k), \qquad u \in V,\ v \in N_u,\ k \in K, \tag{3a}$$

$$\sum_{k \in K} \mu_u(k) = 1, \qquad u \in V, \tag{3b}$$

• *The authors are with the Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University, Karlovo náměstí 13, Praha 12135, Czech Republic. E-mail: {prusapa1, werner}@fel.cvut.cz.*

Manuscript received 10 Feb. 2016; revised 3 June 2016; accepted 6 June 2016. Date of publication 19 June 2016; date of current version 12 June 2017.
Recommended for acceptance by S. Nowozin.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPAMI.2016.2582165

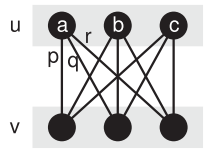


Fig. 1. One object pair $\{u,v\} \in E$ with $|K| = 3$ labels. Objects $u, v \in V$ are depicted as boxes, labels $(u,k) \in I$ as nodes, and label pairs $\{(u,k),(v,\ell)\} \in I$ as edges. Note the meaning of constraints (3): for unary pseudomarginals a, b, c and pairwise pseudomarginals p, q, r, equality (3a) reads $a = p + q + r$ and equality (3b) reads $a + b + c = 1$.

where $N_u = \{\, v \,|\, \{u,v\} \in E \,\}$ denotes the neighbors of object u. Following [4], we refer to Λ as the *local marginal polytope* and to $\mu_u(k), \mu_{uv}(k,\ell)$ as *pseudomarginals*. The meaning of constraints 3a is illustrated in Fig. 1.

A *reparameterization* of a cost vector $\mathbf{g} \in \mathbb{R}^I$ is a cost vector $\mathbf{g}' \in \mathbb{R}^I$ given by

$$g'_u(k) = g_u(k) - \sum_{v \in N_u} \varphi_{uv}(k) \tag{4a}$$

$$g'_{uv}(k,\ell) = g_{uv}(k,\ell) + \varphi_{uv}(k) + \varphi_{vu}(\ell), \tag{4b}$$

where $\varphi_{uv}(k) \in \mathbb{R}$ ($u \in V$, $v \in N_u$, $k \in K$). Reparameterizations preserve $\langle \mathbf{g}, \boldsymbol{\mu} \rangle$ for every $\boldsymbol{\mu}$ satisfying (3).

2.1 Potts Labeling Problem

Problem (1) in which pairwise cost functions g_{uv} satisfy metric axioms has been called the *metric labeling problem* [12], [13], [14], [15]. Its special case is obtained for the *uniform metric* (the *attractive Potts interaction*)

$$g_{uv}(k,\ell) = h_{uv} [\![k \neq \ell]\!], \tag{5}$$

where $h_{uv} \geq 0$, and $[\![k \neq \ell]\!] = 1$ if $k \neq \ell$ and $[\![k \neq \ell]\!] = 0$ if $k = \ell$. We refer to problem (1) with pairwise costs (5) as the *Potts (labeling) problem*.

In this case, problem (2) can be simplified [14] by minimizing out the pairwise pseudomarginals μ_{uv}. For a fixed object pair $\{u,v\} \in E$, minimizing $\langle g_{uv}, \mu_{uv} \rangle$ over $\mu_{uv} \geq 0$ subject to (3a) is a discrete transportation problem with transport costs g_{uv}. If g_{uv} has the form (5), the optimal value of this problem is given explicitly as $\frac{1}{2} h_{uv} \sum_{k \in K} |\mu_u(k) - \mu_v(k)|$. Therefore (2) is equivalent to minimizing

$$\sum_{u \in V} \sum_{k \in K} g_u(k) \mu_u(k) + \sum_{\{u,v\} \in E} \frac{1}{2} h_{uv} \sum_{k \in K} |\mu_u(k) - \mu_v(k)|, \tag{6}$$

over unary pseudomarginals $\mu_u(k) \geq 0$ subject to (3b). This is the relaxation of the Potts problem proposed by Kleinberg and Tardos [13]. It can be written also as

$$\min_{\nu \in \Pi} \langle \mathbf{h}, \nu \rangle, \tag{7}$$

where $\mathbf{h} \in \mathbb{R}^{(V \times K) \cup E}$ is the vector with components $h_u(k) = g_u(k)$ and h_{uv}, and Π is the set of all vectors $\nu \in [0,1]^{(V \times K) \cup E}$ with components $\nu_u(k), \nu_{uv}$ satisfying

$$\sum_{k \in K} |\nu_u(k) - \nu_v(k)| \leq 2\nu_{uv}, \quad \{u,v\} \in E, \tag{8a}$$

$$\sum_{k \in K} \nu_u(k) = 1, \quad u \in V. \tag{8b}$$

We will refer to Π as the *relaxed Potts polytope*.

3 Input Polyhedron

Our key construction in the paper will be a linear-time representation of any convex polyhedron as the optimal set of the LP relaxation of a Potts problem. We assume the input polyhedron in the form

$$P = \{\, \mathbf{x} \in \mathbb{R}^n \,|\, \mathbf{A}\mathbf{x} = \mathbf{0},\ x_n = 1,\ \mathbf{x} \geq \mathbf{0} \,\}, \tag{9}$$

where $\mathbf{A} = [a_{ij}] \in \mathbb{Q}^{m \times n}$ and x_n denotes the last component of the vector $\mathbf{x} = (x_1, \ldots, x_n)$. Note that the equation $x_n = 1$ makes the homogeneous linear system $\mathbf{A}\mathbf{x} = \mathbf{0}$ non-homogeneous, with the right-hand side being the negative last column of $\mathbf{A}$. Each row and column of $\mathbf{A}$ is assumed to have at least one non-zero.

By 'linear time' we mean time $O(N)$ where N is the size of the input, i.e., the number of bits needed to encode matrix $\mathbf{A}$ in binary. That is, we assume the Turing model of computation. Let us define the size of a matrix precisely. For a scalar $a \in \mathbb{Q}$, we define

$$\operatorname{size}(a) = \log_2(|pq| + 1), \tag{10}$$

where $p, q \in \mathbb{Z}$ are such that $a = p/q$ assuming that q does not divide p unless $q = 1$ or $p = 0$. For a matrix $\mathbf{A} \in \mathbb{Q}^{m \times n}$, we define

$$\operatorname{size}(\mathbf{A}) = \sum_{j=1}^{n} \sum_{i=1}^{m} \operatorname{size}(a_{ij}). \tag{11}$$

As $\operatorname{size}(a) = 0$ for $a = 0$, (11) underestimates the true size of matrix $\mathbf{A}$ by neglecting the space needed, e.g., for storing the indices of zero entries. This does not matter because if the time of an algorithm is linear in $\operatorname{size}(\mathbf{A})$, it is at most linear in the true size of $\mathbf{A}$. On the contrary, not counting zero entries makes our results stronger because it allows for a sparse representation of $\mathbf{A}$.

In the rest of this section, we transform the description (9) of the input polyhedron by algebraic manipulations to a form suitable for encoding by a Potts problem.

3.1 From Rationals to Integers

First, the homogeneous linear system $\mathbf{A}\mathbf{x} = \mathbf{0}$ in (9) with rational coefficients is transformed to a linear system with integer coefficients.[1] For each non-zero input coefficient $a_{ij} = p_{ij}/q_{ij} \in \mathbb{Q}$ with $p_{ij}, q_{ij} \in \mathbb{Z}$, we create an auxiliary variable y_{ij} and the equation

$$|q_{ij}| y_{ij} = |p_{ij}| x_j. \tag{12}$$

Then in the input system we replace every non-zero term $a_{ij} x_k$ with $\operatorname{sgn}(a_{ij}) y_{ij}$. The size of the output is clearly linear in the size of the input.[2]

Example 1. The system

$$\begin{aligned} \tfrac{2}{7}x_1 + \tfrac{3}{5}x_2 - 2x_3 &= 0 \\ \tfrac{7}{3}x_1 - \tfrac{1}{2}x_2 &= 0, \end{aligned}$$

is transformed to the system

$$\begin{aligned} 2x_1 &= 7y_{11} & 3x_2 &= 5y_{12} & 2x_3 &= y_{13} \\ 7x_1 &= 3y_{21} & x_2 &= 2y_{22} & & \\ & & y_{11} + y_{12} - y_{13} &= 0 & & \\ & & y_{21} - y_{22} &= 0. & & \end{aligned}$$

3.2 From Integers to $\{-1, 0, 1\}$

The system $\mathbf{A}\mathbf{x} = \mathbf{0}$ with integer coefficients $\mathbf{A} \in \mathbb{Z}^{m \times n}$ is now transformed in linear time to a homogeneous system with coefficients in $\{-1, 0, 1\}$.

Instead of the usual $(x_1, \ldots, x_n)$, let us name the input variables $(x_{10}, \ldots, x_{n0})$. The key idea is similar to [18, Section 3.1]. Suppose

1. In [9] we assumed that the input LP has integer coefficients, in other words, this step was omitted.

2. Note that the most obvious reduction, multiplying each equation by the least common multiple of the denominators, would take more than linear time.

we want to construct a product $a_{ij}x_{j0}$ for some $a_{ij} \in \mathbb{N}$. Create the equation system

$$\begin{array}{ll} x_{j1} = x_{j0} + y_{j0} & y_{j0} = x_{j0} \\ x_{j2} = x_{j1} + y_{j1} & y_{j1} = x_{j1} \\ \vdots & \vdots \\ x_{j,d_j} = x_{j,d_j-1} + y_{j,d_j-1} & y_{j,d_j-1} = x_{j,d_j-1} \end{array} \tag{13}$$

The first line of this system enforces $x_{j1} = 2x_{j0}$, the second line enforces $x_{j2} = 2x_{j1}$, etc. Consequently,

$$x_{jk} = 2^k x_{j0}. \tag{14}$$

The product $a_{ij}x_{j0}$ can be now obtained by summing appropriate bits of the binary code of a_{ij}. E.g., $11x_{j0} = x_{j0} + x_{j1} + x_{j3}$ because $11 = 2^0 + 2^1 + 2^3$.

The whole reduction proceeds as follows:

1) For each $j = 1, \dots, n$, create equation system 13 with $d_j = \lfloor \log_2 \max_{i=1}^m |a_{ij}| \rfloor$.
2) For each $i = 1, \dots, m$, construct non-zero terms $a_{ij}x_{j0}$, sum them, and equate the result to zero.

It is easy to verify that the number of non-zero output terms is linear in $\text{size}(\mathbf{A})$.

Example 2. The system

$$\begin{aligned} 2x_{10} + 11x_{20} - 3x_{30} + x_{40} &= 0 \\ 3x_{10} + 6x_{20} - 5x_{40} &= 0 \end{aligned}$$

is transformed to the system

$$\begin{array}{ll} x_{11} = x_{10} + y_{10} & y_{10} = x_{10} \\ x_{21} = x_{20} + y_{20} & y_{20} = x_{20} \\ x_{22} = x_{21} + y_{21} & y_{21} = x_{21} \\ x_{23} = x_{22} + y_{22} & y_{22} = x_{22} \\ x_{31} = x_{30} + y_{30} & y_{30} = x_{30} \\ x_{41} = x_{40} + y_{40} & y_{40} = x_{40} \\ x_{42} = x_{41} + y_{41} & y_{41} = x_{41} \\ x_{11} + (x_{20} + x_{21} + x_{23}) - (x_{30} + x_{31}) + x_{40} = 0 & \\ (x_{10} + x_{11}) + (x_{21} + x_{22}) - (x_{40} + x_{42}) = 0. & \end{array}$$

3.3 Scaling

A polyhedron (9) with $\mathbf{A} \in \{-1, 0, 1\}^{m \times n}$ is now scaled down such that all its vertices are contained in the box $[0, \frac{1}{n}]^n$. This ensures that all quantities represented by pseudomarginals fit into the interval $[0, 1]$ (see Sections 4.2 and 5.1).

Lemma 1. *Each vertex* $\mathbf{x}$ *of convex polyhedron 9 with* $\mathbf{A} \in \{-1, 0, 1\}^{m \times n}$ *satisfies* $\mathbf{x} \in [0, M]^n$ *where*

$$M = \prod_{j=1}^{n} \sum_{i=1}^{m} |a_{ij}|. \tag{15}$$

Moreover, $\text{size}(M) = O(\text{size}(\mathbf{A}))$.

Proof. See Lemma 4 and Section 4.3 in [9]. □

By Lemma 1, the polyhedron must be scaled down by the factor nM. This can be conveniently done during the transformation in Section 3.2. Let $\mathbf{A} \in \{-1, 0, 1\}^{m \times n}$ be the output matrix and j the index of the last variable of the input system in Section 3.2. Without scaling, we would set $x_j = 1$. To achieve scaling, set $d_j = \lceil \log_2(nM) \rceil$ and $x_{j,d_j} = 1$. By (14), this yields $x_j = 2^{-d_j} \leq (nM)^{-1}$.

Though the number nM can be big, by Lemma 1 its size, and hence the number of added equations, is $O(\text{size}(\mathbf{A}))$.

To summarize Section 3, polyhedron (9) with rational coefficients has been transformed in linear time to a polyhedron of the same form with coefficients $\{-1, 0, 1\}$ and vertices in $[0, \frac{1}{n}]^n$. More precisely, the input polyhedron is a scaled coordinate-erasing projection of the output polyhedron, where the erased coordinates correspond to the auxiliary variables introduced in Sections 3.1 and 3.2. Here, we call a projection *coordinate-erasing* if it acts by erasing a subset of coordinates.

4 Encoding by Potts Problem

Here we will represent the polyhedron obtained in Section 3 by the LP relaxation of a Potts problem. In fact, the output problem will be a *reparameterized Potts problem*, i.e., a min-sum problem with arbitrary unary costs $g_u(k)$ and pairwise costs (4b) with $g_{uv}(k, \ell)$ given by (5). By moving $\varphi_{uv}(k)$ to the unary costs, such a problem can be reparameterized in linear time to a Potts problem with unary costs (4a) and pairwise costs (5).

4.1 Gadgets

We will construct the output problem by gluing small subproblems, called *gadgets*,[3] which encode simple operations on unary pseudomarginals. Each gadget is a reparameterized Potts problem with unary costs $g_u(k) \in \{0, 1\}$ and pairwise costs

$$g_{uv}(k, \ell) = 2[\![k \neq \ell]\!] + \varphi_{uv}(k) + \varphi_{vu}(\ell), \tag{16}$$

(i.e., we set[4] $h_{uv} = 2$ for all $\{u, v\} \in E$ in (5)) where $\varphi_{uv}(k) \in \{-1, 0, 1\}$. In addition, the costs satisfy

$$\min_{k \in K} g_u(k) = 0, \quad u \in V, \tag{17a}$$

$$\min_{k,\ell \in K} g_{uv}(k, \ell) = 0, \quad \{u, v\} \in E. \tag{17b}$$

Each gadget is designed such that its LP relaxation has zero optimal value. It follows that any $\boldsymbol{\mu} \in \Lambda$ is optimal to (2) if and only if

$$g_u(k)\mu_u(k) = 0, \quad u \in V,\ k \in K, \tag{18a}$$

$$g_{uv}(k, \ell)\mu_{uv}(k, \ell) = 0, \quad \{u, v\} \in E,\ k, \ell \in K, \tag{18b}$$

i.e., whenever a cost is positive then the corresponding pseudomarginal must vanish.

We will define gadgets by diagrams such as in Fig. 1, adopting the following conventions. Each non-zero number $\varphi_{uv}(k)$ is written near node (u, k) on the side of object v, where '+' stands for $\varphi_{uv}(k) = 1$ and '−' for $\varphi_{uv}(k) = -1$. A node (u, k) is black if $g_u(k) = 0$ and white if $g_u(k) = 1$. An edge $\{(u, k), (v, \ell)\}$ is drawn only if $g_{uv}(k, \ell) = 0$ and both of its end-nodes are black, otherwise it is invisible. Fig. 2 shows an example.

We will use the following gadgets, defined in Fig. 3:

- Swap swaps two unary pseudomarginals, one of them zero. Precisely, the LP relaxation of this gadget has zero optimal value if and only if the unary pseudomarginals linked by visible edges are equal and the unary pseudomarginals in the white nodes are zero.
- Permute applies Swap several times to arbitrarily permute all the three unary pseudomarginals, one of them zero. The figure shows one possible permutation.
- Copy copies all the three unary pseudomarginals, one of them zero, from one object to another object.

3. When constructing reductions in complexity theory, a *gadget* is a small instance of the output problem that implements a certain simple functionality of the input problem. In [9] we used the term 'elementary construction' instead of 'gadget'.

4. We could have just as well set $h_{uv} = 1$; we chose $h_{uv} = 2$ only for convenience because then all $\varphi_{uv}(k)$ can be integer.

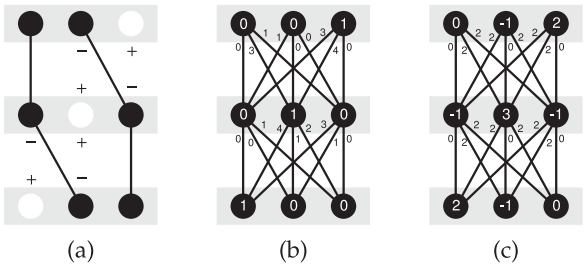


Fig. 2. Our notation for gadgets. (a) shows a gadget in our notation. (b) is the corresponding reparameterized Potts problem with unary costs written inside nodes and pairwise costs written next to edges; each $+$ [$-$] contributes by 1 [-1] to the pairwise cost of the adjacent edge. (c) is the corresponding Potts problem; each $+$ [$-$] contributes by 1 [-1] to the unary cost of the adjacent node, each white node contributes to its unary cost by additional increment 1.

- UNIT enforces a unary pseudomarginal to be 1.
- ADD1 adds two unary pseudomarginals in a single object and copies the result in another object. The third unary pseudomarginal is copied.
- ADD adds two unary pseudomarginals in two different objects. This is done by gluing three ADD1gadgets.

Each gadget has several versions obtained by permuting the three labels. Each interface object of COPY, SWAP and ADD has two black nodes and one white node. This ensures that any versions of COPY and ADD can be glued together, possibly after permuting the nodes by PERMUTE. UNIT can be glued with any gadget with black node linked to the black node labeled 1. When several gadets are glued, the unary costs in identified nodes (u, k) of their interface objects are summed.

4.2 Encoding

We now describe the encoding algorithm. The input of the algorithm is a polyhedron (9) with $\mathbf{A} \in \{-1, 0, 1\}^{m\times n}$ and the vertices in $[0, \frac{1}{n}]^n$. Its output is a reparameterized Potts problem with $|K| = 3$ labels.

First, we rewrite the system $\mathbf{Ax} = \mathbf{0}$ in (9) as

$$\mathbf{A}^+\mathbf{x} = \mathbf{A}^-\mathbf{x}, \tag{19}$$

where $a^+_{ij} = \max\{a_{ij}, 0\}$ and $a^-_{ij} = \max\{-a_{ij}, 0\}$ so that $a^+_{ij}, a^-_{ij} \in \{0, 1\}$. That is, we have moved negative terms in each equation to the other side of the equation.

Let the three labels of the output problem be named $K = \{1, 2, 3\}$. The encoding proceeds as follows:

1) Set $V = \{1, \dots, n\}$ and $E = \emptyset$. Each variable x_j is now represented by unary pseudomarginal $\mu_j(1)$.
2) For each $i = 1, \dots, m$, encode the ith equation of system (19) as follows:
   a) Construct a unary pseudomarginal equal to the LHS of the equation using ADD, permuting labels by PERMUTE if necessary.
   b) Do the same for the RHS.
   c) Equate the LHS and RHS using COPY, permuting labels by PERMUTE if necessary.
3) Encode the equation $x_n = 1$ using UNIT.

Assume that the input polyhedron P is bounded (i.e., a polytope). Due to the scaling done in Section 3.3, $\mathbf{A}^+\mathbf{x} = \mathbf{A}^-\mathbf{x} \le \mathbf{1}$ for all $\mathbf{x} \in P$. Therefore every expression formed in Steps 2a and 2b fits into the feasible interval $[0, 1]$ of pseudomarginals. Recall that the LP relaxation of each gadget has zero optimal value. Since the gadgets are glued such that they encode the input system $\mathbf{Ax} = \mathbf{0}$, the LP relaxation of the output problem will have zero optimal value if and only if P is non-empty. In other words, the output min-sum problem encodes the input polytope as follows:

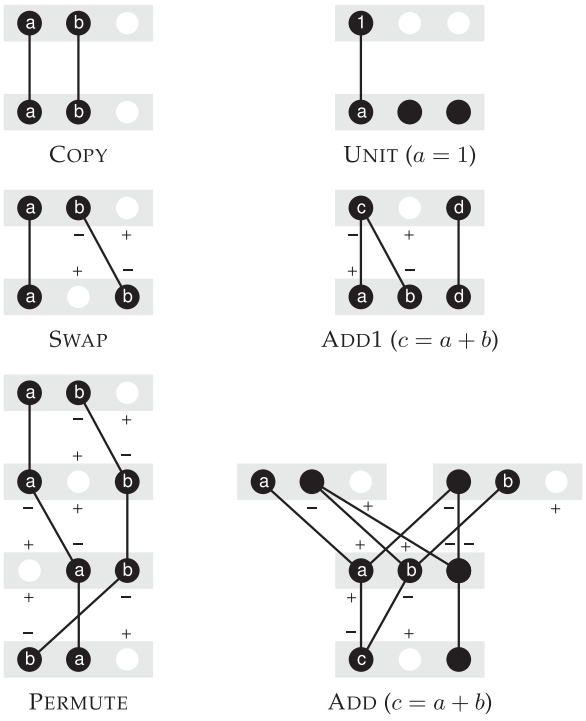


Fig. 3. Potts problems used as gadgets.

- If $P = \emptyset$ then $\min_{\boldsymbol{\mu}\in\Lambda}\langle \mathbf{g}, \boldsymbol{\mu}\rangle > 0$.
- If $P \neq \emptyset$ then $\min_{\boldsymbol{\mu}\in\Lambda}\langle \mathbf{g}, \boldsymbol{\mu}\rangle = 0$ and

$$P = \pi\Big(\operatorname*{arg\,min}_{\boldsymbol{\mu}\in\Lambda}\langle \mathbf{g}, \boldsymbol{\mu}\rangle\Big), \tag{20}$$

where 'argmin' denotes the set of all minimizers and

$$\pi\colon \mathbb{R}^I \to \mathbb{R}^n, \quad \pi(\boldsymbol{\mu}) = (\mu_1(1), \dots, \mu_n(1)), \tag{21}$$

is the coordinate-erasing projection that erases all pseudomarginals not representing the input variables (see Step 1 of the algorithm).

Fig. 4 shows the constructed reparameterized Potts problem for an example input polyhedron.

As for each $a_{ij} \neq 0$ a constant number of objects and object pairs is created, the encoding time is $O(\text{size}(\mathbf{A}))$.

5 OBTAINED REDUCTIONS

In Sections 3 and 4 we described our core construction. Here we describe several reductions that are more or less straightforward consequences of this construction.

5.1 Reduction from LF and LP

The *linear feasibility problem* is the problem of solving a system of linear inequalities. In our formulation, given a matrix $\mathbf{A}$ (with rational entries) the aim is to decide if the polyhedron P is nonempty and if so, to find an element $\mathbf{x} \in P$.

Theorem 2. *The linear feasibility problem reduces in linear time to the LP relaxation of the Potts problem with three labels.*

Proof. If P is bounded, the claim holds by composing the reductions in Sections 3 and 4. If P is unbounded, it has at least one vertex. The reduction in Section 4 cuts off a part of P because the pseudomarginals are bounded by 1. But, due to the scaling in Section 3.3, this part does not contain any vertex. Therefore, cutting this part off preserves at least some solutions to the input problem. □

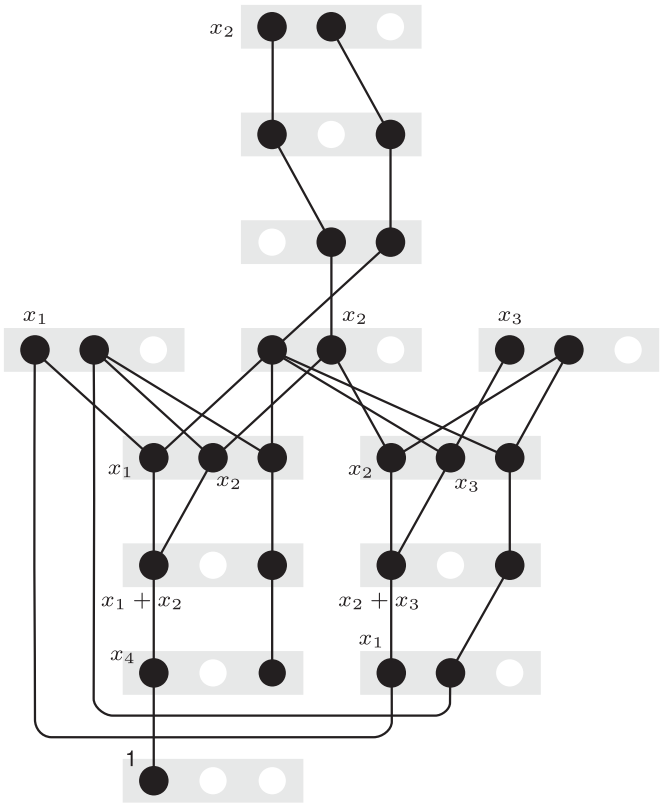


Fig. 4. A reparameterized Potts problem that encodes the polyhedron $P = \{\, (x_1, x_2, x_3, x_4) \in \mathbb{R}^4 \,|\, x_1 + x_2 = x_4,\ x_2 + x_3 = x_1,\ x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0,\ x_4 = 1 \,\}$.

The *linear programming problem* is the problem of minimizing a linear function subject to linear inequalities.

Lemma 3. *The linear programming problem reduces in linear time to the linear feasibility problem.*

Proof. By strong duality, any linear program

$$\min\{\, \langle \mathbf{c}, \mathbf{x} \rangle \,|\, \mathbf{A}\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq \mathbf{0} \,\},$$

can be solved by solving the system

$$\mathbf{A}\mathbf{x} = \mathbf{b},\quad \mathbf{A}^T\mathbf{y} + \mathbf{z} = \mathbf{c},\quad \langle \mathbf{c}, \mathbf{x} \rangle = \langle \mathbf{b}, \mathbf{y} \rangle,\quad \mathbf{x}, \mathbf{z} \geq \mathbf{0}.$$

Plugging $\mathbf{y} = \mathbf{y}_+ - \mathbf{y}_-$ where $\mathbf{y}_+, \mathbf{y}_- \geq \mathbf{0}$ puts this system into form 9. The system is feasible if and only if the input LP is feasible and bounded. The reduction takes linear time because it essentially copies $\mathbf{A}, \mathbf{b}, \mathbf{c}$ twice to the output. □

This gives us the central result of our paper.

Theorem 4. *The linear programming problem reduces in linear time to the LP relaxation of the Potts problem with three labels.*

Proof. Combine Lemma 3 and Theorem 2. □

5.2 Polyhedral Interpretation

Composing the reductions done in Sections 3 and 4 (see equality (20)) yields that the input polytope P is a (scaled) coordinate-erasing projection of a face of a local marginal polytope Λ. This recovers our result [9, Theorem 1] with the constraint that the output problem is a (reparameterized) Potts problem. Here we reformulate this result in terms of the relaxed Potts polytope Π.

By moving the numbers $\varphi_{uv}(k)$ to the unary costs, (20) can be expressed in terms of Π rather than Λ. Defining vector $\mathbf{h} \in \mathbb{R}^{(V \times K) \cup E}$ by $h_u(k) = g_u(k) - \sum_{v \in N_u} \varphi_{uv}(k)$ and $h_{uv} = 2$, we indeed have

$$\pi\Big(\operatorname*{arg\,min}_{\boldsymbol{\mu} \in \Lambda} \langle \mathbf{g}, \boldsymbol{\mu} \rangle\Big) = \pi'\Big(\operatorname*{arg\,min}_{\nu \in \Pi} \langle \mathbf{h}, \nu \rangle\Big), \tag{22}$$

where $\pi' \colon \mathbb{R}^{(V \times K) \cup E} \to \mathbb{R}^n$ is the coordinate-erasing projection given by $\pi'(\nu) = (\nu_1(1), \ldots, \nu_n(1))$. Comparing (20) with (22) yields the following result.

Theorem 5. *Every polytope is (up to scale) a coordinate-erasing projection of a face of a relaxed Potts polytope with three labels, whose description (by a set of linear inequalities) can be computed from the description of the input polytope in linear time.*

5.3 Relation to the Dual LP Relaxation

Rather than linear program (2) it is often better to solve its dual. As shown, e.g., in [6], the dual LP relaxation maximizes the function

$$L(\mathbf{g}) = \sum_{u \in V} \min_{k \in K} g_u(k) + \sum_{\{u,v\} \in E} \min_{k,\ell \in K} g_{uv}(k, \ell), \tag{23}$$

over reparameterization of $\mathbf{g}$ (i.e., we maximize $L(\mathbf{g}')$ over φ, where $\mathbf{g}'$ is given by (4). Function (23) is a lower bound on (2),

$$\min_{\boldsymbol{\mu} \in \Lambda} \langle \mathbf{g}, \boldsymbol{\mu} \rangle \geq L(\mathbf{g}). \tag{24}$$

By strong duality, inequality (24) holds with equality if and only if $\mathbf{g}$ is dual-optimal, i.e., no reparameterization of $\mathbf{g}$ can increase the lower bound.

For the Potts problem, the dual optimal value does not change if we add to the dual the constraints

$$\varphi_{uv}(k) + \varphi_{vu}(k) = 0, \qquad \{u, v\} \in E,\ k \in K \tag{25a}$$

$$|\varphi_{uv}(k)| \leq \frac{1}{2} h_{uv}, \quad u \in V,\ v \in N_u,\ k \in K. \tag{25b}$$

This is proved by writing the dual of the Kleinberg-Tardos relaxation (7), see Theorem 10 in Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPAMI.2016.2582165. Note that the numbers $\varphi_{uv}(k)$ used in our gadgets satisfy (25).

Let us emphasize that the reduction from Section 4 applies only to the primal LP relaxation. The question whether there is a linear-time reduction of the general LP to the dual LP relaxation is left open in this paper. Does our reduction relate in any way to the dual LP relaxation? The reparameterized Potts problem constructed in Section 4.2 satisfies (17), hence it has $L(\mathbf{g}) = 0$. Therefore:

- If $P = \emptyset$ then $\min_{\boldsymbol{\mu} \in \Lambda} \langle \mathbf{g}, \boldsymbol{\mu} \rangle > L(\mathbf{g}) = 0$.
- If $P \neq \emptyset$ then $\min_{\boldsymbol{\mu} \in \Lambda} \langle \mathbf{g}, \boldsymbol{\mu} \rangle = L(\mathbf{g}) = 0$.

This shows that the linear feasibility problem in fact reduces to a simpler problem than the LP relaxation (2), namely, to deciding whether $\mathbf{g}$ is dual optimal.

Theorem 6. *The linear feasibility problem reduces in linear time to the following problem: given a reparameterized Potts problem with three labels, decide if its cost vector is optimal to the dual LP relaxation.*

5.4 Reduction to Multiway Cut Problem

Closely related to the Potts problem is the *multiway cut problem*. For its LP relaxation, given in [17], we prove a result analogous to Theorem 4.

A multiway cut in a graph $(V \cup K, E')$, where K are terminals and $E' \subseteq \binom{V \cup K}{2}$, is a subset of edges whose removal leaves each terminal in a separate component. Given edge costs $h'_{uv} \geq 0$, the goal of the multiway cut problem is to find a multicut with minimum total cost. The LP relaxation of this problem [17] reads

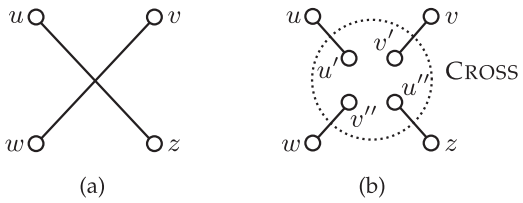


Fig. 5. Eliminating an edge crossing.

$$\text{minimize} \sum_{\{u,v\}\in E'} \frac{1}{2} h'_{uv} \sum_{k\in K} |\mu_u(k) - \mu_v(k)| \tag{26a}$$

$$\text{subject to} \sum_{k\in K} \mu_u(k) = 1, \quad u \in V \cup K \tag{26b}$$

$$\mu_k(k) = 1, \qquad k \in K \tag{26c}$$

$$\mu_u(k) \geq 0, \qquad u \in V,\ k \in K. \tag{26d}$$

Theorem 7. *The LP relaxation of the Potts problem reduces in linear time to the LP relaxation of the multiway cut problem.*

Proof. As shown, e.g., in [12, Section 7.1], the Potts problem with graph (V, E), unary costs $g_u(k)$, and Potts costs h_{uv} reduces to the multiway cut problem with graph $(V \cup K, E')$ where $E' = E \cup \{\, \{u, k\} \,|\, u \in V,\ k \in K \,\}$, and costs $h'_{uv} = h_{uv}$ for $\{u, v\} \in E$ and $h'_{uk} = c_u - g_u(k)$ for $u \in V$, $k \in K$, where $c_u = \max_k g_u(k)$. Since $|K|$ is constant in our case, the reduction takes linear time.

We show that this reduction preserves the LP relaxation. The objective (26a) reads

$$\begin{aligned} &\sum_{\{u,v\}\in E} \frac{1}{2} h_{uv} \sum_{k\in K} |\mu_u(k) - \mu_v(k)| \\ &+ \sum_{u\in V} \sum_{k\in K} \frac{1}{2} [c_u - g_u(k)] \sum_{\ell\in K} |\mu_u(\ell) - \mu_k(\ell)|. \end{aligned} \tag{27}$$

Using (26c) we have

$$\sum_{\ell\in K} |\mu_u(\ell) - \mu_k(\ell)| = [1 - \mu_u(k)] + \sum_{\ell\neq k} \mu_u(\ell) = 2[1 - \mu_u(k)],$$

so the second sum in 27 is

$$\sum_{u\in V} \sum_{k\in K} [c_u - g_u(k)][1 - \mu_u(k)] = C + \sum_{u\in V} \sum_{k\in K} g_u(k)\mu_u(k).$$

Therefore (27) equals (6) up to a constant C. □

Theorem 8. *The linear programming problem reduces in linear time to the LP relaxation of the multiway cut problem with three terminals.*

Proof. Combine Theorems 4 and 7. □

5.5 Reduction to Planar Potts Problem

As in [9], reduction to the Potts problem is possible even if this problem is required to have planar structure, at the expense of increasing the reduction complexity.

Theorem 9. *The linear programming problem reduces in quadratic time to the LP relaxation of the planar Potts problem with three labels, whose size is quadratic.*

Proof. Consider the reparameterized Potts problem constructed in Section 4.2, with graph (V, E). We will replace this problem with a planar reparameterized Potts problem with the same LP relaxation.

Let the graph (V, E) be drawn in the plane, such that the vertices are distinct points and the edges are line segments connecting the vertices. We assume w.l.o.g. that no three edges intersect at a common point, except at graph vertices. We will replace every edge crossing with a planar reparameterized Potts problem.

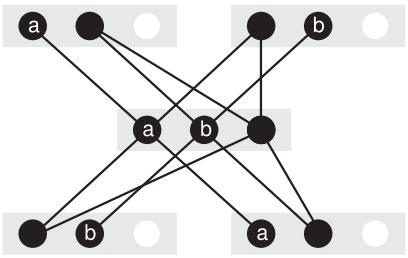


Fig. 6. Gadget CROSS.

Let $\{u, z\}, \{v, w\} \in E$ be a pair of crossing edges, as shown in Fig. 5a. This pair of edges is replaced by a construction outlined in Fig. 5b. Object u is linked to object u' and v' is linked to v' using COPY. Object z is linked to u'' and w is linked to v'', setting $g_{u''z} = g_{uz}$ and $g_{v''w} = g_{vw}$. The encircled objects are linked to a gadget, named CROSS, that enforces unary pseudomarginals in objects u', u'' and v', v'' to be equal. If necessary, labels are again permuted using PERMUTE. The construction can be drawn arbitrarily small so that it is not intersected by any other edges.

The CROSS gadget is shown in Fig. 6. It is composed of four ADD1 gadgets. It works correctly only if $a + b \leq 1$. To ensure this, all pseudomarginals representing input variables in the output problem are scaled down by the factor of 2. This can be done by replacing the equation $x_n = 1$ in (9) with $x_n = \frac{1}{2}$, where the constant $\frac{1}{2}$ is constructed similarly as in Section 3.2, which can be done using a reparameterized Potts problem with planar structure.

Since the total number of edge crossings in a graph (V, E) is $O(|E|^2)$, the reduction time and the output size are quadratic. □

The encoding time and the size of the output can be improved using [9, Lemma 10].

6 CONCLUDING REMARKS

We have constructed a linear-time reduction from the general linear program to the LP relaxation of the Potts problem with three labels. This shows that there is little hope to find a very efficient algorithm (based, e.g., on simple combinatorial principles) to solve the LP relaxation of the Potts problem. This negative result applies also to labeling problems with metric and semimetric pairwise cost functions (of which Potts is a special case), which often arise in computer vision [1], [2].

Let us compare this result with our previous work [9] where we constructed a linear-time reduction from LP to the LP relaxation of the min-sum problem with costs in $\mathbb{Z} \cup \{\infty\}$ [9, Theorem 2] and a quadratic-time reduction from LP to the LP relaxation of the min-sum problem with costs in $\mathbb{Z}$ [9, Theorem 9].

Theorem 4 is stronger than [9, Theorem 9] because the output min-sum problem (being the Potts problem) has costs in $\mathbb{Z}$ and our reduction is in linear time.

Theorem 4 is stronger than [9, Theorem 2] because there costs $\mathbb{Z} \cup \{\infty\}$ are needed for linear-time reduction. However, there is a price for this. The reduction [9, Theorem 2] has the desirable property of preserving approximation ratio: if a sub-optimal (i.e., feasible) solution of the LP relaxation of the output min-sum problem is found, the ratio of the optimal and suboptimal objective value is the same as for the input LP. We did not mention this property in [9] but it is rather obvious. Reductions with finite output costs do not have this property. It is open whether there exists a linear-time reduction from the general LP to the LP relaxation of the

min-sum problem with finite costs (or even the Potts problem) that preserves approximation ratio.

Another difference from [9] is that there we encoded the input polyhedron directly by a min-sum problem while here we first preprocess it to the form with coefficients $\{-1, 0, 1\}$. In fact, this preprocessing could be used to simplify the reduction in [9].

On the other hand, our results in this paper could be proved in an alternative way, shorter but less transparent. In [9, Section 4.2] we constructed a linear-time reduction from LF to the LP relaxation (2) of the min-sum problem with costs in $\{0, \infty\}$. This LP relaxation has the form 9 with coefficients $\{-1, 0, 1\}$ and every $\mathbf{x} \in P$ satisfying $\mathbf{A}^+\mathbf{x} = \mathbf{A}^-\mathbf{x} \leq \mathbf{1}$, so it can be encoded by a Potts problem as described in Section 4.

Finally, our work is related to [18], [19] where polyhedral universality results similar to our Theorem 5 are derived for the three-way transportation polytope and the traveling salesman polytope. However, the reduction time in these works is not shown to be linear.

Acknowledgments

This work has been supported by the Czech ministry of education, youth and sports under the ERC-CZ grant LL1303.

References

[1] R. Szeliski, et al., "A comparative study of energy minimization methods for Markov random fields with smoothness-based priors," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 6, pp. 1068–1080, Jun. 2008.

[2] J. H. Kappes, et al., "A comparative study of modern inference techniques for structured discrete energy minimization problems," *Int. J. Comput. Vis.*, vol. 115, no. 2, pp. 155–184, 2015.

[3] S. Živný, *The Complexity of Valued Constraint Satisfaction Problems*. Berlin, Germany: Springer, 2012.

[4] M. J. Wainwright and M. I. Jordan, "Graphical models, exponential families, and variational inference," *Found. Trends Mach. Learn.*, vol. 1, no. 1–2, pp. 1–305, 2008.

[5] M. I. Shlezinger, "Syntactic analysis of two-dimensional visual signals in noisy conditions," *Cybern. Syst. Anal.*, vol. 12, no. 4, pp. 612–628, 1976.

[6] T. Werner, "A linear programming approach to max-sum problem: A review," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 7, pp. 1165–1179, Jul. 2007.

[7] A. Koster, S. P. van Hoesel, and A. W. Kolen, "The partial constraint satisfaction problem: Facets and lifting theorems," *Oper. Res. Lett.*, vol. 23, no. 3–5, pp. 89–97, 1998.

[8] V. Kolmogorov, J. Thapper, and S. Živný, "The power of linear programming for general-valued CSPs," *SIAM J. Comput.*, vol. 44, no. 1, pp. 1–36, 2015.

[9] D. Průša and T. Werner, "Universality of the local marginal polytope," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 4, pp. 898–904, Apr. 2015.

[10] E. Boros and P. L. Hammer, "Pseudo-Boolean optimization," *Discrete Appl. Math.*, vol. 123, no. 1–3, pp. 155–225, 2002.

[11] C. Rother, V. Kolmogorov, V. S. Lempitsky, and M. Szummer, "Optimizing binary MRFs via extended roof duality," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2007, pp. 1–8.

[12] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, no. 11, pp. 1222–1239, Nov. 2001.

[13] J. Kleinberg and E. Tardos, "Approximation algorithms for classification problems with pairwise relationships: Metric labeling and Markov random fields," *J. ACM*, vol. 49, no. 5, pp. 616–639, 2002.

[14] C. Chekuri, S. Khanna, J. Naor, and L. Zosin, "A linear programming formulation and approximation algorithms for the metric labeling problem," *SIAM J. Discrete Math.*, vol. 18, no. 3, pp. 608–625, 2005.

[15] J. Chuzhoy and J. Naor, "The hardness of metric labeling," *SIAM J. Comput.*, vol. 36, no. 5, pp. 1376–1386, 2007.

[16] P. Kumar, "Rounding-based moves for metric labeling," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 109–117.

[17] G. Călinescu, H. Karloff, and Y. Rabani, "An improved approximation algorithm for multiway cut," in *Proc. 30th Annu. ACM Symp. Theory Comput.*, 1998, pp. 48–52. [Online]. Available: http://doi.acm.org/10.1145/276698.276711

[18] J. A. De Loera and S. Onn, "All linear and integer programs are slim 3-way transportation programs," *SIAM J. Optim.*, vol. 17, no. 3, pp. 806–821, 2006.

[19] L. J. Billera and A. Sarangarajan, "All 0-1 polytopes are traveling salesman polytopes," *Combinatorica*, vol. 16, no. 2, pp. 175–188, 1996.

Graph-based Simplex Method for Pairwise Energy Minimization with Binary Variables

Daniel Průša
Center for Machine Perception, Faculty of Electrical Engineering, Czech Technical University
Karlovo náměstí 13, 121 35 Prague, Czech Republic
prusapa1@fel.cvut.cz

Abstract

We show how the simplex algorithm can be tailored to the linear programming relaxation of pairwise energy minimization with binary variables. A special structure formed by basic and nonbasic variables in each stage of the algorithm is identified and utilized to perform the whole iterative process combinatorially over the input energy minimization graph rather than algebraically over the simplex tableau. This leads to a new efficient solver. We demonstrate that for some computer vision instances it performs even better than methods reducing binary energy minimization to finding maximum flow in a network.

1. Introduction

Energy minimization is a well known NP-hard combinatorial problem which arises in MAP inference in graphical models [16]. It is of great importance in low-level computer vision. A lot of effort has been spent by researchers to invent methods finding exact or approximate solutions.

Pairwise energy minimization with binary variables has a prominent role. It is easily expressible as quadratic pseudo-boolean optimization (QPBO) [1, 8]. Max-flow/min-cut algorithms can be applied to find a partial optimal solution, with some variables undecided. This shrinks the size of the input problem. The undecided variables can be further resolved by applying *e.g.* the *probing* technique [13] or, in general, by branch and bound. A complete optimal solution is always returned by QPBO for *submodular* instances. This is further utilized in some cases of energy minimization with general variables. Every multi-label energy minimization reduces to the binary one [6, 14], preserving the submodularity property. Further, solving submodular binary instances is a crucial part of approximate minimizations [3].

Since binary energy minimization is cast as a max-flow/min-cut problem, the key prerequisite for solving it efficiently has been so far to come up with a max-flow algorithm working well on vision instances. The popular algorithm by Boykov and Kolmogorov [2] fulfills this role. An empirical comparison of max-flow algorithms recently done by Verma and Batra [15] reveals that some other contemporary implementations are more suitable for instances with dense graphs.

In this paper we introduce a different principle to solve the binary case. We do not perform any translation to max-flow. Our starting point is the linear programming (LP) relaxation of the problem [17]. For binary variables, it is known to be half-integral, *i.e.*, all components of each optimal solution are in $\{0, \frac{1}{2}, 1\}$. Moreover, the solution coincides with the result of QPBO – value $\frac{1}{2}$ indicates undecided variables [1]. We show that the simplex method solving the LP relaxation can be turned into a very efficient algorithm, performed purely over the input energy minimization graph. Special versions of the simplex method with similar properties have already been proposed for *transportation*, *assignment* and *minimum cost-flow* problems [4]. They are known as the *network simplex* algorithms. There is even a customization for the maximum flow problem [5], though it does not figure among the leading implementations.

The proposed algorithm has practical benefit. Our experiments demonstrate there are vision instances, where the algorithm performs better than max-flow based solvers. Besides that, it allows to study the behavior of the simplex method on large-scale data and gives a hope for a generalization to multi-label problems. And finally, its impact may extend beyond the scope of energy minimization as it can be applied to min-cut which is expressible as submodular binary energy minimization. Since the formulation and the underlying structure is different than in the case of the network simplex algorithm for max-flow, new ideas for solving min-cut/max-flow may emerge.

We assume the reader is familiar with the simplex method, as presented *e.g.* in [4]. Knowledge of notions like *basis*, *basic variable*, *basic feasible solution*, *pivoting rule* or *minimum ratio test* is essential for understanding the text.

978-1-4673-6964-0/15/$31.00 ©2015 IEEE

2. Energy minimization and its LP relaxation

The task of pairwise energy minimization is to compute

$$\min_{\mathbf{k}\in K^V} \left[\sum_{u\in V} \theta_u(k_u) + \sum_{\{u,v\}\in E} \theta_{uv}(k_u, k_v) \right] \tag{1}$$

where V is a finite set of *objects*, $E \subset \binom{V}{2}$ is a set of *object pairs* (*i.e.*, (V, E) is an undirected graph), K is a finite set of *labels*, and the functions $\theta_u\colon K \to \mathbb{R}$ and $\theta_{uv}\colon K \times K \to \mathbb{R}$ are unary and pairwise *interactions*. We adopt that $\theta_{uv}(k,\ell) = \theta_{vu}(\ell, k)$ and refer to the values of θ_u and θ_{uv} as *potentials*. We shortly write $\theta_u(k)$ as $\theta_{u;k}$ and $\theta_{uv}(k,\ell)$ as $\theta_{uv;k\ell}$. The potentials together form a vector $\boldsymbol{\theta} \in \mathbb{R}^I$ with

$$\begin{aligned} I = &\{\, (u;k) \mid u \in V,\ k \in K \,\} \cup \\ &\{\, (uv;k\ell) \mid \{u,v\} \in E;\ k,\ell \in K \,\}. \end{aligned} \tag{2}$$

Throughout the paper, we consider energy minimization with two labels, so K is fixed as $\{0,1\}$. An instance of problem (1) is thus fully defined by a tuple $(V, E, \boldsymbol{\theta})$. Its linear programming relaxation reads as

$$\operatorname*{argmin}_{\mathbf{x}\in\Lambda} \langle \boldsymbol{\theta}, \mathbf{x} \rangle. \tag{3}$$

Here we optimize over polytope Λ which consists of vectors $\mathbf{x} \in \mathbb{R}^I$ that satisfy the constraints

$$\sum_{\ell\in K} x_{uv;k\ell} = x_{u;k}, \quad u \in V,\ v \in N_u,\ k \in K \tag{4a}$$

$$\sum_{k\in K} x_{u;k} = 1, \qquad u \in V \tag{4b}$$

$$\mathbf{x} \geq \mathbf{0}, \tag{4c}$$

where $N_u = \{\, v \mid \{u,v\} \in E \,\}$ is the set of neighbors of object u. We again adopt $x_{uv;k\ell} = x_{vu;\ell k}$.

3. Applying simplex method – preparation

We work with the variant of the simplex method which assumes the input linear program to be in the *standard form*:

$$\min\{\langle \boldsymbol{\theta}, \mathbf{x} \rangle \mid \mathbf{A}\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq \mathbf{0}\} \tag{5}$$

where $\mathbf{A} \in \mathbb{R}^{m\times n}$, $m \leq n$, $\operatorname{rank}(\mathbf{A}) = m$, $\mathbf{b} \in \mathbb{R}^m$, and $\boldsymbol{\theta} \in \mathbb{R}^n$.

The LP relaxation of an energy minimization given by $(V, E, \boldsymbol{\theta})$ induces overall $n = 2|V| + 4|E|$ variables and $m = |V| + 3|E|$ linearly independent equations [9].

For a given *basis* $\mathcal{B} \subset I$, Figure 1 shows how we visualize *basic* and *nonbasic* LP relaxation variables. The scheme includes the input graph (V, E). The variables form an underlying "microstructure". Nonbasic variables are colored in red. They always attain zero value in the induced feasible solution. Basic variables attaining nonzero or zero value are black or blue, respectively. Each such scheme is called the *LP relaxation diagram*. In accordance with the diagram appearance, every unary or pairwise LP relaxation variable is simply called a *node* (variable) or *edge* (variable), respectively.

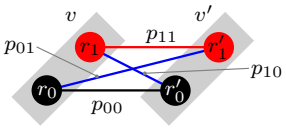


Figure 1. An LP relaxation diagram. The energy minimization graph consists of objects v, v' and object pair $\{v, v'\}$. LP relaxation variables are denoted as $r_k = x_{v;k}$, $r'_k = x_{v';k}$, and $p_{k\ell} = x_{vv';k\ell}$. Nonbasic variables are red, basic variables are black or blue. The selected basis is $\mathcal{B} = \{(v;0), (vv';00), (v';0), (vv';01), (vv';10)\}$.

The set of basic variables represented in Figure 1 results in the following *simplex tableau*.

	r_0	r_1	r'_0	r'_1	p_{00}	p_{01}	p_{10}	p_{11}	
	θ_0	θ_1	θ'_0	θ'_1	θ_{00}	θ_{01}	θ_{10}	θ_{11}	0
r_0	1	1	0	0	0	0	0	0	1
r'_0	0	0	1	1	0	0	0	0	1
p_{00}	0	1	0	1	1	0	0	-1	1
p_{01}	0	0	0	-1	0	1	0	1	0
p_{10}	0	-1	0	0	0	0	1	1	0

Constraints (4a), (4b) give 6 linear equations, but only 5 of them are linearly independent. Each row of the tableau is a linear combination of the constraints expressing a basic variable. The induced basic solution is *feasible* since all elements in the rightmost column are nonnegative. The simplex algorithm could be launched if the row with potentials (*objective costs*) is adjusted to contain zeros in all basic columns.

Performance of the standard, tableau-based simplex algorithm is influenced by two factors.

- The number of performed iterations. It tends to be steadily $\mathcal{O}(n)$ in practical applications [4]. Some pathological LP instances result in an exponential amount of iterations. On the other hand, polynomial upper bounds were proved *e.g.* for the assignment problem mentioned in the introduction.

- Memory and running time required to represent and update the simplex tableau. This is what makes the method computationally infeasible for large-scale instances, quadratic time and space is required. A usage of a sparse matrix might be helpful, however, the number of nonzero elements keeps growing and the

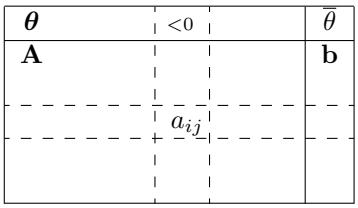


Figure 2. Having chosen pivot a_{ij}, only nonzero elements in the i-th row and j-th column are needed to update $\boldsymbol{\theta}$, $\overline{\theta}$ and $\mathbf{b}$ properly.

update is still a time-consuming operation. The problems are partially addressed by the so called *revised simplex* method [4], but still, the general algorithm is not fast enough to be able to compete with max-flow based QPBO solvers.

The key concept leading to an efficient algorithm is explained in Figure 2. Each iteration of the simplex algorithm updates cost vector $\boldsymbol{\theta}$ and the objective function value $\overline{\theta}$ based on the pivotal row, while the right-hand sides vector $\mathbf{b}$ is updated based on the pivotal column. Moreover, only nonzero values have an impact. We show that the LP relaxation diagram enables a cheap retrieval of all such nonzero elements. It is thus enough to maintain only $\boldsymbol{\theta}$, $\overline{\theta}$ and $\mathbf{b}$.

4. Structure of basis

It is a well known fact that each basic variable is expressible as a linear function of nonbasic variables. Moreover, it is expressed in a unique way.

Theorem 1. *Let $\mathcal{B} \subset I$ be a basis of (3) and let $\mathcal{N} = I \setminus \mathcal{B}$ be the set of nonbasic indices. There are unique coefficients $b_i, a_{ij} \in \mathbb{R}$ $(i \in \mathcal{B}, j \in \mathcal{N})$ such that each feasible vector $\mathbf{x} \in \Lambda$ satisfies*

$$x_i = b_i - \sum_{j \in \mathcal{N}} a_{ij} x_j, \quad \forall i \in \mathcal{B}. \tag{6}$$

Proof. Consider a linear program with constraint equations $\mathbf{C}\mathbf{x} = \mathbf{c}$ where $\mathbf{C} \in \mathbb{R}^{m \times n}$, $\mathbf{c} \in \mathbb{R}^m$ and $\operatorname{rank}(\mathbf{C}) = m$. It can be written as $\mathbf{B}\mathbf{x}_{\mathcal{B}} + \mathbf{N}\mathbf{x}_{\mathcal{N}} = \mathbf{c}$ where $\mathbf{B}$ is an invertible matrix and $\mathbf{x}_{\mathcal{B}}$, $\mathbf{x}_{\mathcal{N}}$ are vectors of basic and nonbasic variables, respectively [4]. This implies that $\mathbf{x}_{\mathcal{B}} = \mathbf{B}^{-1}\mathbf{c} - \mathbf{B}^{-1}\mathbf{N}\mathbf{x}_{\mathcal{N}}$. Assume that also $\mathbf{x}_{\mathcal{B}} = \mathbf{d} - \mathbf{D}\mathbf{x}_{\mathcal{N}}$ for some $\mathbf{D} \in \mathbb{R}^{m \times (n-m)}$, $\mathbf{d} \in \mathbb{R}^m$. Setting nonbasic variables to zeros gives the (only) basic solution corresponding to basis $\mathcal{B}$, hence $\mathbf{d} = \mathbf{B}^{-1}\mathbf{c}$. This further implies that $\mathbf{D}\mathbf{x}_{\mathcal{N}} = \mathbf{B}^{-1}\mathbf{N}\mathbf{x}_{\mathcal{N}}$ for all $\mathbf{x}_{\mathcal{N}} \in \mathbb{R}^{n-m}$, which holds only if $\mathbf{D} = \mathbf{B}^{-1}\mathbf{N}$. □

Note that coefficients a_{ij} and b_i in Theorem 1 correspond to elements in the simplex tableau composed for basis $\mathcal{B}$. In what follows, we examine how they relate to the structure of the LP relaxation diagram.

We say that a basic variable x_i *depends* on a nonbasic variable x_j if $a_{ij} \neq 0$. Two useful corollaries can be obtained from Theorem 1.

Corollary 2. *Let $\mathcal{B} \subset I$ be a basis of (3), $\mathcal{N} = I \setminus \mathcal{B}$ and $\mathbf{y}, \mathbf{z} \in \Lambda$. If $y_i = z_i$ for all $i \in \mathcal{N}$, then $\mathbf{y} = \mathbf{z}$.*

Corollary 3. *Let $\mathcal{B} \subset I$ be a basis of (3), $\mathcal{N} = I \setminus \mathcal{B}$ and $\mathbf{y}, \mathbf{z} \in \Lambda$. Let there be $k \in \mathcal{N}$ such that $y_k \neq z_k$ and $y_i = z_i$ for all $i \in \mathcal{N} \setminus \{k\}$. For $j \in \mathcal{B}$, if x_j is a basic variable in (6) which depends on nonbasic variable x_k, then $y_j \neq z_j$.*

The first corollary states that a feasible vector is fully determined by its nonbasic components. The second corollary states that if two feasible vectors have the same nonbasic components except one, then they differ in all basic components which depend on the distinctive nonbasic component.

Given a basis $\mathcal{B}$, two special elements in (V, E) are important for expressing basic node variables in the form (6).

- An object $u \in V$ is called a *dependency root* if each basic node variable $x_{u;k}$ $(k \in \{0, 1\})$ depends only on the other node $x_{u;1-k}$ and/or on edge variables in object pairs adjacent to u. Two possible configurations forming a dependency root are depicted in Figure 3.

- An object pair $\{u, v\} \in E$ is called a *dependency object pair* if there are exactly two nonbasic edges $x_{uv;k\ell}$ which are either "parallel" or "intersecting" as shown in Figure 4.

Note that Figure 3(b) and Figure 4 show only snippets of an LP relaxation diagram, not standalone diagrams with a valid basis.

Define a *dependency graph* $D(V, E, \mathcal{B}) = (V, E')$ as the subgraph of (V, E) where E' consists of all dependency object pairs in E. Moreover, for $u \in V$, define $D_u(V, E, \mathcal{B})$ as the connected component of $D(V, E, \mathcal{B})$ containing u. An example of a LP relaxation diagram and the induced dependency graph is depicted in Figure 6. The following theorem gives its characterization.

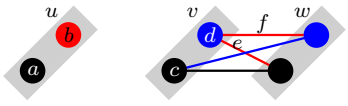
Theorem 4. *Let $(V, E, \boldsymbol{\theta})$ be an instance of binary energy minimization and let $\mathcal{B}$ be a basis of its LP relaxation. Then, $D(V, E, \mathcal{B})$ has the following structure.*

- *Each component has at most one cycle,*

- *if a component is a tree, it contains exactly one dependency root, and*

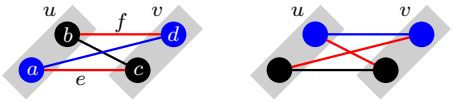
- *if a component has a cycle, it does not contain any dependency root.*

Moreover, each basic node in an object $u \in V$ depends only on nonbasic variables located in the following elements:



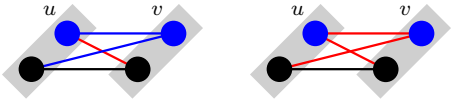
(a) (b)

Figure 3. Objects u and v are dependency roots since (a) $a = 1 - b$, (b) $d = e + f$ and $c = 1 - e - f$.



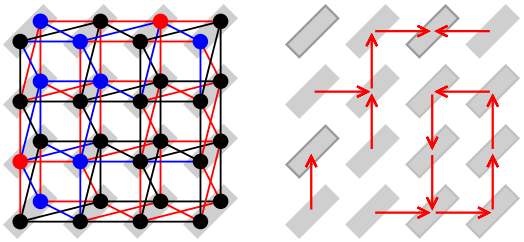
(a) (b)

Figure 4. Two (a) parallel or (b) intersecting nonbasic edges allow to delegate expressing basic nodes in u to expressing basic nodes in v (and vice versa). For example, $a = e - f + d$ and $b = f - e + c$.



(a) (b)

Figure 5. (a) Objects u and v do not depend on a single nonbasic edge in $\{u, v\}$. (b) Three nonbasic edges in $\{u, v\}$ induce dependency roots u, v.



(a) (b)

Figure 6. An example of (a) an LP relaxation diagram and (b) its dependency graph. Dependency object pairs are displayed as directed edges oriented towards the root or cycle of the component. Edges in a cycle follow one of the orientations which closes a loop. Dependency roots as well as objects in cycles are highlighted.

- *objects and object pairs forming the path from u to the dependency root or the cycle of $D_u(V, E, \mathcal{B})$,*
- *objects and object pairs of the cycle of $D_u(V, E, \mathcal{B})$, and*
- *the object pair containing nonbasic edges establishing the root of $D_u(V, E, \mathcal{B})$ (refers to Figure 3(b)).*

Proof. We list all locally admissible configurations of basic and nonbasic variables (up to permutations). After that, we derive how a basic node is expressed in the form (6).

As the first step, observe there is at most one nonbasic node within one object. Since nonbasic nodes attain zero in feasible solutions, two nonbasic nodes do not fulfill constraint (4b). In addition, each object pair $\{u, v\}$ contains one, two or three nonbasic edges. This is proved by contradiction. Let all four edges $x_{uv;ij}$, $i, j \in \{0, 1\}$ be nonbasic (and thus of zero value). Constraint (4a) forces $x_{u;0} = x_{u;1} = 0$ which again does not fulfill (4b). On the other side, let all four $x_{uv;ij}$ be basic. Take a feasible solution $\mathbf{y}$ where $y_{uv;00} = y_{uv;11} = 1/2$ and $y_{uv;01} = y_{uv;10} = 0$. Create vector $\mathbf{z}$ from $\mathbf{y}$ as follows: set $z_{uv;00} = z_{uv;11} = z_{uv;01} = z_{uv;10} = 1/4$ and copy values of all other components. Since $\mathbf{z}$ is again a feasible solution, pair $\mathbf{y}$, $\mathbf{z}$ contradicts Corollary 2.

We further inspect, how the configurations participate in expressing nodes in the form (6). If there is only one nonbasic edge (see Figure 5(a)), we can prove that none of the basic nodes in the adjacent objects depends on it. Consider the same vectors $\mathbf{y}$ and $\mathbf{z}$ as specified before. W.l.o.g., let $x_{uv;10}$ be the only nonbasic edge within $\{u, v\}$ and, w.l.o.g., let $x_{u;0}$ depend on $x_{uv;10}$. Observe that $y_{uv;10}$ differs from $z_{uv;10}$, but $y_{u;0}$ equals $z_{u;0}$. This contradicts Corollary 3.

If there are two nonbasic edges with a shared end-node in object v (Figure 3(b)), then, by definition, v is a dependency root. Three nonbasic edges induce one pair of edges with a shared end-node in each of the objects (Figure 5(b)), two dependency roots are thus formed.

The only configurations of nonbasic edges allowing dependance of basic nodes on more distant nonbasic variables are those ones defining dependency object pairs (Figure 4). In this case, nodes in one object can be locally expressed using nodes in the opposite object (and vice versa). For a parallel dependency object pair (Figure 4(a)) we derive

$$x_{u;0} = x_{uv;00} - x_{uv;11} + x_{v;1},$$

$$x_{u;1} = x_{uv;11} - x_{uv;00} + x_{v;0}.$$

Analogously, for an intersecting object pair (Figure 4(b)) we get

$$x_{u;0} = x_{uv;01} - x_{uv;10} + x_{v;0},$$

$$x_{u;1} = x_{uv;10} - x_{uv;01} + x_{v;1}.$$

Since $x_{uv;00}$ and $x_{uv;11}$ are nonbasic variables, to complete the expression of $x_{u;0}$ and $x_{u;1}$ as (6) would require to express $x_{v;1}$ and $x_{v;0}$. This again could delegate the process to a neighboring object. Such a procedure produces a complete expression of a basic variable when a dependency root is reached, as demonstrated in Figure 7(a). Following the path from u to w yields

$$x_{u;0} = x_{uv;00} - x_{uv;11} + x_{vw;10} - x_{vw;01} + x_{w;1}.$$

If a sequence of objects $(u_i)_{i=1}^k$ forms a path in (V, E), each $\{u_i, u_{i+1}\}$, $1 \leq i \leq k-1$ is w.l.o.g. an intersecting dependency object pair and $x_{u_k;0}$ is the only nonbasic node among all nodes in u_i's, then basic nodes in u_1 are expressed by nonbasic variables as follows.

$$x_{u_1;0} = \sum_{i=1}^{k-1} x_{u_i u_{i+1};01} - \sum_{i=1}^{k-1} x_{u_i u_{i+1};10} + x_{u_k;0}, \tag{7}$$

$$x_{u_1;1} = 1 - \sum_{i=1}^{k-1} x_{u_i u_{i+1};01} + \sum_{i=1}^{k-1} x_{u_i u_{i+1};10} - x_{u_k;0}. \tag{8}$$

The second possibility how to terminate the described process of expressing a basic node is to close a loop. This is demonstrated in Figure 7(b) where

$$x_{u;0} = \frac{1}{2}(x_{uv;00} - x_{uv;11} + x_{vs;10} - x_{vs;01} + x_{st;11} - x_{st;00} + x_{tu;00} - x_{tu;11} + 1).$$

In general, let $(u_i)_{i=1}^k$ be a sequence of objects forming a cycle in (V, E). W.l.o.g., let $\{u_i, u_{i+1}\}$ be intersecting for $1 \leq i \leq k-1$ and let $\{u_1, u_k\}$ be parallel. Assume than all nodes in u_i's are basic. Node $x_{u_1,0}$ can be again expressed as (7), but this is not yet the form (6) since $x_{u_k;0}$ is basic node. However, if we substitute

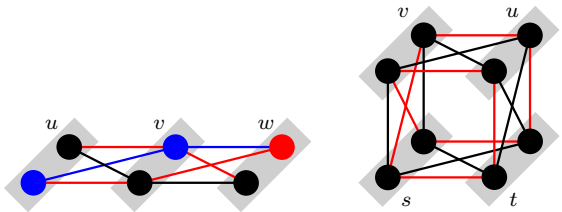
$$x_{u_k;0} = x_{u_k u_1;00} - x_{u_k u_1;11} + 1 - x_{u_1;0},$$

we obtain an equation which gives

$$x_{u_1;0} = \frac{1}{2}\Big(1 + \sum_{i=1}^{k-1} x_{u_i u_{i+1};01} - \sum_{i=1}^{k-1} x_{u_i u_{i+1};10} + x_{u_k u_1;00} - x_{u_k u_1;11}\Big). \tag{9}$$

Node $x_{u_1;1}$ can be now expressed as (6) using the equality $x_{u_1;1} = 1 - x_{u_1;0}$. Note it was essential that $\{u_1, u_k\}$ is parallel. Changing it to intersecting would result in an equation where both, $x_{u_1;0}$ and $x_{u_1;1}$ are missing. To express these variables, it is necessary to have a dependency cycle where all basic edges and nodes form a connected component in the underlying microstructure.

Finally, the most general situation when there is a path starting at object u and leading to object v which is part of a cycle (and it is the first such an object in the path) is handled in two steps. First, nodes in u are expressed along to the path to v as (7), (8), and second, nodes of v are expressed within the cycle as (9). To finish the proof it suffices to realize it is not possible to have more choices of following dependency object pairs to a root or cycle since (6) is unique for each basic variable. □



(a) (b)

Figure 7. Expressing a nonbasic node variable in u along a path formed of dependency object pairs terminates only if (a) a dependency root is reached or (b) a loop is closed.

Theorem 5. *Given a binary energy minimization $(V, E, \boldsymbol{\theta})$ and a basis $\mathcal{B}$ of its LP relaxation, each basic edge of an object pair $e = \{u, v\} \in E$ depends only on a subset of nonbasic variables in e, $C_u = D_u(V, E, \mathcal{B})$, $C_v = D_v(V, E, \mathcal{B})$ and on nonbasic edges establishing dependency roots of C_u and C_v.*

Proof. If there are at least two nonbasic edges in $e = \{u, v\}$, then each basic edges in e is adjacent to a nonbasic edge in some node (inspect all possible configurations in Figures 3(b), 4(a), 4(b), 5(b)). It is thus possible to express every basic edge in e as a linear function of one nonbasic edge and one (basic or nonbasic) node. For example, we derive in Figure 4(a)

$$x_{uv;01} = x_{v;1} - x_{uv;11}. \tag{10}$$

If there is only one nonbasic edge in e (Figure 5(a)), then the basic edge which is not adjacent to it in any node is expressed using two nodes. In Figure 5(a), it holds

$$x_{uv;01} = x_{u;0} - x_{v;0} + x_{uv;10}. \tag{11}$$

Formula (6) for any basic edge in e is obtained if we substitute the basic nodes with their expressions (6). □

Theorem 6. *Given a binary energy minimization $(V, E, \boldsymbol{\theta})$, a feasible basis $\mathcal{B}$ of its LP relaxation, $i \in \mathcal{B}$ and x_i expressed in the form (6). It holds $b_i \in \{0, \frac{1}{2}, 1\}$ and, for all $j \in I \setminus \mathcal{B}$, $a_{ij} \in \{0, \pm\frac{1}{2}, \pm 1, \pm 2\}$.*

Proof. Possible values of b_i are prescribed by the half-integrality of basic feasible solutions [1]. As observable in Figure 6(a), values $\frac{1}{2}$ are assigned to basic variables in dependency components with a cycle and to basic edge variables in object pairs adjacent to such components.

By checking expressions (7), (8) and (9) derived in the proof of Theorem 5, we find that a_{ij} is in $\{0, \pm\frac{1}{2}, \pm 1\}$ for each basic node. The same conclusion holds for basic edges which can be expressed as (10). Consider a basic edge expressed as (11). Substitute for $x_{u;0}$ and $x_{v;0}$ their expressions (6). If $D_u(V, E, \mathcal{B})$ and $D_v(V, E, \mathcal{B})$ differ, then the

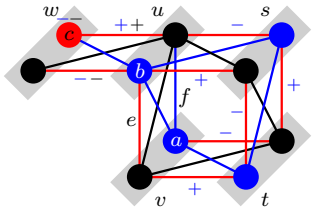
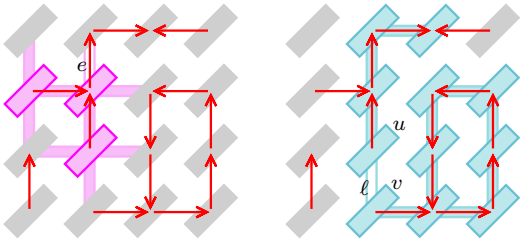


Figure 8. The blue signs show dependency of a on nonbasic variables (each dependency coefficient is either $+1$ or -1), while the black signs show dependency of b. Since $f = e + a - b$, edge f does not depend on node c, neither on the nonbasic edges in $\{w, u\}$. If the length of the dependency path from w to v was an even number, the dependency coefficient of f on c would be 2.



(a) (b)

Figure 9. (a) The highlighted elements contain all the basic variables that depend on a nonbasic edge variable in object pair e. (b) A basic variable in object pair $\ell = \{u, v\}$ is expressed along paths from u and v to the root or cycle.

substituted expressions do not share variables with nonzero coefficients. However, if $D_u(V, E, \mathcal{B}) = D_v(V, E, \mathcal{B})$ then a nonbasic variable x_q with a nonzero coefficient can be present in both expressions, with coefficients of the same absolute value, hence the dependency of $x_{uv;01}$ on x_q either vanishes or the dependency coefficient is doubled. This is demonstrated in Figure 8. □

5. Algorithm

Section 4 has established theoretical basis for a special version of the simplex algorithm executed for the LP relaxation of a binary energy minimization $(V, E, \boldsymbol{\theta})$. Here we put together all needed ingredients and give a description of the algorithm itself.

The algorithm performs steps over the LP relaxation diagram induced by the actual feasible basis $\mathcal{B}$. The diagram enables an efficient retrieval of dependency coefficients which corresponds to elements in the simplex tableau. For a faster traversal of the diagram, the direction of dependency object pairs towards the dependency root or cycle, as given in Figure 6(b), is maintained. Each node and edge variable is assigned by a boolean flag determining its basic/nonbasic status. Black/blue color of basic variables is recorded only for nodes. Color of basic edges can be derived from a local context.

For a nonbasic variable, it is possible to locate all basic variables that depend on it by traversing the dependency component in the direction opposite to the orientation of dependency object pairs in Figure 9(a). Similarly, for a basic variable, it is possible to traverse all nonbasic variables on which it depends. In this case a directed path to the dependency root or cycle is followed in one or two components, as can be seen in Figure 9(b). The dependency coefficient can also be determined by traversing components.

We apply a modified Dantzig's pivoting rule. The original variant chooses a nonbasic variable with the lowest negative cost (such a variable enters the basis), however, this is time-consuming. We rather use r doubly linked lists $\mathcal{L}_1, \ldots, \mathcal{L}_r$ and negative threshold values $\tau_1 < \tau_2 < \ldots < \tau_r = 0$. An objective cost θ is stored in $\mathcal{L}_s$ if $\tau_{s-1} < \theta \leq \tau_s$ (where $\tau_0 = -\infty$). The list can be found by a binary search in $\lceil \log_2 r \rceil$ steps. From performance reasons, we also do not implement any anti-cycling strategy. This is a normal approach in general LP solvers. No cycling has been observed during our numerous experiments. The experiments further revealed that it is sufficient to choose $r = 8$. Setting $r > 8$ has not resulted in any considerable improvement for the tested instances. It has also turned out that the number of iterations performed by the algorithm is nearly identical as in the case of the regular Dantzig rule.

The initial feasible basis follows the uniform pattern in Figure 1. A description of the algorithm follows.

Initialization. Create the initial feasible basis. For every object $u \in V$, mark $x_{u;1}$ as a nonbasic node. For every object pair $\{u, v\} \in E$, mark $x_{uv;11}$ as a nonbasic edge. All the other nodes and edges are basic variables. Apply a reparametrization (see section 2.1 of [8]) to $\boldsymbol{\theta}$ which changes costs of all basic variables to zero. Insert negative costs into lists $\mathcal{L}_i$.

Entering variable selection. Find the first nonempty list $\mathcal{L}_i$ and take its first element. The nonbasic variable x_e referenced by it is the *entering variable*. Assume it is located in an object or object pair e.

Leaving variable selection. Mark each object u that passes through e when traversing from u to the root or cycle, respectively. Search through the dependent variables and find a leaving variable (x_ℓ in an object or object pair ℓ) fulfilling the *minimum ratio test*. Finish the search prematurely if a zero ratio is found. Unmark all the marked objects.

Iteration. Update costs of all variables on which x_ℓ depends. Remove positive costs from lists, append costs that became negative. If a negative cost in $\mathcal{L}_i$ is changed and

belongs now to $\mathcal{L}_j$, remove it form $\mathcal{L}_i$ and append it to $\mathcal{L}_j$. Update basic/nonbasic flag of x_e and x_ℓ. Update the representation of dependency components. Update colors of nodes.

Termination. Finish when all lists $\mathcal{L}_i$ are empty.

Time of one iteration is proportional to size of the involved dependency components. A good performance can be expected for those energy minimization instances which induce dependency components whose average size is either constant or slowly growing in size of the energy minimization graph. The experimental evaluation showed that such small components usually emerge for nonsubmodular instances and submodular instances where pairwise potentials are less dominant. On the other hand, submodular instances with more dominant pairwise potentials (*e.g.* segmentation instances), which result in solutions containing large regions assigned by the same label, tend to form large dependency components during the late iterations of the algorithm. In the next section, we report details on the favorable type of problems.

6. Experiments

We have implemented the specialized simplex algorithm (SA) in C++. It supports the creation of general graphs and computes with double precision floating point numbers. The max-flow based QPBO implementation by Kolmogorov [18] (BK) is used for comparison. All the sources were compiled in Microsoft Visual Studio 2012 and run on a notebook with Intel Core i5-4300M 2.6 GHz, 12 GB RAM and 64-bit Windows 7. The evaluation is done for vision problems and random data.

6.1. Vision problems

Test data are taken mainly from the empirical comparison of max-flow algorithms [15] which targets a wide range of max-flow algorithms based on augmenting-path, push-relabel or pseudoflow principles. Since their performance relative to BK algorithm is known, it is possible to compare SA with them. The data are available at [20] in the form of QPBO graphs. We turned them into an energy minimization format.

Decision Tree Field (DTF) is a recently introduced model by Nowozin *et al*. [11] that combines random forests and conditional random fields. 99 instances are available, giving a sufficiently representative sample. The problem is nonsubmodular and involves dense graphs. We measure performance of SA relative to BK. Ratios of running times (SA/BK) sorted in ascending order are plotted in Figure 10. SA is at least two times faster than BK for 80 instances. It is slower only for 4 instances. The average time of the best performing max-flow algorithm over DTFs reported in [15]

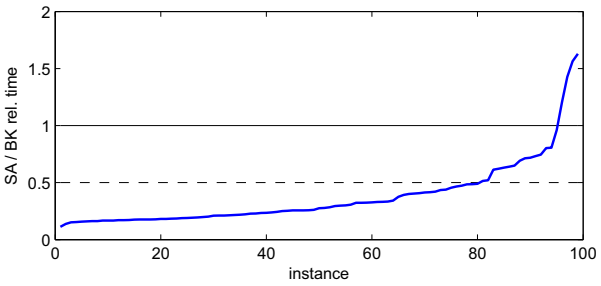


Figure 10. SA/BK relative time for Decision Tree Fields.

is about 46% of the average time of BK (provided that the initialization time is counted). SA achieves 31% of BK.

The amount of samples available for the next problems is considerably lower. We have collected two instances of *Super Resolution* (a sparse nonsubmodular problem) and six instances of *Deconvolution* with 3x3 or 5x5 blur-kernel (a dense nonsubmodular problem, graph connectivity is 24 or 80, respectively). These instances were considered by Rother *et al*. in [13]. Some of them (#3, #7, #9) are provided at [20]. *Shape Fitting* [10] is a representative of a sparse submodular problem. An instance (`LB07-bunny-sml`) has been taken at [19].

Measured absolute running times are in Table 1. Two selected DTFs are included there for comparison. To demonstrate a huge gain over a general LP solver, running times of IBM ILOG CPLEX 12.6 for the instances are also included. We tested the primal as well as the dual simplex method. Time limit of 10 minutes was applied for each computation.

We can see that BK performs better than SA for instances #3 – 8. However, the responsiveness of SA is within reasonable limits. The situation is different for deconvolution with a 5x5 blur-kernel. BK outperforms SA for #9. Conversely, SA outperforms BK for #10. Note that there are max-flow algorithms better than BK for this denser variant of deconvolution (approx. 1.5 times faster [15]).

6.2. Scalable random data

Here we study how the performance of SA scales in the size of the input graph. For this purpose we need a dataset of instances with equally growing number of objects. We generate a subset of square grids from 10×10 to 500×500 with 8-neighborhood system. Unary potentials are generated as independent gaussians $\theta_{u;0}, \theta_{u;1} \sim \mathcal{N}(0,1)$. Pairwise potentials are set to zero for $\theta_{uv;00}$ and $\theta_{uv;11}$. Values of $\theta_{uv;01}$ and $\theta_{uv;10}$ are generated as $\mathcal{N}(0,2)$. This setup is an instance of the Ising model with mixed potentials. Overall, we obtain sparse nonsubmodular inputs containing about 66% undecided variables in the optimal LP relaxation solution.

The evaluation is also done for the variant of SA which strictly follows Danzig's pivoting rule (a binary heap is used to store negative objective costs in this case). This variant

#	description	objects	obj. pairs	SA [ms]	BK [ms]	CPLEX primal [ms]	CPLEX dual [ms]
1	dtf-78	7776	217414	**149**	1337	time limit exceeded	96315
2	dtf-94	8384	234237	**276**	1523	time limit exceeded	129574
3	sup. res. (4-con)	5246	10345	3.7	**1.5**	218	421
4	sup. res. (8-con)	5246	20545	10.4	**3.2**	483	1388
5	shape fit. (6-con)	805800	2391242	628	**135**	out of memory	out of memory
6	deconv. 3x3	1024	11346	5.5	**2.2**	1451	296
7	deconv. 3x3	1000	10968	4.9	**3.1**	1185	358
8	deconv. 3x3	1000	10968	4.6	**2.5**	858	405
9	deconv. 5x5	1000	33900	71.7	**8.3**	17924	1872
10	deconv. 5x5	1024	35400	**7.5**	23.9	22730	2496
11	deconv. 5x5	880	29820	**32.1**	46.4	1841	1342

Table 1. Running time of SA, BK and CPLEX 12.6 for vision instances.

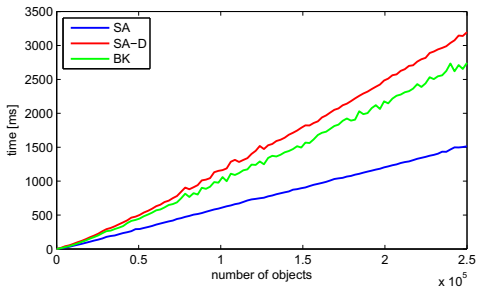


Figure 11. Running time of SA, SA-D and BK for random grids.

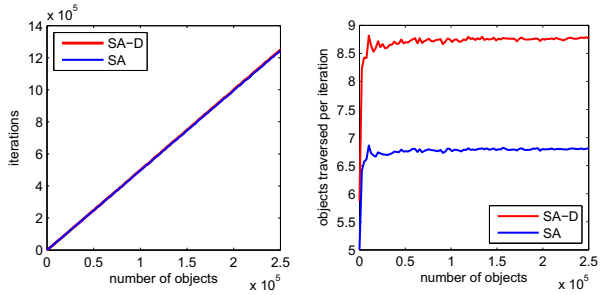


Figure 12. SA and SA-D: the number of performed iterations and the average number of objects traversed per iteration.

is denoted as SA-D. The dependency of running time on the number of objects is plotted in Figure 11. SA performs better than BK and the measured time is linear. In contrast, a non-linearity caused by the binary heap usage is observable for SA-D. It is also interesting that the function for SA is the smoothest one.

Figure 12 reveals two dependencies. The number of iterations performed by the simplex algorithm is linear. It is almost identical for both, SA and SA-D, the displayed functions coalesce. Moreover, the average number of objects traversed within one iteration is almost constant. Surprisingly, the constant is greater for SA-D.

In conclusion, SA demonstrated a very good performance and stability in this test.

7. Conclusion

We have presented a graph-based version of the simplex method for pairwise energy minimization with binary variables. The experiments confirmed that the proposed algorithm is efficient for certain types of vision problems. Outperforming other solvers on DTF instances represents an immediate practical benefit.

We believe the method has a very good potential for further research. We obtained an algorithm competitive with best solvers based on finding maximum flow in a network, which has been intensively studied by many researches for a long time. Our algorithm has not undergone such evolution. Promising opportunities for improvements are thus expectable. For example, a different pivoting rule might result in a smaller size of dependency components, a more advanced representation of components might reduce the number of traversed objects when searching for the leaving variable, *etc*. The algorithm may also be suitable for parallelization.

An interesting question is whether the approach can be efficiently generalized to the LP relaxation of multi-label energy minimization problems. The situation is surely more difficult. More labels induce richer relations among basic and nonbasic variables. Except that, the LP relaxation becomes as hard as general LP [12] and, as a consequence, components of optimal solutions are, roughly speaking, arbitrary fractions. On the other hand, known methods for solving the LP relaxation of multi-label problems like message passing [7] are considerably slower than max-flow algorithms. Even a slower retrieval of simplex tableau elements (*e.g.* by solving subsystems of linear equations within dependency components) could still result in a method with better performance. The presented applicability of the simplex algorithm in the binary setting should encourage such considerations.

Acknowledgment

The author was supported by the Czech Science Foundation project P202/12/2071.

References

[1] E. Boros and P. L. Hammer. Pseudo-Boolean optimization. *Discrete Applied Mathematics*, 123(1-3):155–225, 2002. 1, 5

[2] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, Sept. 2004. 1

[3] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, Nov. 2001. 1

[4] G. Dantzig and M. Thapa. *Linear Programming 1: Introduction*. Springer, 1997. 1, 2, 3

[5] D. Goldfarb and J. Hao. A primal simplex algorithm that solves the maximum flow problem in at most nm pivots and $O(n^2m)$ time. *Mathematical Programming*, 47(1-3):353–365, 1990. 1

[6] H. Ishikawa. Exact optimization for Markov random fields with convex priors. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 25(10):1333–1336, Oct. 2003. 1

[7] V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 28(10):1568–1583, 2006. 8

[8] V. Kolmogorov and C. Rother. Minimizing nonsubmodular functions with graph cuts-a review. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 29(7):1274–1279, 2007. 1, 6

[9] A. Koster, C. P. M. van Hoesel, and A. W. J. Kolen. The partial constraint satisfaction problem: Facets and lifting theorems. *Operations Research Letters*, 23(3–5):89–97, 1998. 2

[10] V. Lempitsky and Y. Boykov. Global optimization for shape fitting. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, June 2007. 7

[11] S. Nowozin, C. Rother, S. Bagon, T. Sharp, B. Yao, and P. Kohli. Decision tree fields. In D. N. Metaxas, L. Quan, A. Sanfeliu, and L. J. V. Gool, editors, *IEEE International Conference on Computer Vision*, pages 1668–1675. IEEE, 2011. 7

[12] D. Průša and T. Werner. Universality of the local marginal polytope. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 37(4):898–904, April 2015. 8

[13] C. Rother, V. Kolmogorov, V. S. Lempitsky, and M. Szummer. Optimizing binary MRFs via extended roof duality. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2007. 1, 7

[14] D. Schlesinger and B. Flach. Transforming an arbitrary MinSum problem into a binary one. Technical Report TUD-FI06-01, Dresden University of Technology, Germany, April 2006. 1

[15] T. Verma and D. Batra. Maxflow revisited: An empirical comparison of maxflow algorithms for dense vision problems. In *Proceedings of the British Machine Vision Conference*, pages 61.1–61.12. BMVA Press, 2012. 1, 7

[16] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008. 1

[17] T. Werner. A linear programming approach to max-sum problem: A review. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 29(7):1165–1179, July 2007. 1

[18] V. Kolmogorov – web pages. http://pub.ist.ac.at/~vnk/. 7

[19] University of Western Ontario – Max-flow problem instances in vision. http://vision.csd.uwo.ca/data/maxflow/. 7

[20] T. Verma, D. Batra – MaxFlow Revisited. http://ttic.uchicago.edu/~dbatra/research/mfcomp/. 7

IJDAR (2016) 19:253–267
DOI 10.1007/s10032-016-0269-z

ORIGINAL PAPER

Online recognition of sketched arrow-connected diagrams

Martin Bresler[1] · **Daniel Průša**[1] · **Václav Hlaváč**[2]

Received: 27 May 2015 / Revised: 15 March 2016 / Accepted: 1 May 2016 / Published online: 19 May 2016

Abstract We introduce a new, online, stroke-based recognition system for hand-drawn diagrams which belong to a group of documents with an explicit structure obvious to humans but only loosely defined from the machine point of view. We propose a model for recognition by selection of symbol candidates, based on evaluation of relations between candidates using a set of predicates. It is suitable for simpler structures where the relations are explicitly given by symbols, arrows in the case of diagrams. Knowledge of a specific diagram domain is used—the two domains are flowcharts and finite automata. Although the individual pipeline steps are tailored for these, the system can readily be adapted for other domains. Our entire diagram recognition pipeline is outlined. Its core parts are text/non-text separation, symbol segmentation, their classification and structural analysis. Individual parts have been published by the authors previously and so are described briefly and referenced. Thorough evaluation on benchmark databases shows the accuracy of the system reaches the state of the art and is ready for practical use. The paper brings several contributions: (a) the entire system and its state-of-the-art performance; (b) the methodology exploring document structure when it is loosely defined; (c) the thorough experimental evaluation; (d) the new annotated database for online sketched flowcharts and finite automata diagrams.

✉ Martin Bresler
breslmar@fel.cvut.cz

[1] Center for Machine Perception, Faculty of Electrical Engineering, Czech Technical University in Prague, Technická 2, 166 27 Prague 6, Czech Republic

[2] Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague, Zikova 4, 166 36 Prague 6, Czech Republic

Keywords Diagram recognition · Online document analysis · Max-sum problem · Segmentation · Text/non-text separation · Flowcharts · Finite automata

1 Introduction

Research in handwritten document analysis has shifted from the recognition of plain text to recognition of more structured inputs such as mathematical and chemical formulas, music scores or diagrams. Even though recognizers with good precision have been presented, e.g. of mathematical formulas [2,21], the availability of diagram recognizers is still limited. One reason may be that diagram domains differ significantly and their structure is loose in comparison with, for example, mathematical formulas. This paper attempts to fill this gap. We present a general recognition system suitable for a variety of online sketched diagrams. We introduced the idea in [9]; since then, the system has evolved into a pipeline depicted in Fig. 1. We provide its comprehensive description, focusing on core principles proposed for segmentation, classification and structural analysis. A brief description and references are also given for those parts, in which existing solutions were adopted. Our experiments yield a thorough evaluation of the system accuracy, the discussion of failure types and the impact of particular steps on the overall performance.

An online input is considered to be a sequence of handwritten strokes, in which a stroke is a sequence of points captured by an *ink input device* between pen-down and pen-up events. The most common device is a tablet, a tablet PC or a smart board. Every stroke point is defined by its coordinates on the planar drawing canvas. Additional data like a time stamp or a pressure value may be provided. The output is a structure which syntactically describes the sketched diagram. Individ-

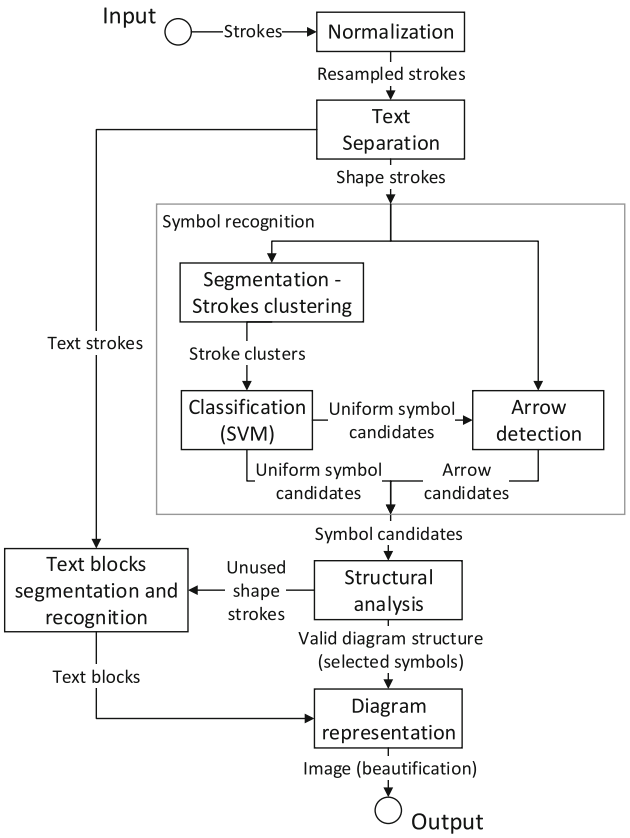


Fig. 1 Proposed recognition pipeline

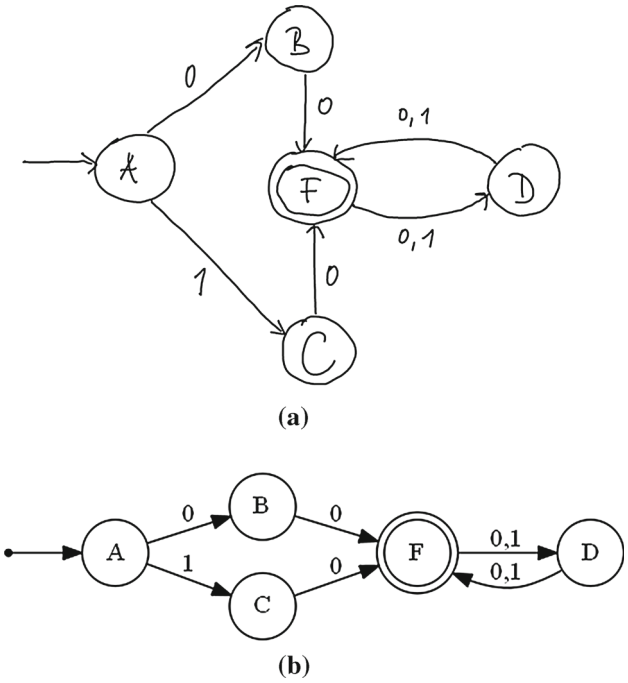


Fig. 2 An example of **a** a typical input and **b** the desired output of the proposed diagram recognizer

ual symbols are identified, and relations between them are detected. Additionally, text is divided into logical blocks with known meaning. Several formats can be used to represent the recognition result, as exchange formats for diagrams do not seem to be unified. We use the DOT graph description language, supported by the popular graph visualizer Graphviz.[1] An example of finite automata beautified by Graphviz is shown in Fig. 2

There has also been some research done in offline diagram recognition [31]. In that case, input is an image, and thus, any temporal information is missing. Offline recognition faces different challenges (especially in segmentation) and typically leads to different applications. This paper does not consider these topics.

We target domains which exhibit the following structure: the diagram consists of *symbols* connected by *arrows*, and there might be *text* labelling the symbols (the text is inside or along the border of a symbol) or arrows (the text is in a vicinity of an arrow). Although this structure is basic, there are various domains which fit it perfectly. We focus on two of them: flowcharts and finite automata. Other domains such as UML use case diagrams, Simulink diagrams, Concur Task Trees or business process diagrams fall into this group as well—we leave adaptation to these domains as future work. Domains with a different structure (e.g. music scores or chemical diagrams) or a structure further extending the basic constructs (e.g. diagrams with structured text inside symbols) are beyond the scope of this paper.

The paper is organized as follows. Section 2 briefly surveys diagram recognition systems. Section 3 defines the example domains and introduces datasets used. Section 4 describes our recognition system, following the recognition pipeline in Fig. 1. Specifically, Sect. 4.1 describes our text/non-text separation, Sect. 4.2 explains the symbol segmentation through strokes clustering, Sect. 4.3 introduces our symbol detectors, and Sect. 4.4 presents the structural analysis. Section 5 contains experimental results. Section 6 analyses the developed system and deals with failure cases, and Sect. 7 presents conclusions.

2 Related work

2.1 Mathematical formulas

Recognition of mathematical formulas is a useful example as it has brought seminal methods and successful recognizers. A practical example is the Math Input Panel delivered by Microsoft since Windows 7. The Competition on Recognition of Online Handwritten Mathematical Expressions (CROHME) has further boosted research and provides a

[1] http://graphviz.org/.

reliable comparison ground. The commercial winner of CROHME 2014 [26], MyScript,[2] achieves 62.7 % accuracy of correctly recognized formulas, while the non-commercial winner—Álvaro [2]—achieves 37.2 %.

Recognition of mathematical formulas and diagrams faces some similar problems. Individual symbols have to be segmented, recognized and embedded into the domain structure. However, the structure of mathematical formulas is strong and recursive, and grammars are thus suited to their expression [2,21]. In contrast, diagram structure is simpler and less formalized and grammars do not seem to be the best model for their capture.

2.2 Diagrams in general

Feng et al. [15] proposed a recognizer for online sketched electric circuits. Hypotheses for symbol segmentation and classifications are generated using hidden Markov models (HMM), and the selection of the best hypotheses subset relies on 2D dynamic programming. A drawback is an extensive search space due to a large number of hypotheses, and this makes the system slow and prohibits it from practical use. Sezgin and Davis [32] used similar approach for recognition of objects in sketches from various domains like stick figures, UML diagrams or digital circuits. They use specific stroke orderings to reduce the search space. Although multiple HMMs are used to model different sketching styles and thus different natural stroke orderings, so-called delayed strokes may impose a problem for this approach. ChemInk, a recognition system for chemical formula sketches [28], represents elements and bonds between them. A hierarchy of three levels of details is used: inkpoints, segments and candidate symbols. The final recognition is performed by a joint graphical model classifier based on conditional random fields (CRF), which combines features from the levels in the classification hierarchy. A similar approach was used by Qi et al. [30] to recognize simple diagrams. The advantage of such methods is the joint training of the classifier for all levels of features. It helps to incorporate the context into the classification; however, it makes the training of the system more difficult. Our approach differs. Although we train the symbol classifier independently on the structure, we do not make any hard decisions at this point. Relations between the symbols are defined later with the help of context. Using binary predicates of the max-sum labelling problem rather than pairwise features does not require additional training and yields optimal solutions. The model is thus much simpler and more open to adaptations.

Finally, there were attempts to develop universal formalisms for sketch recognition. LADDER [16] is a sketch description language that can be used to describe how shapes are drawn as well a the whole syntax specifying a domain. A multi-domain sketch recognition engine SketchREAD [1] is based on this language. Authors evaluated the capabilities of the engine on family trees and electrical circuits. The parsing is based on dynamically constructed Bayesian networks, and it combines bottom-up and top-down algorithms. Although this framework laid the foundations of multi-domain sketch recognition, it has limitations. Individual shapes must be composed solely of predefined primitives according to a fixed graphical grammar. Individual strokes must be thus decomposed into primitives. Although the framework is designed to be recoverable from this low-level errors, it still impose additional source of error.

2.3 Flowcharts and finite automata

To our knowledge, little work has been published in the domains of flowcharts and finite automata. Lemaitre et al. [22] proposed a grammar-based recognition system for flowcharts which uses the Description and MOdification of the Segmentation (DMOS) method for structured document recognition. The applied grammatical language Enhanced Position Formalism (EPF) provides a syntactic and structural description of flowcharts, which is used for joint symbol segmentation and classification. Carton et al. [10] further improved the system by combining structural and statistical approaches; they exploited the nature of the symbols in flowcharts, which are closed loops. Such closed loops are detected first and classified later using the structural approach. Although statistics are used, it is hard to find a suitable threshold determining whether a loop is really closed. Users often draw carelessly, and the appearance of symbols can be far from closed loops. Additional difficulties are caused by the need to divide strokes into line segments. Experiments demonstrated that the grammar-based approach still has trouble, with big uncertainty in the input. Experiments were performed on a benchmark database, which allows comparison. Work by Szwoch and Mucha [34] is another effort to recognize flowcharts using grammars. The authors assume that symbols consist of single strokes, and this simplification forbids experiments on the benchmark database, and thus, it cannot be compared with other methods.

Delaye [11] has recently introduced a purely statistical approach to diagram recognition based on strokes clustering and CRFs: clusters represent graph nodes. A hierarchical model is used by applying several values of clustering thresholds. The graphs created are trees, and thus, the task can be solved efficiently by the belief propagation algorithm, which makes the system extremely fast. However, the approach is purely statistical, which does not use information about the diagram structure. Inconsistent labellings can occur.

[2] https://dev.myscript.com/technology/math/.

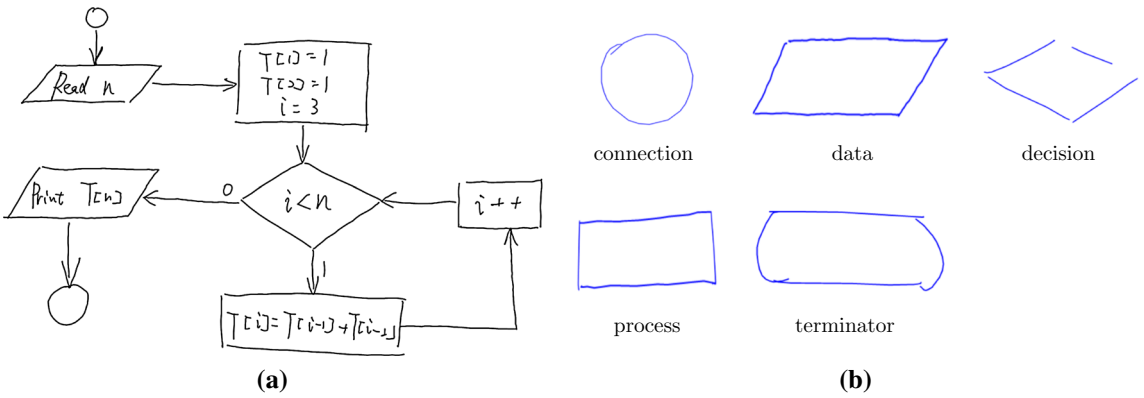


Fig. 3 Example of a flowchart (**a**) with examples of uniform symbol classes (**b**)

2.4 Alternative approaches

Though there are few systems directly comparable to ours, interest in diagram design/sketching is evident. Miyao and Maruyama [25] created a flowchart designer based on the iterative recognition principle. Input is processed in small pieces, and immediate user feedback is awaited. If the user does not indicate any error in the recognition, it is considered as ground truth. It is further possible to connect symbols by gestures and to input text for a selected symbol. This works well for flowcharts since symbols are loopy. However, the system puts unnatural requirements on the user.

In some cases, it is desired to keep the sketchy appearance of diagrams; thus, smart sketching tools allowing common editing operations without any formalization of the input have been proposed [3,29].

In conclusion, although there exist various systems for structured handwriting, there is no system for flowchart-like diagrams allowing practical use. Existing methods are either purely statistical or they rely on grammars which are too impractical for the minimalistic structure of diagrams.

3 Diagram structure and supported domains

The diagram recognition system we propose is general and can be used in several domains. Nevertheless, it requires adaptation for a chosen domain mostly by retraining classifiers. We introduce domains, together with benchmark databases used for training/testing of the system. Supported diagrams consist of symbols with a relatively stable appearance (called *uniform symbols*), interconnected by arrows. Arrows and uniform symbols may consist of arbitrary number of strokes, and both are possibly assigned a text label: text inside the uniform symbol or in the vicinity of the arrow. The domain syntax can bring additional constraints, e.g. forbid connecting some symbols. We worked in two diagram domains, flowcharts (FC) and finite automata (FA), for both of which there are freely available benchmark databases. FC and FA share important properties; however, their specifics should be considered when adapting the recognizer to achieve best results.

3.1 Flowcharts

Figure 3 shows an example flowchart, together with an enumeration of five uniform symbol classes: *connection*, *data*, *decision*, *process* and *terminator*.

Awal et al. [4] released a benchmark database of flowcharts (referenced as FC_A), on which several state-of-the-art methods were tested. The database consists of 327 diagrams drawn by 35 users: predefined diagram patterns representing well-known algorithms were used. The samples are divided into training and test datasets. The biggest disadvantage of the database is the lack of annotations providing information about the diagram structure. Only individual symbols are identified, and no temporal information is available. The data are of low quality (sampling frequency).

These deficiencies have motivated us to collect our own flowchart database and make it public:[3] we reference this database as FC_B. We collected 28 diagram patterns drawn by 24 users, resulting in 672 samples. They were divided into training, validation and test datasets. Some of the patterns were taken from the FC_A database, and others represent procedures for daily tasks. The database contains annotation of symbols and relations among them. Arrows are provided with connection points and heads annotated. Text blocks have their meaning attached.

3.2 Finite automata

The finite automata domain includes three uniform symbol classes: *state* (a circle), *final state* (two concentric circles) and *initial arrow* (straight arrow entering the initial state). The

[3] http://cmp.felk.cvut.cz/~breslmar/diagram_database.

text is usually simple—often just a single letter naming the state or indicating an input attached to the arrow. It may also contain a lower index. Arrows are typically curved, except the initial arrow which does not act as a connector of two states. An example is shown in Fig. 2.

No publicly available database of online sketched finite automata is known. We gathered and annotated our own database (referenced as the FA database) and make it public[3]. Compared to the previous release [9], the annotation was improved—arrows are provided with their connection points and heads annotated. The database contains samples of 12 diagram patterns drawn by 25 users, which results in 300 diagrams divided into training, validation and test datasets.

4 Recognition pipeline

4.1 Text detection and recognition

Separating text is motivated by the observation that the text bears almost no information about the diagram structure. Ideally, all text strokes are removed and the diagram without text is recognized. This divide and conquer strategy reduces computational complexity significantly since the number of strokes is much lower. Text strokes are replaced after recognition. The diagram structure helps forming text blocks and finding symbols, to which the blocks are assigned.

The text/non-text separation algorithm classifies single strokes into two classes—*text* and *shapes*. Although there exist quite precise algorithms for such separation [5,14,19,27,35], they are not able to separate all text strokes. Their accuracy achieved on the benchmark IAMonDo database[4] is between 97 and 98 %. Moreover, these classifiers tend to have a higher error in class *shapes* due to using unbalanced training datasets. Our goal is the opposite. We attempt achieving the minimal possible error rate for class *shapes* while a slightly higher error rate in class *text* is acceptable. The justification is that removing a shape stroke can easily cause a symbol not to be recognized, because it becomes incomplete. Some remaining text strokes are not a problem as symbol classifiers are robust enough to deal with noise.

We bias the classifier result, and thus, only strokes where the classifier is almost certain are marked as text. We implemented two classifiers performing best on the IAMonDo database [27,35] and tested them on flowcharts and finite automata. The best performing was the classifier proposed by Phan and Nakagawa [35], which uses unary features to classify individual strokes into two classes: *text* and *non-text*. It also considers relationships between adjacent strokes in the writing order and uses binary features to classify transitions between strokes into three classes: *text–text*, *text–non-text*, *non-text–non-text*. Since it can be seen as a sequence labelling task, BLSTM RNN classifiers are used to capture the global context. The probabilistic outputs from the two BLSTM neural networks are combined together to obtain the final labelling probability. It achieves precisions 98.62 % (98.75 % in the *shapes* class and 98.53 % in the *text* class) for FC_A, 99.53 % (98.94 % in *shapes*, 99.74 % in *text*) for FC_B and 98.84 % (99.85 % in *shapes*, 97.83 % in *text*) for FA in the unbiased case. We were able to reach a biased result in *shapes*/*text* class of 99.68/95.20, 99.41/98.75 and 100.00/93.31 % for FC_A, FC_B and FA, respectively. The bias value is simply used to increase the confidence of the label *shapes* for individual strokes. Its selection is a trade-off between the accuracy in both classes. Results for various bias values are shown in Fig. 4. We chose 0.99 for FC_A, 0.8 for FC_B and 0.7 for FA. Note that Phan and Nakagawa [35] suggest to use sum rule when combining the probability outputs of individual classifiers. Therefore, the final labelling confidence is from interval $[0, 4)$, which explains the high values of the bias.

Text recognition is performed when the diagram structure is already known. Strokes removed during the text/non-text separation step together with strokes not identified as a symbol are processed; the diagram structure provides enough information to group the unused strokes into text blocks. Two types of text blocks are distinguished, labelling the uniform symbol and labelling the arrow. The grouping is accomplished in two steps. Text blocks inside symbols are found first; text blocks labelling arrows are found next. Each text block labelling a uniform symbol U is determined by the area of the symbol. It includes all unused strokes whose bounding box centroid is located inside the bounding box of U. Text blocks labelling arrows are found easily by a grouping based on the spatiotemporal proximity. These blocks are salient objects; hence, it is easy to find a suitable grouping threshold. Afterwards, each text block is attached to the closest arrow. Recognition of text inside a block is beyond the scope of this paper.

4.2 Symbol segmentation

Segmentation is a process which divides strokes into subsets, each forming a particular symbol. Ideally, the subsets should be disjoint and cover all the strokes. However, it cannot reasonably be done without knowledge of the entire structure. It is unwise to make hard decisions at this early step, and so *over-segmentation* is better. It supplies a larger number of subsets, which may share some strokes. The final decision on which subsets fit the structure of the input diagram best is left for the structural analysis performed later. Our system needs to segment uniform symbols only: arrows are detected after

[4] http://www.iam.unibe.ch/fki/databases/iam-online-document-database.

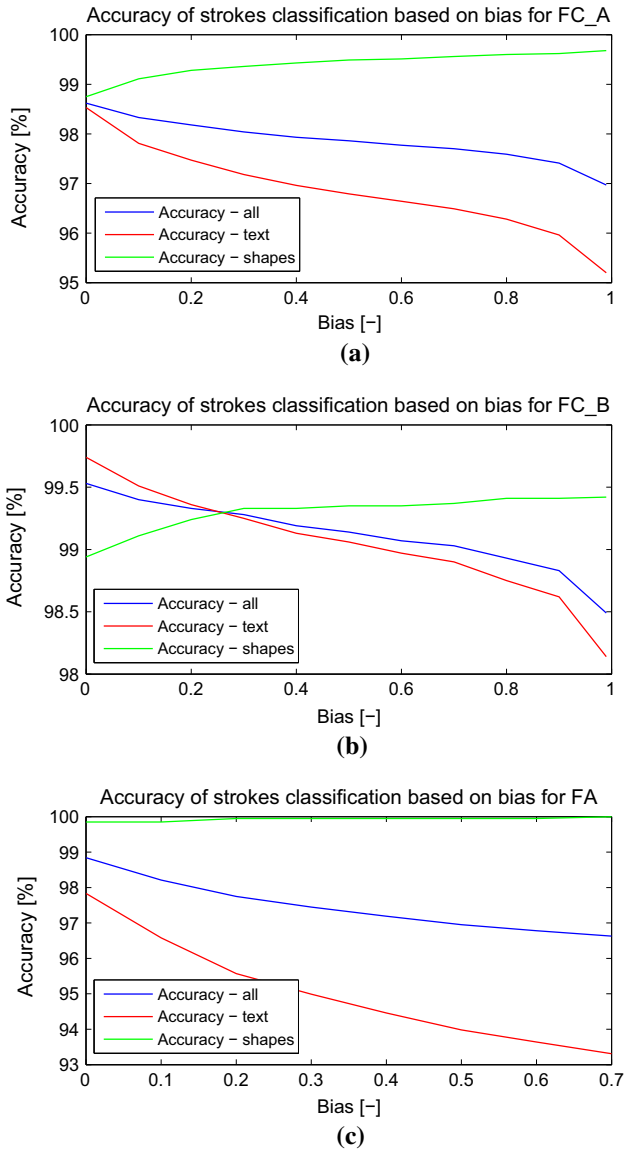


Fig. 4 Accuracy of stroke classification in relation to bias. Selection of the best bias is a trade-off between accuracy in *text* and *shapes* classes. **a** FC_A, **b** FC_B and **c** FA

the initial segmentation using the knowledge of recognized uniform symbols. The text is recognized even later when the entire structure is known.

The most common approach to over-segmentation works under the assumption that symbols are formed of spatially and temporally close strokes. Strokes grouping is performed iteratively. Within the first iteration, every single stroke forms a subset of size 1. Further, subsets of size k are created by adding a single spatially and temporally close stroke to subsets of size $k-1$. The maximal size of the subsets is given by the domain and user's drawing conventions, and spatial proximity is determined simply by Euclidean distance. Temporal proximity is hinted at by stroke indices in the drawing sequence. The approach has been used in our previous recognizer [9] as well as by others [2,15,28].

Delaye and Lee [13] showed that symbols may be segmented using single-linkage agglomerative clustering (SLAC) using a properly trained distance function defined as a weighted sum of several simple features. In addition to Euclidean distance, the features express difference between geometric and temporal characteristics of two strokes. The distance is defined as:

$$d(s,t;\mathbf{w}) = \sum_{i=1}^{k} w_i d_i(s,t), \tag{1}$$

where s and t are two given strokes and w_i is the weight for the feature d_i that needs to be learned. It is a hierarchical bottom-up clustering technique where larger clusters are created by iteratively merging the two closest clusters based on distance. The usage of a single-linkage clustering approach implies that the distance between two clusters is given by the distance between their two closest elements. This permits an efficient real-time implementation $\mathcal{O}(n^2)$, where n is the number of strokes. The clustering gives a final segmentation, which reaches typically lower recall rates. We improved it in our previous work [8], performing the over-segmentation by successive clusterings with varying thresholds. In comparison with grouping-based over-segmentation, this increased the precision (i.e. generated significantly fewer subsets), at the cost of only very slightly decreased recall. In the current implementation, we achieved 95.1/16.7, 98.4/27.5, 99.8/26.5 % recall/precision on FC_A, FC_B, FA databases. High recall is the main objective directly affecting the precision of the whole system while segmentation precision is a secondary objective, affecting mainly the speed of the system. The recall achieved allows overall high precision. The segmentation precision achieved shows that on average we do not generate more than 4–6 times more segments than is the true number of symbols, which is important for fast recognition.

4.3 Symbol recognition

Symbol recognition aims at classifying subsets of strokes (clusters) produced by segmentation. Each cluster is either assigned a symbol class or is rejected. We treat arrows in a special way since their form and shape vary, which is a difficulty for traditional classifiers. There are two stages. First, uniform symbols are recognized using a standard classifier. Second, arrows are detected as connectors between symbol candidates found earlier. Both recognizers/detectors provide an ordered list of symbol candidates. The structural analysis selects actual symbols from these lists.

4.3.1 Uniform symbol classifier

The classifier has to fulfil three requirements: (1) it has to be fast since many stroke clusters need to be processed; (2) the rejection ability is mandatory as many stroke clusters do not represent anything meaningful; (3) each classification produces a score (e.g. a posterior probability) to compare candidates' quality.

We used an off-the-shelf solution and combined the trajectory-based normalization and direction features proposed by Liu and Zhou [23] as a descriptor, which serves as the input to the multiclass SVM classifier. The descriptor is based on hybrid features capturing dynamic information as well as the visual appearance of symbols. Besides the trajectory-based normalization, we do not perform any particular pre-processing. The descriptor consists of 512 features; it was primarily designed for recognition of Japanese characters and thus works well for high number of visually similar symbol classes since individual Japanese symbols often differ in small details only. This property is desirable to enable rejection of incomplete symbols produced by the over-segmentation. The analogy with Japanese characters is illustrated in Fig. 5, and the achieved results confirm suitability of the selected descriptor. Although there are different descriptors available [12], we did not experiment with them because we achieved satisfactory results with the chosen solution. We trained the classifier with negative examples to obtain the rejection ability. The dataset of symbols for training was obtained by applying the strokes clustering introduced in Sect. 4.2. If the cluster of strokes is annotated as a uniform symbol in the database, it is labelled by that symbol. Otherwise, it is labelled as *no_match*, which denotes a negative example. Arrows as well as incomplete parts of symbols are labelled as negative examples. Because the FC_A database does not contain a validation set, we used a fivefold cross-validation. Therefore, we merged the training and validation datasets in case of FC_B and FA databases to have the same conditions.

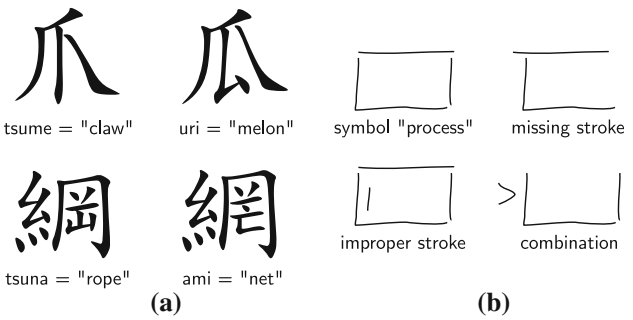


Fig. 5 Example of small differences between individual symbol classes in the case of Japanese characters (**a**) and uniform symbols in diagrams (**b**)

The number of negative examples is much higher than the number of uniform symbols. Moreover, they are very inhomogeneous. It is thus necessary to cluster them into subclasses. We employed k-means based on the descriptor to create m *no_match* subclasses, where m is domain dependent ($m = 30$ for flowcharts, $m = 20$ for finite automata). The appropriate values of m were estimated from the data while pursuing clusters with desirable properties such as homogeneity and separability. The larger number of symbol classes in the flowchart domain naturally results in a greater m. This brings a need for a modified loss function, which gives zero penalty when a negative example is classified into a different *no_match* subclass. Additionally, a greater penalty is required for misclassification of a uniform symbol as a negative example than in the opposite case. The ratio between these two penalties depends also on the ratio between the number of uniform symbols and negative examples. A properly chosen loss function can overcome the problem with an unbalanced database [9]. However, our current implementation uses artificially synthesized samples to balance the database. The samples were synthesized using the approach of Martín-Albo et al. [24]. It is based on kinematic theory and the distortion of the Sigma–Lognormal parameters in order to generate human-like synthetic samples. We generated up to 20 artificial samples from each uniform symbol taken from the training dataset. From all the synthesized samples of one class, we randomly chose a subset to get the desired number of symbols for training. This approach not only helps to balance the dataset, but also supplies additional information on handwriting and makes the classifier more robust. Therefore, we empirically set the smaller penalty to 1 and the bigger penalty to 2 just to increase recall at the cost of very small precision decrease. In the finite automata case, non-initial arrows and stroke subsets of final states have exactly the same appearance as initial arrows and states, respectively. To benefit from this knowledge of the domain, we excluded these two from negative examples, which increases the recall of the symbol classifier. Specifically, the recall increased from 96.23 to 98.98 % reported later in this section. Unfortunately, the FC_A database does not contain any time information, which is crucial for the synthesis; thus, artificial samples cannot be obtained for this database.

Without negative examples, the proposed classifier achieves precision of 98.9, 97.5 and 100.0 % for FC_A, FC_B and FA, respectively. If rejection is incorporated through negative examples, we keep the two topmost results of classification for each stroke cluster to make the symbol candidate detection more robust. It might happen that both results are *no_match*. Then, the corresponding cluster is rejected. This yields recall/precision of 94.13/43.13, 96.63/45.33 and 98.98/37.15 % for FC_A, FC_B and FA, respectively. The average recognition time per sample is 0.7 ms.

4.3.2 Arrow detector

As stated earlier, it is difficult to perform recognition of an arrow based on its appearance, even if several arrow subclasses are considered. This was our initial approach [7] which did not lead to satisfactory results. Some generic recognizers for arrows are also known [17,18,20,33]; however, they expect a fixed form of arrows—the number of strokes as well as the range of angles between them is restricted. This conforms to some domain specific solutions, but is impractical in general. The requirements on the user's drawing style are unnatural; a new model for each arrow form has to be defined, etc.

Our detector exploits the special property of arrows (they connect two symbols) and detects candidates after uniform symbol candidates have been recognized. We consider each pair of detected uniform symbol candidates and try to find an arrow as an arbitrarily shaped connector linking them. Each arrow consists of a shaft and a head; hence, the arrow candidates detector works in two sequential steps:

1. Find the shaft of an arrow connecting two given symbols. The shaft is a sequence of strokes leading from the vicinity of the first symbol to the vicinity of the second symbol and is undirected.
2. Find the head located around one of the end points of the shaft. It defines the arrow orientation.

The shaft detection is done by adding strokes iteratively to a sequence such that the first stroke starts in the vicinity of the first symbol and the last stroke ends in the vicinity of the second symbol. Once the shaft is found, the head is detected by classification of strokes around both ends of the shaft into two classes: *head* and *not head*. This classification is based on relative positioning of strokes. Detected arrows are assigned a score given by the quality of the shaft and the head. More details can be found in our original paper [6].

To test the arrow detector, we took all annotated uniform symbols and tried to find arrows connecting them. Detected arrows were compared with the annotated arrows. Note that text strokes were removed by our text/non-text separator and all pairs of symbols were considered. The result of the arrow detector is a list of arrow candidates since it may detect several conflicting arrows between each pair of symbols. These conflicts are intended to be solved by the structural analyser. We achieved a recall/precision of 92.4/63.1, 94.7/75.0 and 95.1/44.6 % for FC_A, FC_B and FA, respectively. Our arrow detector performs 88 stroke classifications on average per diagram when searching for arrow heads when there are ten arrows on average per diagram.

4.4 Structural analysis

The input to the structural analysis comprises symbol candidates assigned by score. Candidates for arrows also identify which two symbols they connect. The task is to detect a subset of the candidates forming a valid diagram. The score of the individual candidates itself does not suffice (even a bad candidate might have a high score); relations between the candidates have to be examined. Each relation is assigned its own score. The score of the entire diagram is calculated as the sum of scores of all selected symbol candidates and relations among them. The highest score solution is sought in an optimization task. Having the candidates selected, the diagram structure can be easily reconstructed—all the necessary information is carried by arrows.

We define three types of relations between symbol candidates: (1) conflict—two candidates share one or more strokes or two arrows are connected to the same connection points of uniform symbols (this forbids parallel arrows in flowcharts); (2) overlap—two uniform symbol candidates have overlapping bounding boxes; (3) endpoint—each arrow requires existence of both uniform symbols it connects. All possible pairs of candidates are examined to find first two relations. Potential conflicts are detected first, and if none occurs, relations of type (2) are evaluated. Relations of type (3) are explicitly given by the definition of arrows. Relations of type (1) get score $s_c = -\infty$. Each relation of type (2) gets the score

$$s_o = -S_{A\cap B}/\min(S_A, S_B), \tag{2}$$

where A and B are bounding boxes of the first and the second symbol. S_A, S_B and $S_{A\cap B}$ are areas of A, B and of their intersection. Finally, relations of type (3) get score $s_e = -\infty$. The first two relations are effective if both symbol candidates are selected in the solution; the third one is effective when the arrow is selected and one of the connected uniform symbols is not. Negative scores of the relations express that they are unwanted.

4.4.1 Max-sum formulation

The pairwise max-sum labelling problem [36] (a.k.a. computing the MAP configuration of a Markov random field) is defined as maximizing the sum of unary and binary cost functions (potentials of discrete variables)

$$\max_{\boldsymbol{k}\in K^V}\left[\sum_{u\in V} g_u(k_u) + \sum_{\{u,v\}\in E} g_{uv}(k_u, k_v)\right], \tag{3}$$

where an undirected graph $G = (V, E)$, a finite set of labels K and costs $g_u(k_u)$, $g_{uv}(k_u, k_v) \in \mathbb{R} \cup \{-\infty\}$ are given. We maximize over assignments of labels from K to nodes of G. Each node u and edge $\{u, v\}$ are then evaluated by the cost given by g_u and g_{uv}.

In our model, each symbol candidate defines a single node of the graph G. The edge is defined for each pair of interacting nodes (i.e. two symbol candidates in a relation). Two labels are used, $K = \{0, 1\}$, where 0 means the candidate is not selected as a part of the solution while 1 means it is. Values $g_u(k_u)$, $g_{uv}(k_u, k_v)$ are set to express scores of symbol candidates and relations and to model natural restrictions as follows: $g_u(0) = 0$ and $g_u(1) = s$ for each symbol candidate u with the score s. Further, for all pairs of objects $\{u, v\} \in E$

1. $g_{uv}(1, 1) = s_c = -\infty$ if u and v are in conflict.
2. $g_{uv}(0, 1) = s_e = -\infty$ if u is a symbol candidate and v is an arrow connected to that symbol.
3. $g_{uv}(1, 1) = s_o$ if u, v are two non-arrow symbol candidates with overlapping boxes (s_o is given by (2)).
4. $g_{uv}(k, \ell) = 0$ otherwise.

A good commensurability of various scores in the model is confirmed by experiments. We also tested a set-up of the unary and binary potentials based on logarithms of scores, which did not lead to better results.

4.4.2 Example

We illustrate the structural analysis by a simple example. Figure 6a contains an input flowchart with labelled strokes. We assume that the following symbol candidates were detected in this diagram:

1: process $\{t_1\}$ with score s_1,
2: connection $\{t_4\}$ with score s_2,
3: connection $\{t_8\}$ with score s_3,
4: terminator $\{t_8\}$ with score s_4,
5: arrow $\{t_2, t_3\}$ $[1 \rightarrow 2]$ with score s_5,
6: arrow $\{t_5, t_6, t_7\}$ $[1 \rightarrow 3]$ with score s_6,
7: arrow $\{t_5, t_6, t_7\}$ $[1 \rightarrow 4]$ with score s_7.

The strokes forming each symbol candidate are in curly brackets. For arrow candidates, the values in the square brackets say which symbols are connected by the arrow. The resulting max-sum model is depicted in Fig. 6b.

4.4.3 Solving the optimization task

The max-sum problem is NP hard, although some of its special forms, such as the submodular max-sum problem, can be solved in a polynomial time [36]. Unfortunately, this is not our case. However, the size of generated graphs is not

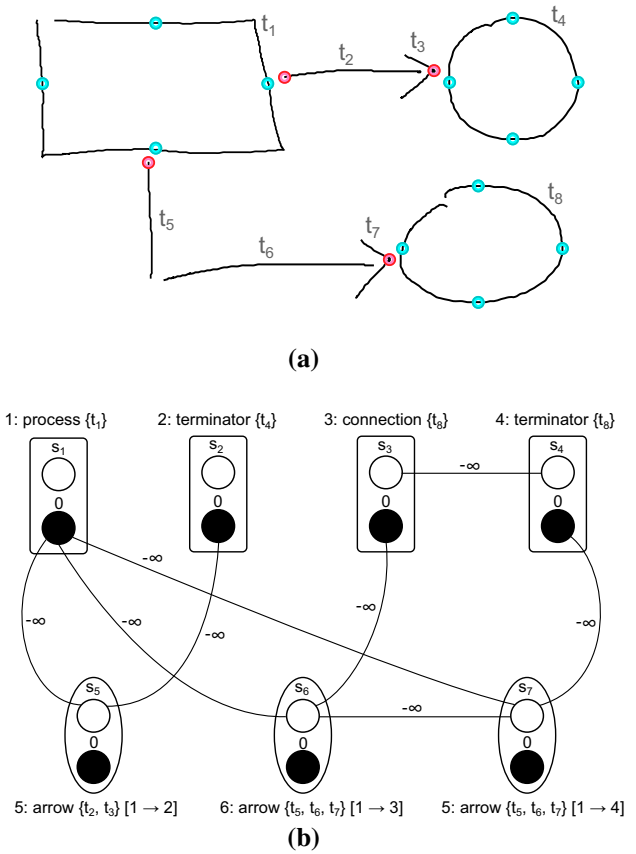


Fig. 6 Input flowchart (**a**) consisting of eight strokes $t_1, \ldots, t_8$ forming three uniform *symbols* and two *arrows*. Connection points are marked in *blue* and *red* for uniform *symbols* and *arrows*, respectively. There are four uniform *symbol* candidates since stroke t_8 is classified as *connection* (with score s_3) and *terminator* (with score s_4). The structural analysis is cast as a max-sum problem **b** where *rectangular* nodes represent uniform *symbol* candidates while elliptic nodes represent *arrow* candidates (formally, the nodes are of the same kind). Both possibilities for labelling a particular node are represented inside the node (*white circle* label 1, *black circle* label 0). Each label k in a node u is assigned by value $g_u(k)$. An edge connecting a label k in a node u and a label ℓ in a node v is assigned by value $g_{uv}(k, \ell)$. Edges with zero costs are not shown (colour figure online)

so big (72/50/84 nodes and 673/305/721 edges in average for FC_A/FC_B/FA). Therefore, general branch and bound solvers are able to solve them fast (see Sect. 5). We tested the max-sum solver Toulbar2.[5] Its minor disadvantage is that it supports only non-negative integer costs, and thus, it is necessary to transform the score values. Another option is to formulate the max-sum problem as an integer linear program (ILP) and solve it using a general ILP solver. We tested IBM ILOG CPLEX library.[6] The conversion is based on linear programming relaxation of the problem [36].

[5] http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/ToolBarIntro.

[6] http://www.ibm.com/software/integration/optimization/cplex-optimizer/.

We made experiments with both formulations and compared the performance of Toulbar2 and CPLEX. Both solvers find the exact solution, and the recognition result is the same. Toulbar2 is approx. 60 % faster on average. It appears slower for the smallest diagrams; this is caused by the data exchange trough files, which is the only possibility supported by the binary distribution of Toulbar2. Some minimal time is thus needed for the solver initialization. This minor technical limitation may be resolved in future. Details on runtimes are provided in Sect. 5.

5 Experiments

We present an overall performance evaluation of the proposed diagram recognition system (abbreviated γ). We made a comparison with two state-of-the-art methods: the grammar-based method (α) by Carton et al. [10] and the purely statistical method (β) by Delaye [11], and compared our performance to their published results. The former was evaluated on the FC_A database only, while the second was evaluated on the FA database as well. FC_B is new (introduced in this work), and results achieved on this database are thus reported without a comparison to others. However, it shows that the higher quality data match well to capabilities of current devices. A significantly higher accuracy is achieved.

We use two basic metrics to measure the recognition precision. The first (SL) assesses the correct stroke labelling. We assign each stroke a label of the symbol class it is classified to, and this assignment is checked against the ground truth in the database. The second metric (SR1) assesses the correct symbol segmentation and classification. It is more informative and provides a better insight into the recognition result quality. The most direct way to decide whether a symbol was correctly recognized is to check whether it comprises exactly the same strokes and has the same label as the annotation. We call this criterion the strict one. The metrics SL and SR1 are common and were used by authors of both systems α and β, and thus, they allow fair comparison of individual systems. However, exact stroke matching is not necessarily required for the correct diagram structure recognition. Users sometimes draw symbols by multiple strokes when correcting or when beautifying symbols. Some strokes are redundant in some way. It might happen that although the symbol is recognized correctly, it is not formed of exactly the same strokes as its annotated pattern in the database. For more insight, we define an additional more relaxed criterion (SR2) based on matching each annotated symbol with one of the recognized symbols. Two symbols match if they are of the same class and their bounding boxes overlap by 80 %. This value was estimated empirically. It must be high enough to forbid matching of different symbols. Contrary, it must be low enough to allow matching of symbols with shrunk bounding boxes due to missing redundant strokes. In the case of arrows, we also require that they connect the same symbols and have the same direction. The error rate is expressed as the number of unmatched annotated symbols. This criterion is more meaningful. It was used to assemble the histogram in Fig. 7 showing how many diagrams were recognized with a particular number of errors. The best results were achieved for the FA database, where nearly 80 % of diagrams were recognized correctly. The worst results were achieved for the FC_A database, probably because of its low quality. Inputs are noisy, and there is no temporal information; thus, it has not been possible to synthesize additional samples to train our classifiers. Even so, we have still achieved state-of-the-art precision, see details in Tables 1, 2 and 3. Differences in symbol segmentation and recognition results given by the two criteria SR1 and SR2 are shown in Table 4.

The system has been implemented in C#, and tests were performed on a standard tablet PC Lenovo X230 (Intel Core i5 2.6 GHz, 8 GB RAM) with 64-bit Windows 7. The average runtime needed for recognition was 0.78, 0.89 and 0.69 s

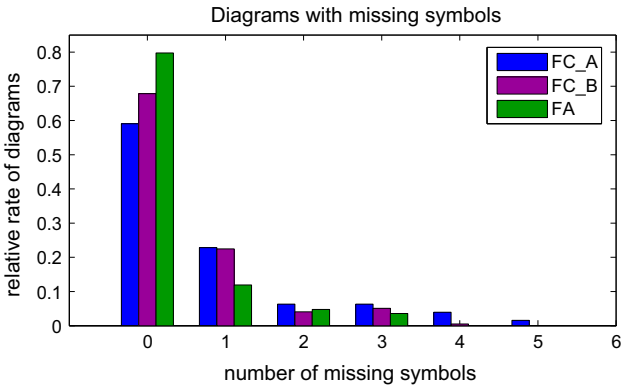


Fig. 7 Counts of diagrams by the number of missing symbols in the result

Table 1 Recognition results for FC_A database

Class	SL (%)			SR1 (%)		
	(α)	(β)	(γ)	(α)	(β)	(γ)
Arrow	83.8	–	87.5	70.2	–	76.6
Connection	80.3	–	94.1	82.4	–	95.1
Data	84.3	–	95.3	80.5	–	90.5
Decision	90.9	–	88.2	80.6	–	72.9
Process	90.4	–	96.3	85.2	–	88.6
Terminator	69.8	–	90.7	72.4	–	89.0
Text	97.2	–	99.2	74.1	–	89.7
Total	92.4	93.2	96.3	75.0	75.5	84.2

Comparison of the proposed recognizer (γ) to the grammar-based method (α) and to the purely statistical method (β). We list correct stroke labelling (SL) and symbol segmentation and recognition measured with the strict (SR1) method

Table 2 Results for FC_B database

Class	SL (%)	SR1 (%)
Arrow	93.8	93.2
Connection	88.4	88.4
Data	96.1	93.8
Decision	90.3	92.0
Process	98.4	97.6
Terminator	99.7	98.9
Text	99.6	97.1
Total	98.4	95.3

We list correct stroke labelling (SL) and symbol segmentation and recognition measured with strict (SR1) method

Table 3 Recognition results for FA database

Class	SL (%)		SR1 (%)	
	(β)	(γ)	(β)	(γ)
Arrow	–	98.0	–	97.5
Initial arrow	–	98.6	–	97.3
Final state	–	99.2	–	99.2
State	–	98.3	–	98.2
Label	–	99.7	–	99.2
Total	98.4	99.0	97.1	98.5

We compared the proposed system (γ) with the purely statistical method by Delaye (β). We list correct stroke labelling (SL) and symbol segmentation and recognition measured with strict (SR1) method

Table 4 Comparison of the symbol segmentation and recognition rates using strict (SR1) and structure-based (SR2) method

Class	SR1 (%)/SR2 (%)		
	FC_A	FC_B	FA
Arrow	76.6/78.3	93.2/94.3	97.5/98.1
Connection	95.1/96.0	88.4/89.3	–
Data	90.5/91.4	93.8/94.7	–
Decision	72.9/76.1	92.0/95.1	–
Process	88.6/89.9	97.6/98.4	–
Terminator	89.0/89.7	98.9/99.6	–
Text/label	89.5/91.6	97.1/98.7	99.2/99.4
Initial arrow	–	–	97.3/97.3
Final state	–	–	98.2/98.6
State	–	–	99.2/99.2
Total	84.2/85.4	95.3/96.6	98.5/98.8

for diagrams from FC_A, FC_B and FA, respectively. This means that our system is faster than the grammar-based system by Carton et al., which has an average recognition time 1.94 s and slower than the purely statistical approach by Delaye and Lee with an average recognition time 80 and 52 ms for FC_A and FA, respectively. Table 5 lists the minimal, maximal, average and median time needed to solve the max-sum problem and to perform the entire recognition. The values confirm that the optimization is solved relatively fast as it consumes only a small proportion of the whole processing time. The speed of the solver Toulbar2 is compared with CPLEX in Table 6.

Table 5 Optimization/total running time

Dtb.	Running time (s)			
	Minimal	Maximal	Average	Median
FC_A	0.11/0.19	0.66/4.61	0.14/0.78	0.12/0.71
FC_B	0.11/0.46	0.22/3.56	0.13/0.89	0.12/0.83
FA	0.12/0.27	0.37/1.43	0.14/0.69	0.13/0.62

Table 6 Running time consumed to solve the optimization by Toulbar2/CPLEX

Database	Running time (ms)			
	Minimal	Maximal	Average	Median
FC_A	114/2	655/917	136/230	123/218
FC_B	114/3	216/598	126/129	122/48
FA	116/19	370/720	137/235	129/229

6 System analysis

Here, we report on additional experiments performed to analyse the impact of the individual steps of the pipeline on the overall precision. We investigated which steps of the recognition pipeline are responsible for misrecognition of symbols from individual classes. Some of the recognition failures are illustrated by examples and commented. We also tested the system having the advanced pipeline steps disabled one by one.

6.1 Failure analysis

The system fails to recognize some diagrams even when the best approaches are used in each step of the pipeline. There are four possible reasons why a symbol may not be recognized properly: (a) some of its strokes were misclassified as text; (b) the symbol was not properly segmented; (c) the symbol was rejected by the classifier; (d) the structural analyser did not choose the symbol. The special case is an arrow not segmented as a uniform symbol. We say that wrong segmentation is the reason for its misrecognition if the symbols it connects were not segmented correctly. Another exception is a text block, misrecognition of which is always caused by the misrecognition of a symbol it labels, and thus, it is not a part

Table 7 Numbers of misrecognized symbols from the individual symbol classes with rates of the reasons for their misrecognition (dominant shown bolded)

Class	# of misrecognized symbols (–)	Text separation (%)	Segmentation (%)	Classification (%)	Structural analysis (%)
Arrow	220/91/18	0.3/7.7/0.0	**58.5**/**51.7**/5.6	35.2/37.4/33.3	6.0/3.3/**61.1**
Connection	5/13/–	0.0/0.0/–	17.7/0.0/–	**82.3**/38.5/–	0.0/**61.5**/–
Data	20/22/–	2.3/4.5/–	**79.1**/13.6/–	7.0/31.8/–	11.6/**50.0**/–
Decision	35/18/–	0.0/5.5/–	**75.0**/**77.7**/–	4.2/16.7/–	20.8/0.0/–
Process	35/9/–	2.7/11.1/–	**67.6**/33.3/–	13.5/**55.6**/–	16.2/0.0/–
Terminator	17/3/–	5.9/0.0/–	**76.5**/0.0/–	5.9/0.0/–	11.8/**100.0**/–
Initial arrow	–/–/2	–/–/0.0	–/–/0.0	–/–/**100.0**	–/–/0.0
Final state	–/–/1	–/–/0.0	–/–/0.0	–/–/0.0	–/–/**100.0**
State	–/–/5	–/–/0.0	–/–/20.0	–/–/0.0	–/–/**80.0**

The displayed result correspond to FC_A/FC_B/FA databases

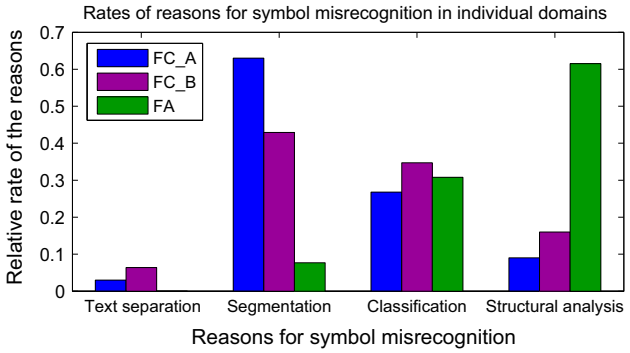


Fig. 8 Rates of reasons for symbol misrecognition

of the analysis. The rate of each reason for symbol misrecognition with respect to the symbol class is shown in Table 7. The average rates are shown for individual datasets in Fig. 8. It has turned out that the most frequent reason for failure is the symbol misclassification in the case of flowcharts and wrong selection of symbol candidates by the structural analyser in the case of finite automata. Diagrams with the highest number of misclassified symbols are analysed in Fig. 9.

6.2 Advanced techniques analysis

We replaced the most advanced techniques for text/non-text separation, symbol segmentation and symbol classification by our previous or naive approaches. The results are summarized in Table 8.

Our recognition system is robust enough to handle some remaining text in a diagram. On the other hand, it can barely recover when some shape strokes are removed. However, if there is a lot of remaining text, the system needs significantly more time for recognition and can eventually get confused. To demonstrate how the system is susceptible to the result of text/non-text separation, we evaluated it with three different text/non-text separation settings: (a) using unbiased text/non-text classifier, (b) using the perfect text removal based on the annotation, (c) performing no text/non-text separation.

We used strokes grouping instead of the clustering to perform over-segmentation. The iterative strokes grouping is a naive method achieving high recall values at the cost of lower precision. SLAC is a more sophisticated method with significantly increased precision and only slightly worse recall which guarantees a speed up of the system.

We evaluated the system with uniform symbols classifiers trained without artificial samples. These classifiers achieve a lower precision reflected in the lower accuracy of the whole system. Moreover, it has reduced ability to reject clusters which represent no symbols, and thus, the recognition time slightly increases. It is obvious that the importance of classifiers trained with artificial samples increases in the flowchart domain with a higher number of symbol classes. Unfortunately, the FC_A database does not contain time information which is necessary for synthesis of the artificial samples; thus, we could not do this analysis on the database.

6.3 Analysis findings

Based on the performed analysis, we come to the following conclusions:

The *text/non-text separation* step is very precise. It achieved 100 % precision in the shapes class on the FA database, and thus, it was not responsible for a single error there. Even in the case of the flowcharts, it was responsible for symbol misrecognition only in a few cases. Further analysis showed the importance of the text/non-text separation step. Without text being separated, the time complexity increased and the precision dropped significantly. On the other hand, it turned out that the results achieved with unbiased classifiers or ground truth-based separation are comparable to the baseline. Naturally, the use of ground truth led to faster recognition, because all text strokes were removed.

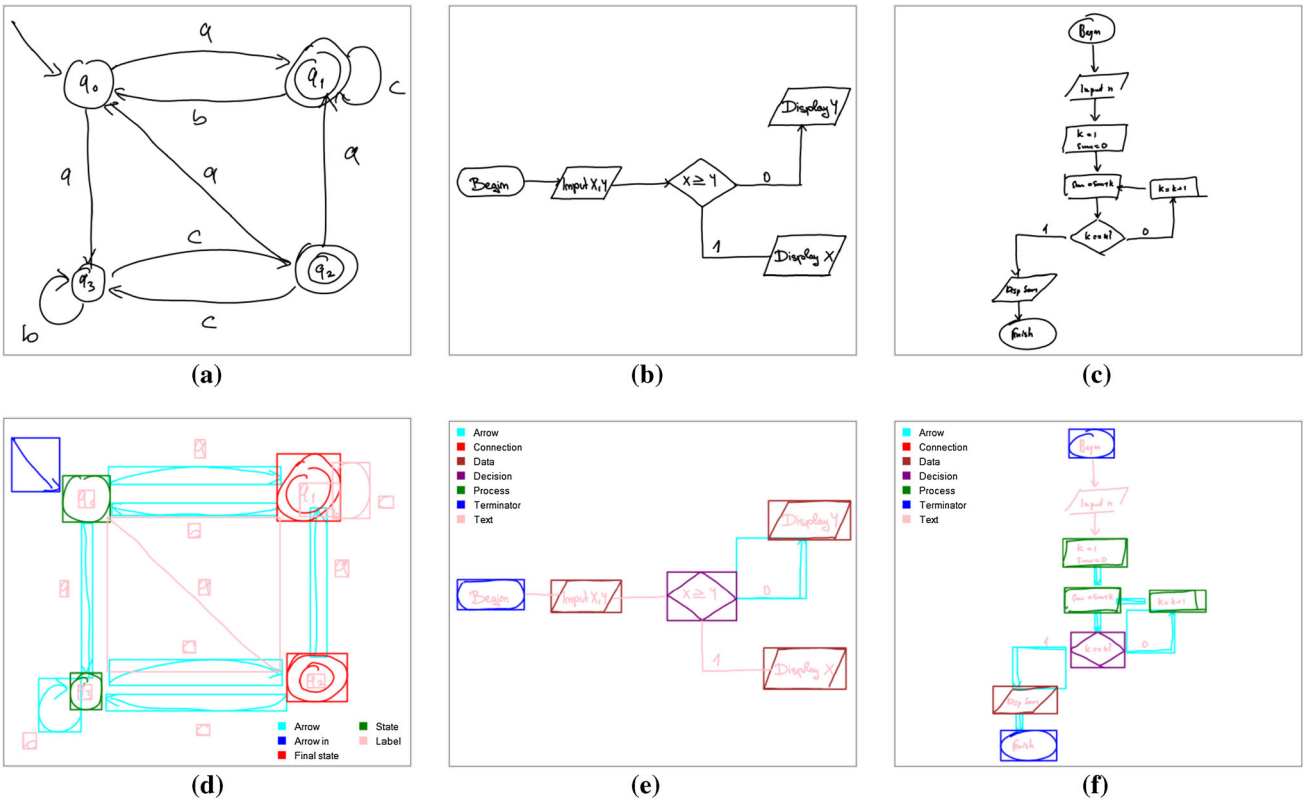


Fig. 9 Examples of misrecognized diagrams from the FA and FC_B databases. The input diagrams are shown in the *upper row*; recognition results are in the *bottom row*. *Symbol colouring* is explained in the legends which are parts of the images. Each recognized *symbol* is surrounded by its *bounding box*. **a** Input diagram writer020_fa_002, **b** input diagram writer018_fc_003b, **c** input diagram writer019_fc_006b, **d** misrecog. *arrows*—colliding heads, **e** misrecog. *arrows*—a missing head and **f** unrecognized data—interstroke gaps (colour figure online)

Table 8 Results of various analyses showing the effect of individual solutions to the steps of the pipeline

	Analysis					
	Baseline	Text separation			Symbol segmentation	Symbol classification
		Unbiased	Ground truth	No separation	Grouping	No artificial samples
FA						
SL (%)	99.0	99.0	99.3	83.7	98.8	98.7
SR1 (%)	98.5	98.5	98.9	82.0	98.2	98.1
SR2 (%)	98.8	98.8	99.2	82.7	98.7	98.5
AT (ms)	690	704	650	962	1220	705
FC_A						
SL (%)	96.3	96.2	96.6	94.1	96.5	–
SR1 (%)	84.2	84.1	84.6	79.4	84.4	–
SR2 (%)	86.4	86.2	86.6	82.7	86.7	–
AT (ms)	780	770	762	2273	1060	–
FC_B						
SL (%)	98.5	98.5	98.7	96.5	98.6	94.3
SR1 (%)	95.6	95.5	96.0	90.5	95.2	87.7
SR2 (%)	97.1	96.9	97.3	93.2	97.8	89.2
AT (ms)	891	892	815	3429	1205	930

It is compared to the baseline which was used to obtain the results in Sect. 5. We measured correct rate of stroke labelling (SL), symbol segmentation and recognition measured with strict (SR1) and structure-based (SR2) method, and average recognition time (AT)

Symbol segmentation is the main culprit in symbol misrecognition for both flowchart databases. This is caused by the fact that symbols consist of more strokes and users sometimes retrace them. Moreover, when a symbol is not segmented correctly, it inherently causes misrecognition of arrows connected to it. Further analysis showed that the naive strokes grouping can increase the precision. However, it cannot compensate the increase of processing time.

Symbol classification is the second most responsible step for symbol misrecognition. Its failure means that the classifier rejected a symbol candidate. The use of artificial samples to train the classifiers is more important in the case of the flowchart domain, due to the higher number of symbol classes and the fact that some of them might be of a very similar appearance.

Structural analysis is the last step of the recognition pipeline, in which the symbols are selected from the symbol candidates. An error occurs in this step when the classifier does not reject a symbol, but gives more alternatives for classification, and the structural analyser picks the wrong one. It typically happens in the case of two similarly looking symbols like state/final state or connection/terminator.

7 Conclusion

We have designed a recognizer that understands diagrams such as flowcharts or finite automata. We studied which methods are optimal for the implementation of individual pipeline steps. Some existing algorithms were suitable for this purpose and were thus integrated. Beyond these, several new approaches were introduced. The system performed well in comparison with the state of the art. We encourage the reader to experiment with our demo application available at http://cmp.felk.cvut.cz/~breslmar/diagram_recognizer/.

The low quality of the benchmark flowchart database motivated us to gather our own database. The main weaknesses of the FC_A database are the lack of temporal information and incomplete annotation—our database contains temporal information as well as information about pressure which may be useful for development of new methods. It provides complex annotations, including meaning of the text and the annotation of arrow heads. This higher quality database better reflects current standards in ink input interfaces. Experiments proved that high-quality data yield more accurate results. As another contribution, we make this database available for the community.

In future work, we would like to adapt the proposed method for other diagrammatic domains, and we are planning to cooperate with diagram users professionally. We are interested especially in the usage of diagrams during a creative process of sharing fresh ideas among cooperating people. This feedback should identify the most important domains and use cases. Next, we would like to experiment with iterative recognition using immediate feedback to the user of intermediate results. This may well reduce recognition time, and we believe that the proposed system is capable of such adaptation. Another possibility is to extend it to support domains beyond the scope of arrow-connected diagrams. This extension would require modification of the max-sum model. It is possible if the fundamental relation between arrows and symbols can be replaced by another relation defining the structure of the handwriting. For examples, the relation between notes and staff lines in the case of music scores.

Acknowledgments The first author was supported by the Grant Agency of the CTU under the project SGS16/085/OHK3/1T/13. The second and the third authors were supported by the Czech Science Foundation under grant no. 15-04960S. The authors thank Truyen van Phan for his help with the text/non-text separation, Daniel Martín-Albo for creating the synthesized samples for the SVM classifiers and Roger Boyle for proofreading of the paper.

References

1. Alvarado, C., Davis, R.: SketchREAD: a multi-domain sketch recognition engine. In: UIST '04: 17th Annual ACM Symposium on User Interface Software and Technology. UIST '04, pp. 23–32. ACM, New York (2004)
2. Álvaro, F., Sánchez, J.A., Benedí, J.M.: Recognition of on-line handwritten mathematical expressions using 2d stochastic context-free grammars and hidden Markov models. Pattern Recogn. Lett. **35**, 58–67 (2014)
3. Arvo, J., Novins, K.: Appearance-preserving manipulation of hand-drawn graphs. In: 3rd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia, GRAPHITE '05, pp. 61–68. ACM (2005)
4. Awal, A.M., Feng, G., Mouchere, H., Viard-Gaudin, C.: First experiments on a new online handwritten flowchart database. In: DRR'11, pp. 1–10 (2011)
5. Blagojevic, R., Plimmer, B., Grundy, J., Wang, Y.: Using data mining for digital ink recognition: dividing text and shapes in sketched diagrams. Comput. Graph. **35**(5), 976–991 (2011)
6. Bresler, M., Průša, D., Hlaváč, V.: Detection of arrows in on-line sketched diagrams using relative stroke positioning. In: WACV '15: IEEE Winter Conference on Applications of Computer Vision, pp. 610–617. IEEE Computer Society (2015)
7. Bresler, M., Průša, D., Hlaváč, V.: modeling flowchart structure recognition as a max-sum problem. In: O'Conner, L. (ed.) ICDAR '13: 12th International Conference on Document Analysis and Recognition, pp. 1247–1251. IEEE Computer Society (2013)
8. Bresler, M., Průša, D., Hlaváč, V.: Using agglomerative clustering of strokes to perform symbols over-segmentation within a diagram recognition system. In: Paul Wohlhart, V.L. (ed.) CVWW '15: Proceedings of the 20th Computer Vision Winter Workshop, pp. 67–74. Graz University of Technology (2015)
9. Bresler, M., Van Phan, T., Průša, D., Nakagawa, M., Hlaváč, V.: Recognition system for on-line sketched diagrams. In: Guerrero, J.E. (ed.) ICFHR '14: 14th International Conference on Frontiers in Handwriting Recognition, pp. 563–568. IEEE Computer Society (2014)
10. Carton, C., Lemaitre, A., Couasnon, B.: Fusion of statistical and structural information for flowchart recognition. In: ICDAR '13:

12th International Conference on Document Analysis and Recognition, pp. 1210–1214 (2013)
11. Delaye, A.: Structured prediction models for online sketch recognition (2014). Unpublished manuscript. https://sites.google.com/site/adriendelaye/home/news/unpublishedmanuscriptavailable
12. Delaye, A., Anquetil, E.: HBF49 feature set: a first unified baseline for online symbol recognition. Pattern Recogn. **46**(1), 117–130 (2013)
13. Delaye, A., Lee, K.: A flexible framework for online document segmentation by pairwise stroke distance learning. Pattern Recogn. **48**(4), 1197–1210 (2015)
14. Delaye, A., Liu, C.L.: Contextual text/non-text stroke classification in online handwritten notes with conditional random fields. Pattern Recogn. **47**(3), 959–968 (2014)
15. Feng, G., Viard-Gaudin, C., Sun, Z.: On-line hand-drawn electric circuit diagram recognition using 2D dynamic programming. Pattern Recogn. **42**(12), 3215–3223 (2009)
16. Hammond, T., Davis, R.: LADDER, a sketching language for user interface developers. Comput. Graph. **29**, 518–532 (2005)
17. Hammond, T., Davis, R.: Tahuti: A geometrical sketch recognition system for UML class diagrams. In: ACM SIGGRAPH 2006 Courses, SIGGRAPH '06. ACM, New York (2006)
18. Hammond, T., Paulson, B.: Recognizing sketched multistroke primitives. ACM Trans. Interact. Intell. Syst. **1**(1), 4:1–4:34 (2011)
19. Indermühle, E., Frinken, V., Bunke, H.: Mode detection in online handwritten documents using BLSTM neural networks. In: ICFHR '12: 13th International Conference on Frontiers in Handwriting Recognition, pp. 302–307 (2012)
20. Kara, L.B., Stahovich, T.F.: Hierarchical parsing and recognition of hand-sketched diagrams. In: 17th Annual ACM Symposium on User Interface Software and Technology, UIST '04, pp. 13–22. ACM (2004)
21. Le, A.D., Van Phan, T., Nakagawa, M.: A system for recognizing online handwritten mathematical expressions and improvement of structure analysis. In: DAS '14: 11th IAPR International Workshop on Document Analysis Systems, pp. 51–55 (2014)
22. Lemaitre, A., Mouchére, H., Camillerapp, J., Coüasnon, B.: Interest of syntactic knowledge for on-line flowchart recognition. In: GREC '11: 9th IAPR International Workshop on Graphics Recognition, pp. 85–88 (2011)
23. Liu, C.L., Zhou, X.D.: Online Japanese character recognition using trajectory-based normalization and direction feature extraction. In: Lorette, G. (ed.) Tenth International Workshop on Frontiers in Handwriting Recognition. Université de Rennes 1, Suvisoft (2006)
24. Martín-Albo, D., Plamondon, R., Vidal, E.: Training of on-line handwriting text recognizers with synthetic text generated using the kinematic theory of rapid human movements. In: Guerrero, J.E. (ed.) ICFHR '14: 14th International Conference on Frontiers in Handwriting Recognition, pp. 543–548. IEEE Computer Society (2014)
25. Miyao, H., Maruyama, R.: On-line handwritten flowchart recognition, beautification and editing system. In: ICFHR '12: 13th International Conference on Frontiers in Handwriting Recognition, pp. 83–88 (2012)
26. Mouchère, H., Viard-Gaudin, C., Zanibbi, R., Garain, U.: ICFHR 2014 competition on recognition of on-line handwritten mathematical expressions (CROHME 2014). In: J.E. Guerrero (ed.) ICFHR '14: 14th International Conference on Frontiers in Handwriting Recognition, pp. 791–796. IEEE Computer Society (2014)
27. Otte, S., Krechel, D., Liwicki, M., Dengel, A.: Local feature based online mode detection with recurrent neural networks. In: ICFHR '12: 13th International Conference on Frontiers in Handwriting Recognition, pp. 531–535 (2012)
28. Ouyang, T.Y., Davis, R.: Chemink: A natural real-time recognition system for chemical drawings. In: 16th International Conference on Intelligent User Interfaces, IUI '11, pp. 267–276. ACM (2011)
29. Plimmer, B., Purchase, H.C., Yang, H.Y.: Sketchnode: intelligent sketching support and formal diagramming. In: 22nd Conference of the Computer-Human Interaction Special Interest Group of Australia on Computer-Human Interaction, OZCHI '10, pp. 136–143. ACM (2010)
30. Qi, Y., Szummer, M., Minka, T.P.: Diagram structure recognition by Bayesian conditional random fields. In: Conference on Computer Vision and Pattern Recognition, pp. 191–196. IEEE Computer Society (2005)
31. Refaat, K., Helmy, W., Ali, A., AbdelGhany, M., Atiya, A.: A new approach for context-independent handwritten offline diagram recognition using support vector machines. In: IJCNN '08: IEEE International Joint Conference on Neural Networks, pp. 177–182 (2008)
32. Sezgin, T.M., Davis, R.: HMM-based efficient sketch recognition. In: IUI '05: 10th International Conference on Intelligent User Interfaces. IUI '05, pp. 281–283. ACM, New York (2005)
33. Stoffel, A., Tapia, E., Rojas, R.: Recognition of on-line handwritten commutative diagrams. In: ICDAR '09: 10th International Conference on Document Analysis and Recognition, pp. 1211–1215 (2009)
34. Szwoch, W., Mucha, M.: Recognition of Hand Drawn Flowcharts, Advances in Intelligent Systems and Computing, vol. 184. Springer, Berlin (2013)
35. Van Phan, T., Nakagawa, M.: Text/non-text classification in online handwritten documents with recurrent neural networks. In: J.E. Guerrero (ed.) ICFHR '14: 14th International Conference on Frontiers in Handwriting Recognition, pp. 23–28. IEEE Computer Society (2014)
36. Werner, T.: A linear programming approach to max-sum problem: a review. IEEE Trans. Pattern Anal. Mach. Intell. **29**(7), 1165–1179 (2007)

Theoretical Computer Science 679 (2017) 118–125

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Undecidability of the emptiness problem for context-free picture languages ☆

Daniel Průša [a,*], Klaus Reinhardt [b]

[a] Czech Technical University, Faculty of Electrical Engineering, Karlovo náměstí 13, 121 35 Prague 2, Czech Republic
[b] Universität Halle, Institut für Informatik, Von-Seckendorff-Platz 1, 06120 Halle (Saale), Germany

ARTICLE INFO

Article history:
Received 1 December 2015
Received in revised form 22 March 2016
Accepted 24 March 2016
Available online 6 April 2016

Keywords:
Picture languages
Context-free picture grammars
Emptiness problem
Undecidability

ABSTRACT

A two-dimensional Kolam grammar as defined by Siromoney et al. in 1972 and independently by Matz in 1997 and Schlesinger in 1989 allows context-free productions of the form $A \to a$, $A \to B\,C$, $A \to \frac{B}{C}$, and $S \to \lambda$ which concatenate sub-pictures produced by B and C horizontally respectively vertically if their height respectively width fits. We demonstrate that this grammar is quite powerful by proving undecidability of the emptiness problem. We further analyze consequences of this finding and give additional characteristics related to size of generated pictures.

1. Introduction

The theory of *two-dimensional languages* generalizes notions from the theory of formal languages. The basic entity, which is the *string*, is replaced by a rectangular array of symbols, called a *picture*. A motivation for such a generalization comes from the area of image processing, image recognition and two-dimensional pattern matching.

Several models of two-dimensional automata and grammars have been proposed to recognize/generate pictures. The early models of context-free picture grammars include matrix and Kolam type grammars of Siromoney et al. [2,3]. Kolam grammars form a natural extension of the context-free grammars in the Chomsky normal form. This explains why they have attracted the attention of several researchers and were proposed independently more times [4–6]. Beside that, their extensions have been studied. They include the two-dimensional context-free grammars of Průša [7] and regional tile grammars of Pradella et al. [8]. There is also a close relation to the grid grammars of Drewes et al. [9]. And finally, a variant of the regional tile grammars producing hexagonal arrays was introduced by Kamaraj and Thomas [10].

It is a well known phenomenon that the more complex two-dimensional topology changes a lot of properties of accepted/generated languages. For example, the four-way finite automaton of Blum and Hewitt [11], which is a straightforward generalization of the two-way finite automaton, is more powerful with nondeterminism than without it. We focus on another property which is the undecidability of the emptiness problem. It is known that the emptiness is not decidable for four-way automata, even over unary alphabets [12]. Recently, we have studied the emptiness for extensions of the Kolam grammar. We have shown that it is undecidable for a two-dimensional context-free grammar [13], based on more general

☆ Some of the results of this paper have been announced at CIAA 2015 in Umeå, Sweden, August 2015. As a new contribution, here we solve the main problem which remained open in [1].

* Corresponding author.
E-mail addresses: prusapa1@cmp.felk.cvut.cz (D. Průša), klaus.reinhardt@informatik.uni-halle.de (K. Reinhardt).

http://dx.doi.org/10.1016/j.tcs.2016.03.025
0304-3975/

context-free productions, and a three-dimensional Kolam grammar [1]. In this paper, we prove the undecidability of the problem for the two-dimensional Kolam grammar, resolving thus the open problem from [1].

The undecidability result has several consequences. For example, there is no analogy to the pumping lemma known from the one-dimensional setting [14]. It is possible to construct a sequence of grammars generating one-picture languages where the picture size grows faster than any recursive function of the grammar size. On the other hand, we show that the number of rows and columns of such pictures cannot be in an arbitrary relation since a sort of pumping lemma holds for wide and high pictures. We further inspect functions representable by the Kolam grammar and show results which coincide with those known for functions representable by tiling systems which define the family of recognizable picture languages (REC) [15].

The text of the paper is organized as follows. The basic notions and notations on picture languages and a definition of Kolam context-free grammars are introduced in Section 2. Results on the emptiness problem are presented in Section 3, results on properties of picture languages generated by Kolam grammars are presented in Section 4. Finally, a concluding summary and discussion is given in Section 5.

2. Context-free picture grammars

We use the common notation and terms on pictures and picture languages (see, e.g., [15]). If Σ is a finite alphabet, then $\Sigma^{*,*}$ is used to denote the set of all rectangular pictures over Σ, that is, if $P \in \Sigma^{*,*}$, then P is a two-dimensional array of symbols from Σ. If P has m rows and n columns, we say it is of size $m \times n$, and we write $P \in \Sigma^{m,n}$, $\mathrm{rows}(P) = m$ and $\mathrm{cols}(P) = n$. If P is a square picture of size $n \times n$, we shortly say P is of size n. We also write $a^{m,n}$ to denote the picture over $\{a\}$ of size $m \times n$. The *empty picture* λ is in $\Sigma^{m,0}$ and in $\Sigma^{0,n}$ for all $m, n \in \mathbb{N}$. Moreover, $\Sigma^{+,+}$ is the set of non-empty pictures, i.e., $\Sigma^{+,+} = \Sigma^{*,*} \setminus \{\lambda\}$. Each $a \in \Sigma$ is also treated as a picture of size 1×1.

Two (partial) binary operations are introduced to concatenate pictures. Let A be a picture of size $k \times \ell$ such that a_{ij} is the symbol in the i-th row and j-th column. Similarly, let B be a picture of size $m \times n$ with symbols b_{ij}. The *column concatenation* $A \oplus B$ is defined iff $k = m$, and the *row concatenation* $A \ominus B$ is defined iff $\ell = n$. The products are specified by the following schemes:

$$A \oplus B = \begin{bmatrix} a_{11} & \dots & a_{1\ell} & b_{11} & \dots & b_{1n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{k1} & \dots & a_{k\ell} & b_{m1} & \dots & b_{mn} \end{bmatrix} \text{ and } A \ominus B = \begin{bmatrix} a_{11} & \dots & a_{1\ell} \\ \vdots & \ddots & \vdots \\ a_{k1} & \dots & a_{k\ell} \\ b_{11} & \dots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{m1} & \dots & b_{mn} \end{bmatrix}.$$

Beside that, both operations are always defined when at least one of the operands is λ. In this case, λ is the neutral element, so $\lambda \ominus P = P \ominus \lambda = \lambda \oplus P = P \oplus \lambda = P$ for any picture P.

The operations extend to picture languages. For $L_1, L_2 \in \Sigma^{*,*}$, we define

$$L_1 \oplus L_2 = \{P \mid P = P_1 \oplus P_2 \wedge P_1 \in L_1 \wedge P_2 \in L_2\},$$
$$L_1 \ominus L_2 = \{P \mid P = P_1 \ominus P_2 \wedge P_1 \in L_1 \wedge P_2 \in L_2\}.$$

Definition 1. A *two-dimensional Kolam grammar* (2KG) is a tuple $\mathcal{G} = (V_N, V_T, \mathcal{P}, S_0)$, where V_N is a finite set of nonterminals, V_T is a finite set of terminals, $S_0 \in V_N$ is the initial nonterminal and $\mathcal{P}$ is a finite set of productions in one of the following forms:

$$N \to a \quad (1) \qquad S_0 \to \lambda \quad (2)$$
$$N \to A\,B \quad (3) \qquad N \to \begin{matrix} A \\ B \end{matrix} \quad (4)$$

where $N, A, B \in V_N$ and $a \in V_T$.

Definition 2. Let $\mathcal{G} = (V_N, V_T, \mathcal{P}, S_0)$ be a 2KG. For each $N \in V_N$, $L(\mathcal{G}, N)$ is the set of pictures *generated* by $\mathcal{G}$ from N. All these sets are the smallest sets fulfilling the following rules.

1. If $N \to a$ is a production in $\mathcal{P}$ then $a \in L(\mathcal{G}, N)$,
2. if $S_0 \to \lambda$ is in $\mathcal{P}$ then $\lambda \in L(\mathcal{G}, S_0)$,
3. if $N \to A\,B$ is in $\mathcal{P}$, $P = P_1 \oplus P_2$, $P_1 \in L(\mathcal{G}, A)$ and $P_2 \in L(\mathcal{G}, B)$, then $P \in L(\mathcal{G}, N)$, and
4. if $N \to \begin{matrix} A \\ B \end{matrix}$ is in $\mathcal{P}$, $P = P_1 \ominus P_2$, $P_1 \in L(\mathcal{G}, A)$ and $P_2 \in L(\mathcal{G}, B)$, then $P \in L(\mathcal{G}, N)$.

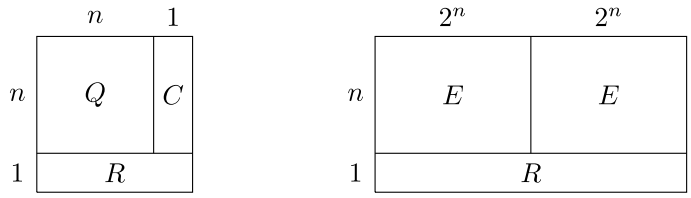


Fig. 1. Schemes showing how pictures $(n+1)\times(n+1)$ and $(n+1)\times 2^{n+1}$ in Examples 1 and 2, respectively, are assembled from smaller parts.

The picture language generated by $\mathcal{G}$ is defined as $L(\mathcal{G}) = L(\mathcal{G}, S_0)$.

Example 1 *(Square pictures).* Let $\mathcal{G} = (V_N, V_T, \mathcal{P}, Q)$ be a 2KG where $V_N = \{R, C, U, Q\}$, $V_T = \{a\}$ and $\mathcal{P}$ is the set of productions

$$R \to a, \qquad R \to R\,R, \qquad C \to a, \qquad C \to \begin{matrix} C \\ C \end{matrix},$$
$$Q \to a, \qquad Q \to \begin{matrix} U \\ R \end{matrix}, \qquad U \to Q\,C\,.$$

Then, $L(\mathcal{G}, R)$ consists of all one-row pictures of a's, $L(\mathcal{G}, C)$ consists of all one-column pictures of a's, $L(\mathcal{G}, U)$ consists of pictures of size $n\times(n+1)$, $n\in\mathbb{N}^+$, and $L(\mathcal{G}, Q) = L(\mathcal{G})$ is the picture language of non-empty square pictures.

Example 2 *(Exponentially sized pictures).* Let $\mathcal{G} = (V_N, V_T, \mathcal{P}, E)$ be a 2KG where $V_N = \{A, R, D, E\}$, $V_T = \{a\}$ and $\mathcal{P}$ is the set of productions

$$R \to a, \qquad R \to R\,R, \qquad A \to a,$$
$$E \to A\,A, \qquad E \to \begin{matrix} D \\ R \end{matrix}, \qquad D \to E\,E\,.$$

Again, $L(\mathcal{G}, R)$ consists of all one-row pictures of a's. The picture languages $L(\mathcal{G}, D)$ and $L(\mathcal{G}, E) = L(\mathcal{G})$ consist of all pictures over $\{a\}$ of size $n\times 2^{n+1}$ and $n\times 2^n$, respectively ($n\in\mathbb{N}^+$). Recursive patterns utilized in the examples are depicted in Fig. 1.

A matrix grammar, the former model of Siromoney et al. [2], can be seen as a special type of Kolam grammar with the usage of productions restricted in the following way. Productions of type (3) generate a row of nonterminals from S_0, then productions of type (4) generate columns of terminals of the same length from the nonterminals. A formal definition follows.

Definition 3. A *two-dimensional matrix grammar* (2MG) is a tuple $\mathcal{G} = (V_1, V_2, V_T, \mathcal{P}, S_0)$ where

- $(V_1 \cup V_2, V_T, \mathcal{P}, S_0)$ is a 2KG,
- $S_0 \in V_1$,
- if $N \to A\,B$ is a production in $\mathcal{P}$ then $N \in V_1$,
- if $N \to \begin{matrix} A \\ B \end{matrix}$ is a production in $\mathcal{P}$ then $N, A, B \in V_2$, and
- if $N \to a$ is a production in $\mathcal{P}$ then $N \in V_2$.

3. Emptiness problem

Kari and Moore proved that the emptiness in undecidable for the four-way automaton working over a unary alphabet [12]. They observed a direct correspondence between this automaton and the 2-counter Minski machine [16]. Indeed, moving the head horizontally or vertically changes horizontal or vertical head coordinate, respectively, by one. And it is easy to check whether the head scans the first row/column.

We show that the undecidable halting problem for a 2-counter Minski machine reduces also to the emptiness problem for a 2KG. The machine is equipped by two counters and a finite-state control unit. Depending on the state, it can either increment one of the counters and change the state or decrement one of the counters and change the state if possible (not zero) and change to a different state otherwise. The machine halts if a final state is reached.

Theorem 1. *Emptiness for* 2KG *is undecidable.*

Proof. Let M be a 2-counter Minski machine with n states and w.l.o.g. one halting state q_n. We will construct a 2KG $\mathcal{G} = (V_N, \{a\}, \mathcal{P}, A_n)$ with

$$V_N = \{A_i, B_i, C_i, D_i, E_i, F_i, G_i \mid i \le n\} \cup \{Q^{j,k}, M^{j,k}, N^{j,k} \mid j,k \le 8\} \cup \{Q, C, R\}$$

such that by induction on the steps of M the following holds:

The counter machine M can reach the configuration with state q_i and the counter values $c, d \in \mathbb{N}$ if and only if there is an $e \in \mathbb{N}$ with the square picture $a^{2^e3^c5^d,2^e3^c5^d} \in L(\mathcal{G}, A_i)$; furthermore $L(\mathcal{G}, A_i)$ consists only of square pictures of this form. (The extra exponent e allows a growing number even if counter values decrease.) Thus $L(\mathcal{G}) = L(\mathcal{G}, A_n)$ is nonempty if and only if M halts.

The set of productions contains all productions from Example 1, hence the nonterminal Q produces squares, R produces rows and C produces columns.

Picture languages $L(\mathcal{G}, Q^{i,j}) = \{a^{in,jn} \mid n \in \mathbb{N}^+\}$ for selected pairs $i, j \le 8$ are generated using the productions

$$Q^{1,2} \to Q\,Q, \qquad Q^{1,3} \to Q^{1,2}\,Q, \qquad Q^{1,4} \to Q^{1,3}\,Q, \qquad Q^{1,5} \to Q^{1,4}\,Q,$$

$$Q^{2,1} \to \begin{matrix} Q \\ Q \end{matrix}, \qquad Q^{3,1} \to \begin{matrix} Q^{2,1} \\ Q \end{matrix}, \qquad Q^{2,3} \to \begin{matrix} Q^{1,3} \\ Q^{1,3} \end{matrix}, \qquad Q^{2,5} \to Q^{2,3}\,Q,$$

$$Q^{3,3} \to \begin{matrix} Q^{1,3} \\ Q^{2,3} \end{matrix}, \qquad Q^{3,4} \to Q^{3,1}\,Q, \qquad Q^{3,8} \to Q^{3,4}\,Q^{3,4},$$

$$Q^{4,5} \to \begin{matrix} Q^{2,5} \\ Q^{2,5} \end{matrix}, \qquad Q^{5,3} \to \begin{matrix} Q^{2,3} \\ Q \end{matrix}, \qquad Q^{5,5} \to \begin{matrix} Q^{1,5} \\ Q^{4,5} \end{matrix}.$$

Square pictures in $L(\mathcal{G}, N^{3,1})$ respectively $L(\mathcal{G}, N^{3,2})$ having size congruent 1 respectively 2 modulo 3 are produced by

$$M^{3,1} \to Q^{3,3}\,C, \qquad N^{3,1} \to \begin{matrix} M^{3,1} \\ R \end{matrix},$$

$$M^{3,2} \to N^{3,1}\,C, \qquad N^{3,2} \to \begin{matrix} M^{3,2} \\ R \end{matrix}.$$

Square pictures in $L(\mathcal{G}, N^{5,m})$ being congruent $0 < m < 5$ modulo 5 are produced by

$$M^{5,1} \to Q^{5,5}\,C, \qquad N^{5,1} \to \begin{matrix} M^{5,1} \\ R \end{matrix},$$

$$M^{5,2} \to N^{5,1}\,C, \qquad N^{5,2} \to \begin{matrix} M^{5,2} \\ R \end{matrix},$$

$$M^{5,3} \to N^{5,2}\,C, \qquad N^{5,3} \to \begin{matrix} M^{5,3} \\ R \end{matrix},$$

$$M^{5,4} \to N^{5,3}\,C, \qquad N^{5,4} \to \begin{matrix} M^{5,4} \\ R \end{matrix}.$$

Now, we are ready to list productions generating representatives of M's configurations. We start the induction by the production $A_0 \to a \in \mathcal{P}$ for the initial state q_0 of M placing $a = a^{2^03^05^0,2^03^05^0} \in L(\mathcal{G}, A_0)$ for the initial configuration with state q_0 and both counters empty.

In the following, we consider the induction step for all six possible types of transitions of M and, at the same time, mention all other productions in $\mathcal{P}$. It does not matter if M is deterministic or nondeterministic.

1. If M increments the first counter going from q_i to q_j then the production $B_i \to A_i Q^{1,2}$ allows $a^{2^e3^c5^d,2^e3^{c+1}5^d} \in L(\mathcal{G}, B_i)$ for $a^{2^e3^c5^d,2^e3^c5^d} \in L(\mathcal{G}, A_i)$. The width is thus multiplied by 3. Then the production $A_j \to \begin{matrix} B_i \\ Q^{2,3} \end{matrix}$ allows $a^{2^e3^{c+1}5^d,2^e3^{c+1}5^d} \in L(\mathcal{G}, A_j)$ which completes the induction step to the following configuration in this case.
2. If M increments the second counter going from q_i to q_j then the production $C_i \to A_i Q^{1,4}$ allows $a^{2^e3^c5^d,2^e3^c5^{d+1}} \in L(\mathcal{G}, C_i)$ for $a^{2^e3^c5^d,2^e3^c5^d} \in L(\mathcal{G}, A_i)$. Here the width is multiplied by 5. Then the production $A_j \to \begin{matrix} C_i \\ Q^{4,5} \end{matrix}$ allows $a^{2^e3^c5^{d+1},2^e3^c5^{d+1}} \in L(\mathcal{G}, A_j)$ analogously.
3. If M decrements the first counter going from q_i to q_j then the production $D_i \to A_i Q^{3,1}$ allows $a^{2^e3^c5^d,2^{e+2}3^{c-1}5^d} \in L(\mathcal{G}, D_i)$ for $a^{2^e3^c5^d,2^e3^c5^d} \in L(\mathcal{G}, A_i)$. Here the width is multiplied by 4/3 (by adding 1/3 of the height), which is possible if and only if the first counter is not zero. Then the production $A_j \to \begin{matrix} D_i \\ Q^{1,4} \end{matrix}$ allows $a^{2^{e+2}3^{c-1}5^d,2^{e+2}3^{c-1}5^d} \in L(\mathcal{G}, A_j)$.
4. If M decrements the second counter going from q_i to q_j then the production $E_i \to A_i Q^{5,3}$ allows $a^{2^e3^c5^d,2^{e+3}3^c5^{d-1}} \in L(\mathcal{G}, E_i)$ for $a^{2^e3^c5^d,2^e3^c5^d} \in L(\mathcal{G}, A_i)$. Here the width is multiplied by 8/5, which is possible if and only if the second counter is not zero. Then the production $A_j \to \begin{matrix} E_i \\ Q^{3,8} \end{matrix}$ allows $a^{2^{e+3}3^c5^{d-1},2^{e+3}3^c5^{d-1}} \in L(\mathcal{G}, A_j)$ analogously.
5. If M zero-tests the first counter going from q_i to q_j then the productions $F_i \to A_i N^{3,1}$ and $F_i \to A_i N^{3,2}$ allow $a^{2^e3^05^d,2^{e+1}3^05^d} \in L(\mathcal{G}, F_i)$ for $a^{2^e3^05^d,2^e3^05^d} \in L(\mathcal{G}, A_i)$. Here the width is multiplied by 2 and the first counter is zero

which corresponds to width and height of a picture in $L(\mathcal{G}, N^{3,1})$ respectively $L(\mathcal{G}, N^{3,2})$ being congruent 1 respectively 2 modulo 3. Then the production $A_j \to {}^{F_i}_{Q^{1,2}}$ allows $a^{2^{e+1}3^0 5^d, 2^{e+1}3^0 5^d} \in L(\mathcal{G}, A_j)$.

6. If M zero-tests the second counter going from q_i to q_j then the productions $G_i \to A_i N^{5,1}$ and $G_i \to A_i N^{5,2}$ and $G_i \to A_i N^{5,3}$ and $G_i \to A_i N^{5,4}$ allow $a^{2^e 3^c 5^0, 2^{e+1}3^c 5^0} \in L(\mathcal{G}, G_i)$ for $a^{2^e 3^c 5^0, 2^e 3^c 5^0} \in L(\mathcal{G}, A_i)$. Here the width is multiplied by 2 and the second counter is zero which corresponds to width and height of a picture in $L(\mathcal{G}, N^{5,m})$ being congruent $0 < m < 5$ modulo 5. Then the production $A_j \to {}^{G_i}_{Q^{1,2}}$ allows $a^{2^{e+1}3^c 5^0, 2^{e+1}3^c 5^0} \in L(\mathcal{G}, A_j)$.

In each case, the production from A_j takes care to complete again to a square picture thus exactly those pictures representing an encoding of a reachable configuration with state q_j are produced. Since $L(\mathcal{G}) = L(\mathcal{G}, A_n)$ is non-empty if and only if M halts, this concludes the proof. □

Decidability of the emptiness for one-dimensional context-free grammars is guaranteed by the well known *pumping lemma*. We give its full formulation here as we will utilize it in the proofs of the two next theorems.

Theorem 2. *(See [14].) Let $\mathcal{G} = (V_N, V_T, \mathcal{P}, S)$ be a context-free grammar in the Chomsky normal form. Let $p = 2^{|V_N|-1}$ and $q = 2^{|V_N|}$. If $z \in L(\mathcal{G})$ and $|z| > p$, then z can be written as $z = uvwxy$, where $|vwx| \le q$ and $|vx| > 0$, such that for each $i \in \mathbb{N}$, $uv^i wx^i y \in L(\mathcal{G})$.*

Theorem 3. *Emptiness for* 2MG *is decidable.*

Proof. Let $\mathcal{G} = (V_1, V_2, V_T, \mathcal{P}, S)$ be a 2MG where, w.l.o.g., $V_T = \{a\}$. Denote $n = |V_N|$. Assume there is $P \in L(\mathcal{G})$. By inspecting how P is generated, we show it is always possible to generate a picture whose dimensions do not exceed $\mathcal{O}(2^{n^3})$. To obtain P, productions of type (3) generate a string $w \in V_1^+$ from S. Then, nonterminals of w are substituted by one-column pictures of the same length. Define the string language

$$L = \bigcap_{N \in V(w)} L(\mathcal{G}, N)$$

where $V(w)$ is the set of all nonterminals appearing in w. It holds $L \neq \emptyset$.

We can assume $|w| \le 2^n$, otherwise Theorem 2 provides w' over $V(w)$ of length at most 2^n generated from S which can be considered instead of w. The language L is the intersection of unary context-free languages where each context-free grammar has at most n nonterminals. Results in [17] show that each such a grammar has an equivalent deterministic finite automaton (DFA) with $\mathcal{O}(2^{n^2})$ states, where the hidden multiplicative constant does not depend on the grammar because the lengths of the productions are fixed. Thus, L is accepted by the product automaton of at most n DFAs. This automaton has $\mathcal{O}(2^{n^3})$ states. If it accepts a nonempty language, it accepts a string of length $\mathcal{O}(2^{n^3})$. Hence, $L(\mathcal{G})$ contains a picture with $\mathcal{O}(2^n)$ rows and $\mathcal{O}(2^{n^3})$ columns. □

Corollary 4. 2MG *is less powerful than* 2KG *(even over unary alphabets).*

4. Properties of unary context-free picture languages

The result from the previous section indicates there is no general variant of the pumping lemma for picture languages generated by 2KG. On the other hand, we formulate and prove a pumping lemma for a sort of "wide" and "high" pictures.

Theorem 5. *Let L be a picture language over $\{a\}$ generated by a* 2KG *with a set of nonterminals V_N. Let $a^{m,n}$ be a picture in L. It holds that $n \ge 2^{m|V_N|}$ implies $a^{m,n+i\cdot n!} \in L$ and $m \ge 2^{n|V_N|}$ implies $a^{m+i\cdot m!,n} \in L$ for all $i \in \mathbb{N}$.*

Proof. We prove the theorem w.l.o.g. for wide pictures. To simplify the notation within the proof, we write (m, n) to denote the picture $a^{m,n}$. Let $\mathcal{G} = (V_N, \{a\}, \mathcal{P}, S_0)$ be a 2KG. Define the one-dimensional context-free grammar $\mathcal{G}' = (V_N, \{a\}, \mathcal{P}', S_0)$ where $\mathcal{P}'$ consists of those productions in $\mathcal{P}$ which are in the form (1), (2) and (3). For every $N \in V_N$ and $m \in \mathbb{N}^+$, define a picture language $L(N, m)$ as follows:

$$L(N, m) = \left\{ P \mid P \in L(\mathcal{G}, N) \wedge \operatorname{rows}(P) = m \wedge \operatorname{cols}(P) \ge 2^{m|V_N|} \right\}.$$

Proceed by induction on m. Let P be a picture in $L(N, 1)$. It is a one-row picture of length $n = \operatorname{cols}(P) \ge 2^{|V_N|}$. Theorem 2 is applicable and it yields $(1, n + j \cdot k) \in L(N, 1)$ for every $j \in \mathbb{N}$ and some $1 \le k \le n$, thus $(1, n + i \cdot n!) \in L(N, 1)$ for every $i \in \mathbb{N}$ by choosing $j = i \cdot (n!/k)$.

Let $m > 1$. A picture $P \in L(N, m)$ can be written as

1. $P = P_1 \ominus P_2$ where $P_1 \in L(\mathcal{G}, A_1)$, $P_2 \in L(\mathcal{G}, A_2)$, $N \to \begin{matrix} A_1 \\ A_2 \end{matrix} \in \mathcal{P}$, or

2. $P = P_1 \oplus P_2$ where $P_1 \in L(\mathcal{G}, A_1)$, $P_2 \in L(\mathcal{G}, A_2)$, $N \to A_1\, A_2 \in \mathcal{P}$.

Assume, w.l.o.g, that the initial nonterminal S_0 is not a part of the right-hand side of any production, hence P_1, P_2 are nonempty. Denote again $n = \mathrm{cols}(P)$. In the first case it holds $P_1 = (m_1, n)$, $P_2 = (m_2, n)$ where $m_1, m_2 < m$, $P_1 \in L(A_1, m_1)$ and $P_2 \in L(A_2, m_2)$. The induction hypotheses yields $(m_1, n + i \cdot n!) \in L(A_1, m_1)$ and $(m_2, n + i \cdot n!) \in L(A_2, m_2)$ for every $i \in \mathbb{N}$. It is thus possible to generate any $(m, n + i \cdot n!)$ from N.

In the second case, consider a more extensive decomposition of P defined as follows. Take a picture P_i, $i \in \{1, 2\}$ with the maximal number of columns. There is again a production of type (2) or (3) and a decomposition of P_i into two parts proving that $P_i \in L(\mathcal{G}, A_i)$. The process decomposing a picture with the maximal number of columns can be repeated at most $2^{|V_N|} - 1$ times until one of two following states is reached:

1. $P = U_1 \oplus \ldots \oplus U_s$ where $s = 2^{|V_N|}$, or
2. $P = U_1 \oplus \ldots \oplus U_{j-1} \oplus (U_j \ominus U_{j+1}) \oplus U_{j+2} \oplus \ldots \oplus U_s$ where $s \leq 2^{|V_N|}$.

Let B_i be that nonterminal on the right-hand side of the production used during the decomposition process to produce U_i, so it holds $U_i \in L(\mathcal{G}, B_i)$. In the first case, only one-row productions of $\mathcal{P}'$ are used, we can thus write $N \Rightarrow^*_{\mathcal{G}'} B_1 \ldots B_s$, meaning that a sentential form of length $s = 2^{|V_N|}$ is generated from N in $\mathcal{G}'$. Theorem 2 applies to it. Substituting U_i's for B_i's in the pumped sentential forms proves that $(m, n + j \cdot k) \in L(N, m)$ for $k \leq n$ and all $j \in \mathbb{N}$. Again, choosing $j = i \cdot (n!/k)$ shows that $(m, n + i \cdot n!) \in L(N, m)$ for all $i \in \mathbb{N}$.

In the second case, let $U = U_j \ominus U_{j+1}$ denote the picture decomposed as the last one. Its number of columns is maximal when compared to the number of columns of pictures U_i, $i \in \{1, \ldots, s\} \setminus \{j, j+1\}$, hence $\mathrm{cols}(U) \geq \mathrm{cols}(P)/(s-1) \geq \mathrm{cols}(P)/s \geq 2^{m|V_N|}/2^{|V_N|} = 2^{(m-1)|V_N|}$. Since $\mathrm{rows}(U_j), \mathrm{rows}(U_{j+1}) \leq m - 1$, by the induction hypotheses, it is possible to pump U_j and U_{j+1} so that any $(m, n + i \cdot n!)$ is in $L(\mathcal{G}, N)$. □

Giammarresi and Restivo described important classes of functions that can be represented by picture languages in REC [15]. They also gave an upper bound on the growth of such functions. Here we study functions representability by picture languages generated by 2KG and come to the same conclusions – all our results coincide with their analogs from [15].

Definition 4. A function $f : \mathbb{N}^+ \to \mathbb{N}^+$ is called *representable* by 2KG if the picture language $L(f) = \{ a^{n, f(n)} \mid n \in \mathbb{N}^+ \}$ is generated by a 2KG.

We will utilize the fact that the class of languages generated by 2KG is closed under the concatenation operations.

Lemma 6. *Let $\mathcal{G}_1 = (V_1, V_T, \mathcal{P}_1, S_1)$, $\mathcal{G}_2 = (V_2, V_T, \mathcal{P}_2, S_2)$ be 2KG. Then, $L(\mathcal{G}_1) \oplus L(\mathcal{G}_2)$ as well as $L(\mathcal{G}_1) \ominus L(\mathcal{G}_2)$ can be generated by a 2KG.*

Proof. W.l.o.g, assume $V_1 \cap V_2 = \emptyset$ and $S \notin V_1 \cup V_2$. Then, e.g., $L(\mathcal{G}_1) \oplus L(\mathcal{G}_2)$ is generated by $\mathcal{G} = (V_1 \cup V_2, V_T, \mathcal{P}_1 \cup \mathcal{P}_2 \cup \{S \to S_1\, S_2\}, S)$. □

Lemma 7. *If f, g are two functions representable by 2KG and $c \in \mathbb{N}^+$, then cf and $f + g$ are also representable by 2KG.*

Proof. We can write $L(f + g) = L(f) \oplus L(g)$ and $L(cf) = L(\lfloor c/2 \rfloor f) \oplus L(\lceil c/2 \rceil f)$, which can be recursively applied to reduce the multiplier c to 1. By Lemma 6, the concatenation products can be generated by 2KG. □

Lemma 8. *For every $d \in \mathbb{N}$, function $f(n) = n^d$ is representable by 2KG.*

Proof. We prove the lemma by induction on d. If $d = 0$, the constant function $f(n) = 1$ is represented by a 2KG generating all one-column pictures. If $d = 1$, function $f(n) = n$ is represented by the picture language of squares from Example 1. Let $d > 1$. For $n > 1$, an application of the binomial theorem gives

$$n^d = ((n-1) + 1)^d = (n-1)^d + \sum_{i=1}^{d} \binom{d}{i} (n-1)^{d-i} = (n-1)^d + h(n-1)$$

where h is a polynomial of degree $d - 1$. By the induction hypothesis and Lemma 7, there is a 2KG $\mathcal{G} = (V_N, \{a\}, \mathcal{P}, H)$ such that $L(\mathcal{G}) = L(h)$. Extend $\mathcal{G}$ to $\mathcal{G}' = (V_N \cup \{S, U, R\}, \{a\}, \mathcal{P}', S)$ where S, U, R are not contained in V_N and $\mathcal{P}'$ is $\mathcal{P}$ extended by productions

$$R \to a, \qquad R \to R\, R, \qquad S \to a, \qquad S \to \begin{matrix} U \\ R \end{matrix}, \qquad U \to S\, H.$$

Then, $L(\mathcal{G}') = L(n^d)$. □

Lemma 9. *For each integer $c \geq 2$, the exponential function $f(n) = c^n$ is representable by* 2KG.

Proof. The lemma is valid for $c = 2$ since $L(2^n)$ is the picture language from Example 2. The productions in Example 2 can be generalized to generate $L(c^n)$ for a given $c \geq 2$ as follows:

$$R \to a, \qquad R \to R\,R, \qquad A \to a,$$

$$E \to \overbrace{A \ldots A}^{c}, \qquad E \to \begin{matrix} D \\ R \end{matrix}, \qquad D \to \overbrace{E \ldots E}^{c}. \qquad \square$$

Taking into account Theorem 5, we can observe that functions which are of a greater than exponential growth cannot be represented by a 2KG.

Corollary 10. *If f is representable by* 2KG *then $f(n) = 2^{\mathcal{O}(n)}$.*

5. Conclusion

We have shown that the emptiness is undecidable for the two-dimensional Kolam grammar. This gives a final answer to the problem we have recently stated [1,13].

The obtained result is consistent with the known fact that the two-dimensional topology strengthens properties of languages accepted/generated by generalizations of one-dimensional automata/grammars. Moreover, two-dimensional context-free productions in the Chomsky normal form are a minimal setting. The proved emptiness undecidability is thus inherited by all two-dimensional grammars that in some form include productions of the Kolam grammar, such as those in [7,8,10].

The result has several consequences. By [13], we can now claim that transforming a Kolam grammar to an equivalent deterministic four-way automaton (provided that such an automaton exists) yields a non-recursive trade-off. This holds even for grammars generating finite unary picture languages. It complements the analogous result known for the transformation in the opposite direction.

Another consequence worth to be pointed out is a separation result between the Kolam grammar over a unary alphabet and less complex models of automata/grammars having the emptiness decidable (e.g. the matrix grammar).

Acknowledgement

The first author was supported by the Czech Science Foundation under grant no. 15-04960S.

References

[1] D. Průša, (Un)decidability of the emptiness problem for multi-dimensional context-free grammars, in: F. Drewes (Ed.), Implementation and Application of Automata – 20th International Conference, Proceedings, CIAA 2015, Umeå, Sweden, August 18–21, 2015, in: Lecture Notes in Computer Science, vol. 9223, Springer, 2015, pp. 250–262.

[2] G. Siromoney, R. Siromoney, K. Krithivasan, Abstract families of matrices and picture languages, Comput. Graph. Image Process. 1 (3) (1972) 284–307, http://dx.doi.org/10.1016/S0146-664X(72)80019-4.

[3] G. Siromoney, R. Siromoney, K. Krithivasan, Picture languages with array rewriting rules, Inf. Control 22 (5) (1973) 447–470, http://dx.doi.org/10.1016/S0019-9958(73)90573-1.

[4] O. Matz, Regular expressions and context-free grammars for picture languages, in: 14th Annual Symposium on Theoretical Aspects of Computer Science, Springer-Verlag, 1997, pp. 283–294.

[5] M.I. Schlesinger, Matematiceskie sredstva obrabotki izobrazenij (Mathematic tools for image processing), in Russian, Naukova Dumka, Kiev, 1989.

[6] M.I. Schlesinger, V. Hlaváč, Ten Lectures on Statistical and Structural Pattern Recognition, Computational Imaging and Vision, 1st edition, Springer, 2012.

[7] D. Průša, Two-dimensional languages, Ph.D. thesis, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic, 2004.

[8] M. Pradella, A. Cherubini, S.C. Reghizzi, A unifying approach to picture grammars, Inform. and Comput. 209 (9) (2011) 1246–1267, http://dx.doi.org/10.1016/j.ic.2011.07.001.

[9] F. Drewes, S. Ewert, R. Klempien-Hinrichs, H. Kreowski, Computing raster images from grid picture grammars, J. Autom. Lang. Comb. 8 (3) (2003) 499–519.

[10] T. Kamaraj, D.G. Thomas, Regional hexagonal tile rewriting grammars, in: R. Barneva, V. Brimkov, J. Aggarwal (Eds.), Combinatorial Image Analysis, in: Lecture Notes in Computer Science, vol. 7655, Springer Berlin Heidelberg, 2012, pp. 181–195.

[11] M. Blum, C. Hewitt, Automata on a 2-dimensional tape, in: Proceedings of the 8th Annual Symposium on Switching and Automata Theory (SWAT 1967), FOCS '67, IEEE Computer Society, Washington, DC, USA, 1967, pp. 155–160.

[12] J. Kari, C. Moore, Rectangles and squares recognized by two-dimensional automata, in: J. Karhumäki, H.A. Maurer, G. Paun, G. Rozenberg (Eds.), Theory Is Forever, in: Lecture Notes in Computer Science, vol. 3113, Springer, 2004, pp. 134–144.

[13] D. Průša, Non-recursive trade-offs between two-dimensional automata and grammars, in: H. Jürgensen, J. Karhumäki, A. Okhotin (Eds.), Proceedings of the 16th International Workshop on Descriptional Complexity of Formal Systems, DCFS 2014, in: Lecture Notes in Computer Science, vol. 8614, Springer International Publishing, Berlin, Germany, 2014, pp. 352–363.

[14] J. Hopcroft, J. Ullman, Formal Languages and Their Relation to Automata, Addison–Wesley, 1969.
[15] D. Giammarresi, A. Restivo, Two-dimensional languages, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Vol. 3, Springer, New York, 1997, pp. 215–267.
[16] M.L. Minsky, Computation: Finite and Infinite Machines, Prentice–Hall, 1971.
[17] G. Pighizzini, J. Shallit, M. Wang, Unary context-free grammars and pushdown automata, descriptional complexity and auxiliary space lower bounds, J. Comput. System Sci. 65 (2) (2002) 393–414, http://dx.doi.org/10.1006/jcss.2002.1855.

Theoretical Computer Science 610 (2016) 121–132

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Non-recursive trade-offs between two-dimensional automata and grammars

Daniel Průša

Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University, Zikova 1903/4, 166 36 Prague 6, Czech Republic

ARTICLE INFO

Article history:
Received 31 October 2014
Received in revised form 15 April 2015
Accepted 22 May 2015
Available online 28 May 2015

Keywords:
Picture languages
Four-way automata
Two-dimensional context-free grammars
Descriptional complexity
Non-recursive trade-offs

ABSTRACT

We study succinctness of descriptional systems for picture languages. Basic models of two-dimensional finite automata and generalizations of context-free grammars are considered. They include the four-way automaton of Blum and Hewitt, the two-dimensional online tessellation automaton of Inoue and Nakamura and the context-free Kolam grammar of Siromoney et al. We show that non-recursive trade-offs between the systems are very common. Basically, each separation result proving that one system describes a picture language which cannot be described by another system can usually be turned into a non-recursive trade-off result between the systems. These findings are strongly based on the ability of the systems to simulate Turing machines.

1. Introduction

Many concepts and techniques from the theory of formal languages have been generalized to two-dimensional (2D) languages, where the basic entity, the *string*, has been replaced by a rectangular array of symbols, called a *picture*. The *four-way finite automaton* of Blum and Hewitt [1] was introduced already in 1967. It has a finite-state control unit and a head that traverses the input picture by performing movements right, left, up, and down.

Another notable device is the *two-dimensional online tessellation automaton* (2OTA) proposed by Inoue and Nakamura [2]. This is a restricted nondeterministic cellular automaton where a "transition wave" passes once diagonally across the cells. Its recognition power coincides with the power of tiling systems [3] which are the basis for the well known family of recognizable picture languages (REC) of Giammaresi and Restivo [4].

The early models of picture grammars include matrix and Kolam grammars of Siromoney et al. [5,6]. Kolam grammars were independently proposed and studied by Matz [7] and also by Schlesinger [8,9] who designed them as a tool for structural pattern recognition. The grammars are characterized by the form of productions which resembles the Chomsky normal form. Two extensions of the grammars are known – the first one described by Průša [10] and the second one, even more general, studied by Pradella et al. [11].

Since the beginning, it was evident that the two-dimensional topology of pictures changes a lot those properties of automata and grammars known from the one-dimensional case. For example, Blum and Hewitt proved that the nondeterministic four-way finite automaton (4NFA) is more powerful than the deterministic four-way finite automaton (4DFA). Equipping 4DFA by a pebble again results in a more powerful device. Advantages of the four-way alternating automaton (4AFA) were described by Kari and Moore [12]. The mentioned 2D grammars generate different classes of picture languages. Many differences have also been revealed for closure properties and decidability problems.

E-mail address: prusapa1@cmp.felk.cvut.cz.

http://dx.doi.org/10.1016/j.tcs.2015.05.033
0304-3975/

$P_{1,1}$	$P_{1,2}$	$P_{1,3}$
$P_{2,1}$	$P_{2,2}$	$P_{2,3}$

Fig. 1. The product of $\bigoplus[P_{ij}]_{2\times 3}$.

Despite these extensive studies, so far, no comparison has been done with respect to the descriptional complexity of the considered models. This paper aims to fill this gap. We show that there are many non-recursive trade-offs between the descriptional systems for picture languages.

Examples of such trade-offs are well known in the case of descriptional systems for string languages. The first non-recursive trade-off was presented by Meyer and Fisher [13]. They showed that the gain in economy of description can be arbitrary when the size of finite automata and general context-free grammars generating regular languages is compared. Since their work, other non-recursive trade-offs were reported, such as in [14–18]. Besides these particular results, the important properties of systems leading to non-recursive trade-offs were identified and generic proof schemes were established [19,20].

Our results are based on the ability of two-dimensional systems to simulate Turing machines. Known principles of a Turing machine simulation by a 4DFA are exploited to prove non-recursive trade-offs between 2D automata. In addition, a new technique of a Turing machine simulation by the 2D grammar from [10] is presented. It is utilized to prove non-recursive trade-offs between automata and grammars and also between the grammars themselves. Applicability of this result goes even beyond the scope of descriptional complexity, as it also answers some decidability questions.

The paper is structured as follows. In Section 2 we give the basic notions and notations on picture languages. In Section 3 we show non-recursive trade-off between 4DFA and 4NFA. It is also explained how this result extends to other automata. Descriptional complexity of 2D grammars is studied in Section 4. A detailed construction of a 2D grammar simulating a Turing machine is included here. The paper closes with a summary and some open problems in Section 5.

2. Preliminaries

Here we use the common notation and terms on pictures and picture languages (see, e.g., [21]). If Σ is a finite alphabet, then $\Sigma^{*,*}$ is used to denote the set of all rectangular pictures over Σ, that is, if $P \in \Sigma^{*,*}$, then P is a two-dimensional array (matrix) of symbols from Σ. If P is of size $m \times n$, this is denoted by $P \in \Sigma^{m,n}$. We also write $\mathrm{rows}(P) = m$ and $\mathrm{cols}(P) = n$. If P is a square picture $n \times n$, we shortly say P is of size n. $\Sigma^{+,+} = \{P \in \Sigma^{*,*} \mid \mathrm{rows}(P) > 0 \wedge \mathrm{cols}(P) > 0\}$ is the set of non-empty pictures. The empty picture Λ is defined as the only picture of size 0×0.

We use $[a_{ij}]_{m\times n}$ as a notation for a general matrix with m rows and n columns where the element in the i-th row and j-th column is denoted as a_{ij}.

Two (partial) binary operations are introduced to concatenate pictures. Let $A = [a_{ij}]_{k\times \ell} \in \Sigma^{k,\ell}$ and $B = [b_{ij}]_{m\times n} \in \Sigma^{m,n}$. The *column concatenation* $A \oplus B$ is defined iff $k = m$, and the *row concatenation* $A \ominus B$ is defined iff $\ell = n$. The products are specified by the following schemes:

$$A \oplus B = \begin{bmatrix} a_{11} & \dots & a_{1\ell} & b_{11} & \dots & b_{1n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{k1} & \dots & a_{k\ell} & b_{m1} & \dots & b_{mn} \end{bmatrix} \text{ and } A \ominus B = \begin{bmatrix} a_{11} & \dots & a_{1\ell} \\ \vdots & \ddots & \vdots \\ a_{k1} & \dots & a_{k\ell} \\ b_{11} & \dots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{m1} & \dots & b_{mn} \end{bmatrix}.$$

We generalize $\ominus$ and $\oplus$ to a *grid concatenation* which is applied to a matrix of pictures $[P_{ij}]_{m\times n}$ where each $P_{ij} \in \Sigma^{*,*}$. The operation $\bigoplus[P_{ij}]_{m\times n}$ is defined iff

$$\mathrm{rows}(P_{i1}) = \mathrm{rows}(P_{i2}) = \ldots = \mathrm{rows}(P_{in}) \quad \forall i = 1, \ldots, m\,,$$
$$\mathrm{cols}(P_{1j}) = \mathrm{cols}(P_{2j}) = \ldots = \mathrm{cols}(P_{mj}) \quad \forall j = 1, \ldots, n\,.$$

Then, $\bigoplus[P_{ij}]_{m\times n} = P_1 \ominus P_2 \ominus \ldots \ominus P_m$, where $P_k = P_{k1} \oplus P_{k2} \oplus \ldots \oplus P_{kn}$ for $k = 1, \ldots, m$. An example is given in Fig. 1.

In order to enable all considered finite automata to detect the border of an input picture P, they always work over the *boundary picture* $\widehat{P}$ over $\Sigma \cup \{\#\}$ of size $(\mathrm{rows}(P) + 2) \times (\mathrm{cols}(P) + 2)$, defined by the scheme in Fig. 2. We assume that the *background symbol* # is not contained in any considered input alphabet Σ.

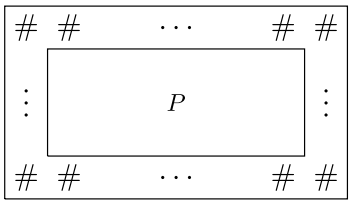


Fig. 2. The boundary picture $\widehat{P}$.

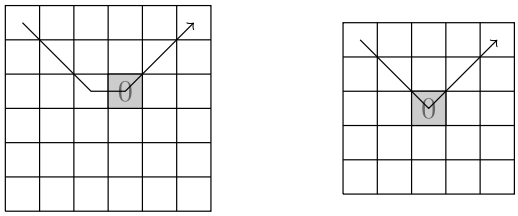


Fig. 3. Two pictures in L_{cent}. 4NFA recognizes the pictures by guessing the shown trajectories (they end in the top-right corner) and checking the highlighted central field.

3. Two-dimensional finite automata

We first outline informally the proof pattern applied throughout the paper. Then we demonstrate its usage in detail to show that transforming 4NFA to 4DFA yields a non-recursive trade-off. For every automaton $\mathcal{A}$ considered, we define its *size measure* $c(\mathcal{A})$ simply as the number of states of $\mathcal{A}$ (i.e., $c(\mathcal{A})$ is state complexity).

Let $\mathcal{D}_1$, $\mathcal{D}_2$ be two (automata based) descriptional systems fulfilling the following requirements.

- There is $\mathcal{A}_1 \in \mathcal{D}_1$ accepting a picture language over Σ which is not accepted by any $\mathcal{A}_2 \in \mathcal{D}_2$.
- The reason why there is no suitable $\mathcal{A}_2 \in \mathcal{D}_2$ can be stated as follows. There is a non-decreasing unbounded recursive function $f : \mathbb{N} \to \mathbb{N}$ such that any automaton in $\mathcal{D}_2$ needs $\Omega(f(n))$ states to correctly recognize all square pictures of size n, i.e. to accept every picture in $L(\mathcal{A}_1) \cap \Sigma^{n,n}$ and to reject every picture in $\Sigma^{n,n} \setminus L(\mathcal{A}_1)$.
- Automata in $\mathcal{D}_1$ can be extended to compare the size of the input picture with the length of a computation of an arbitrary (one-dimensional) Turing machine started over a blank tape. The number of states added by the extension is recursive in the size of the given Turing machine.

These assumptions allow us to construct automata of "small" size from $\mathcal{D}_1$ accepting finite subsets of $L(\mathcal{A}_1)$ that include "large" pictures. Each such finite language is also accepted by an automaton from $\mathcal{D}_2$, however, the automaton is forced to have a "large" number of states.

Blum and Hewitt [1] proved that 4NFA is more powerful than 4DFA using the picture language of odd length square pictures over $\{0, 1\}$ containing 0 in the center. Here we slightly modify the language to include even length square pictures as well and we follow their proof to derive an estimate on the number of states needed by a 4DFA to recognize all square pictures of size n in the language.

Let L_{cent} be a picture language over $\Sigma = \{0, 1\}$ consisting of all square pictures P that contain 0 at (the central) position $(\lceil \frac{n+1}{2} \rceil, \lceil \frac{n+1}{2} \rceil)$ where $n = \text{rows}(P)$. This language is accepted by a 4NFA by checking that the input is a square picture, followed by guessing the trajectory depicted in Fig. 3 and verifying that the central field stores 0.

Lemma 1. *For $n \in \mathbb{N}$, let $\mathcal{A}$ be a 4DFA such that $L(\mathcal{A}) \cap \Sigma^{n,n} = L_{\text{cent}} \cap \Sigma^{n,n}$. Then, $\mathcal{A}$ has $\Omega(n/\log n)$ states.*

Proof. Let Q be the set of states of $\mathcal{A}$ and let $P \in L_{\text{cent}}$ be a picture over $\{0, 1\}$ of size $n \times n$. Define $k = \lceil \frac{n-1}{2} \rceil$ and take a block (sub-picture) B of size $k \times k$ in P which does not contain the top left corner of P. Consider the behavior of $\mathcal{A}$ when working over B. It can enter it at one of $4k-4$ positions of the perimeter, being in one of $|Q|$ states. After performing some steps, it either leaves the block, again at some position and in one of the states, or it accepts or rejects (rejecting includes cycling inside the block). Thus the block defines a mapping from $\{1, \ldots, 4k-4\} \times Q$ to $(\{1, \ldots, 4k-4\} \times Q) \cup \{\text{acc}, \text{rej}\}$ where acc and rej are constants representing accepting and rejecting of $\mathcal{A}$, respectively.

Observe that it is not possible to have two different blocks B_1, B_2 with the same mapping. Consider that (i, j) is a position where the blocks differ. Construct the picture P_1 of size $n \times n$ which includes B_1, having its field at (i, j) placed in the center of P_1. The fields of P_1 outside B_1 are filled by 0. Similarly, construct P_2 by extending B_2. If the mappings for B_1 and B_2 are identical, pictures P_1 and P_2 are either both accepted by $\mathcal{A}$ or both rejected, which is a contradiction.

There are $(|Q|(4k-4)+2)^{|Q|(4k-4)}$ mappings defined by blocks of size $k \times k$. On the other hand, a two-letter alphabet forms 2^{k^2} pictures of size $k \times k$. This results in the inequality $(|Q|(4k-4))\log_2(|Q|(4k-4)+2) \geq k^2$ which implies a lower bound of the form

$$|Q| = \Omega\left(\frac{k}{\log_2 k}\right) = \Omega\left(\frac{n}{\log n}\right). \qquad \square$$

In contrast to finite automata working over strings, two-dimensional finite automata are able in some sense to simulate (one-dimensional) Turing machines. Two principles of the simulation can be pointed out. The first one is content dependent. 4DFA can check whether rows of the input picture encode consecutive configurations of a Turing machine. This is applicable to prove undecidability of the emptiness problem for 4DFAs as well as other models [4].

The second, content independent approach better suits our needs. As has been observed by Kari and Moore [22], a 4DFA can operate as a 2-counter machine [23] which consists of a finite-state control unit and two integer registers, called counters. It is possible to represent the value of the first and second register by the horizontal and vertical distance of the automaton head from the initial top-left corner position, respectively. Moving the head by one tape field thus increments/decrements a register. Checking a register value for zero is done by detecting whether the head scans the first row or column, respectively. Naturally, the simulated registers are not unbounded, the maximal value they can store is limited by the size of the input.

As showed by Minsky [23], 2-counter machines are equivalent to Turing machines. The conversion of a binary alphabet Turing machine to a 2-counter machine is done in three steps. The input to the Turing machine is assumed to be encoded in unary. The steps are as follows. First, a Turing machine $\mathcal{T}$ can be simulated by a finite-state automaton equipped by two stacks. The head of $\mathcal{T}$ splits the tape into two halves. Each half of the tape can be treated as a stack, where the top is the cell nearest the head. Moving the head left or right is equivalent to popping a symbol from one stack and pushing it onto the other. Second, a stack can be simulated using two counters. The content of a stack is considered as a binary number, represented in one counter. Operations divide by 2 and multiply by 2 are needed to implement pushing and popping of symbols. They are realized trough the second counter. And third, four counters can be simulated by two counters. One counter holds an integer with prime factorization $2^a 3^b 5^c 7^d$ where exponents a, b, c, d are values of the represented counters. Increments and decrements performed over them can be simulated using the second counter.

In the next text, we work with the busy beaver function defined as follows.

Definition 1. Let T_n be the set of all binary alphabet n-state one-dimensional Turing machines. For $\mathcal{T} \in T_n$, let $t(\mathcal{T})$ be the number of transitions performed by $\mathcal{T}$ when started over a blank tape in the case it halts, otherwise let $t(\mathcal{T}) = 0$. The *busy beaver function*, $\mathrm{bb} : \mathbb{N} \to \mathbb{N}$, is defined such that

$$\mathrm{bb}(n) = \max_{\mathcal{T} \in T_n} t(\mathcal{T}) \qquad \forall n \in \mathbb{N}.$$

It is known that the busy beaver function grows faster than any recursive function [24]. Halting Turing machines in T_n performing the maximal number of transitions are referred as n-state busy beavers.

Proposition 2. *For any finite alphabet Σ, there is an infinite sequence of 4DFAs $\{\mathcal{A}_k\}_{k=1}^{\infty}$ over Σ and a recursive function $f : \mathbb{N} \to \mathbb{N}$, such that $c(\mathcal{A}_k) \leq f(k)$ and $\mathcal{A}_k$ accepts a finite picture language which for some $n \geq \mathrm{bb}(k)$ includes all pictures in $\Sigma^{n,n}$.*

Proof. Let $\mathcal{T}_1, \ldots, \mathcal{T}_s$ be a recursive enumeration of all one-dimensional deterministic binary k-state Turing machines. Each $\mathcal{T}_i$ (started over a blank tape) can be simulated by a 2-counter machine $\mathcal{C}_i$. W.l.o.g, we assume that $\mathcal{C}_i$ increases the length of $\mathcal{T}_i$'s configuration encoding each time it simulates a step of $\mathcal{T}_i$. This ensures that $\mathcal{C}_i$ never goes into a cycle. Design $\mathcal{A}_k$ as it follows.

- $\mathcal{A}_k$ checks whether the input $P \in \Sigma^{*,*}$ is a square picture, if not it rejects.
- $\mathcal{A}_k$ follows computations of $\mathcal{C}_1, \ldots, \mathcal{C}_s$ to simulate machines $\mathcal{T}_1, \ldots, \mathcal{T}_s$ one by one. If there is a simulation which ends at the moment when the head of $\mathcal{A}_k$ scans the last row or column of P, then $\mathcal{A}_k$ accepts P. If the simulation exceeds the area of P or finishes off the last row and column of P, the simulation of the next $\mathcal{T}_i$ is launched.
- If all simulations of machines $\mathcal{T}_i$ pass without accepting P, $\mathcal{A}_k$ rejects.

$\mathcal{A}_k$ has to memorize in states instructions for each $\mathcal{C}_i$ as well as the index of the currently simulated $\mathcal{T}_i$. However, this is no problem, since the number of states of automata $\mathcal{A}_k$ can grow as fast as any recursive function.

$L(\mathcal{A}_k)$ contains square pictures whose area exactly fits to the k-state busy beaver simulation. The runtime of the Turing machines can only increase when they are turned into counter machines, which is then simulated by a 4DFA, thus the requirement claimed by the proposition is fulfilled. $\square$

Theorem 3. *The trade-off between 4NFAs and 4DFAs is non-recursive.*

Proof. Let $\mathcal{M}$ be a 4NFA accepting L_{cent}. For each k, combine $\mathcal{A}_k$ from Proposition 2 and $\mathcal{M}$ to obtain a 4NFA $\mathcal{M}_k$ accepting the finite language $L_{\text{cent}} \cap L(\mathcal{A}_k)$. By Lemma 1, a 4DFA needs $\Omega(\text{bb}(k)/\log \text{bb}(k))$ states to recognize $L(\mathcal{M}_k)$. □

The used approach generalizes to other pairs of systems where a picture language separating the classes of accepted picture languages is known. Here we list the most fundamental ones.

1. One-pebble 4DFA is more powerful than 4DFA since the pebble usage allows to recognize whether a picture over $\Sigma = \{a, b\}$ contains exactly one connected component of a's [1]. On the other hand, such a picture language is not accepted even by any 4NFA [25].
2. 2OTA is more powerful than 4NFA and than deterministic 2OTA [2]. This also applies to REC as tiling systems and 2OTAare equally powerful.
3. 4AFA is more powerful than 4NFA and its power is incomparable with 2OTA [12]. Kari and Moore showed that 4AFA accepts a picture language of permutation copies. Accepted pictures are of the form $P\Phi C\Phi P$ where P is a square over $\{0, 1\}$, containing exactly one symbol 1 in each its row and column, and C is a column of 2's. This language is not in REC. On the other hand, its complement is in REC, but it is not accepted by any 4AFA.
4. 4DFA is more powerful than two-dimensional three-way alternating automaton (3AFA) [26] which is a 4AFA allowed to move the head only in three directions.

All the witnessing picture languages can be restricted or modified to contain only square pictures. For example, the picture language of permutations accepted by 4AFA contains rectangular pictures of size $n \times (2n+1)$. These pictures can be extended to squares $(2n+1) \times (2n+1)$ by duplicating each row and inserting one more uniform initial row. Since the listed automata are at least as powerful as 4DFA (except 3AFA, however, we compare it with 4DFA), Proposition 2 can be generalized to all the mentioned pairs of models.

4. Two-dimensional context-free grammars

This section gives details on two-dimensional grammars mentioned in the introduction. It focuses mainly on the 2D context-free grammar from [10]. Its relation to Kolam grammar as well as to the *regional tile grammar* proposed by Pradella et al. [11] is explained.

Succinctness of the grammars is compared with that of four-way automata. It is important to emphasize that the result by Meyer and Fisher regarding the non-recursive trade-off between context-free grammars and finite automata applies to 2D context-free grammars and four-way automata, since these models working over one-row inputs collapse to 1D context-free grammars and two-way automata, which accept regular languages, respectively. However, there are still two cases to study. The trade-off in the one-dimensional setting is recursive if unary [27] or finite [13] languages are considered. We show that such restrictions do not play any role in the case of 2D systems. Besides that, non-recursive trade-offs are demonstrated in both directions. Another comparison of succinctness is done between the 2D context-free grammars and Kolam grammars, again resulting in a non-recursive trade-off.

Definition 2. A *two-dimensional context-free grammar* (2CFG) is a tuple $\mathcal{G} = (V_N, V_T, \mathcal{P}, S_0)$, where V_N is a finite set of nonterminals, V_T is a finite set of terminals, $S_0 \in V_N$ is the initial nonterminal and $\mathcal{P}$ is a finite set of productions. Productions are of the form $N \to W$ where N is a nonterminal in V_N and W is a non-empty matrix whose elements are terminals and nonterminals, i.e., $W \in (V_N \cup V_T)^{+,+}$. $\mathcal{P}$ can optionally contain the production $S_0 \to \Lambda$. If so, S_0 cannot be a part of the right-hand side of any production.

Definition 3. Let $\mathcal{G} = (V_N, V_T, \mathcal{P}, S_0)$ be a 2CFG. For each $N \in V_N$, $L(\mathcal{G}, N)$ denotes the set of pictures *generated* by $\mathcal{G}$ from N. A picture $P \in \{V_T\}^{*,*}$ belongs to $L(\mathcal{G}, N)$ iff

1. $N \to P$ is a production, or
2. There is a production $N \to [A_{ij}]_{m\times n}$ and $P = \bigoplus [P_{ij}]_{m\times n}$ where

$$\begin{array}{ll} P_{ij} = A_{ij} & \text{if } A_{ij} \in V_T, \\ P_{ij} \in L\left(\mathcal{G}, A_{ij}\right) & \text{if } A_{ij} \in V_N. \end{array}$$

The picture language generated by $\mathcal{G}$ is defined as $L(\mathcal{G}) = L(\mathcal{G}, S_0)$.

Example 1. Let $\mathcal{G} = (V_N, V_T, \mathcal{P}, R)$ be a 2CFG where $V_N = \{H, V, A, R\}$, $V_T = \{a\}$ and $\mathcal{P}$ consists of the following productions

$$H \to a, \qquad H \to a \;\; H\,, \qquad V \to a, \qquad V \to \begin{matrix} a \\ V \end{matrix},$$

$$A \to V, \qquad A \to V \;\; A\,, \qquad R \to a, \qquad R \to \begin{matrix} R & V \\ H & a \end{matrix}.$$

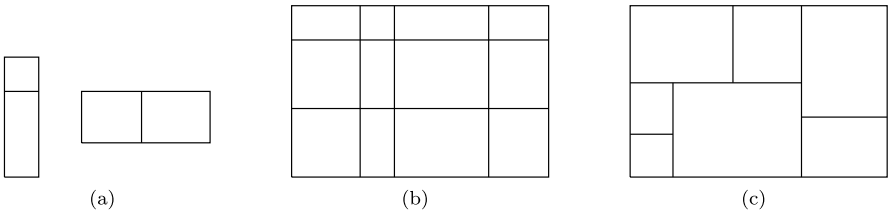


Fig. 4. An illustration in what ways are bigger pictures composed from smaller ones in the case of (a) Kolam grammar, (b) 2CFG, and (c) regional tile grammar.

1	1	0	1	0	1
0	·	·	·	·	0
0	·	·	·	·	0
0	·	·	·	·	0
1	·	·	·	·	1
1	1	0	1	0	1

(a)

0	1	0	0	0	0
0	1	0	0	0	0
1	1	1	1	1	1
0	1	0	0	0	0
0	1	0	0	0	0
0	1	0	0	0	0

(b)

Fig. 5. An example of a picture from (a) L_{perim}, where the content of inner fields is hidden, and from (b) L_{cross}.

Then, $L(\mathcal{G}, H)$ consists of all one-row pictures of a's, $L(\mathcal{G}, V)$ consists of all one-column pictures of a's, $L(\mathcal{G}, A) = \{a\}^{+,+}$ and $L(\mathcal{G}, R) = L(\mathcal{G})$ contains all non-empty square pictures.

Note that 2D context-free grammars do not have any equivalent to Chomsky normal form. Limiting the size of production's right-hand sides affects the power [10]. A size measure taking into account the number of elements in productions is preferable here over a simple counting of productions.

Definition 4. Let $\mathcal{G} = (V_N, V_T, \mathcal{P}, S_0)$ be a 2CFG where $\mathcal{P} = \{\pi_1, \ldots, \pi_k\}$. For a production $\pi = N \to [A_{ij}]_{m\times n}$ in $\mathcal{P}$, its size is defined as $c(\pi) = mn$. Beside that, $c(S_0 \to \Lambda) = 1$. The size of $\mathcal{G}$ is defined as $c(\mathcal{G}) = \sum_{i=1}^{k} c(\pi_i)$.

Kolam grammar is a special variant of 2CFG with productions of the form $N \to [A_{ij}]_{p\times q}$ where $p = 1$ or $q = 1$ and both $p, q \leq 2$. The mechanism behind the regional tile grammar allows to combine pictures into bigger ones using a freer concatenation, the alignment of subpictures into a grid is not required. It is known that Kolam grammars are less powerful than 2CFGs, while regional tile grammars are more powerful than 2CFGs [11]. Production styles of the grammars are depicted in Fig. 4.

To demonstrate limits of the grammars, we define two picture languages that are easily recognizable by 4DFA. Let L_{perim} be a picture language over $\Sigma = \{0, 1\}$ where $P \in L_{\text{perim}}$ if and only if the first row of P equals the last row, the first column equals the last column and all inner fields contain 0.

The second picture language $L_{\text{cross}} \subseteq \{0, 1\}^{*,*}$ comes from [7]. It consists of pictures where 1's appear exactly in one row and one column. Examples of pictures from both languages are depicted in Fig. 5.

We can easily design a 2CFG $\mathcal{G}$ such that $L(\mathcal{G}) = L_{\text{cross}}$ [10]. However, the following lemma indicates that L_{cross} cannot be generated by any Kolam grammar.

Lemma 4. *(See [7].) For $n \in \mathbb{N}$, let $\mathcal{G} = (V_N, \{0, 1\}, \mathcal{P}, S_0)$ be a Kolam grammar such that $L(\mathcal{G}) \cap \Sigma^{n,n} = L_{\text{cross}} \cap \Sigma^{n,n}$. Then, $c(\mathcal{G}) = \Omega(n)$.*

In the next paragraphs we will show that L_{perim} cannot be generated by any 2CFG.

Lemma 5. *Every 2CFG $\mathcal{G} = (V_N, V_T, \mathcal{P}, S_0)$ has an equivalent grammar $\mathcal{G}' = (V'_N, V_T, \mathcal{P}', S_0)$ such that $c(\mathcal{G}') \leq 3c(\mathcal{G})$ and each production $S_0 \to [A_{ij}]_{p\times q}$ in $\mathcal{P}'$ fulfills*

- *If $p = q = 1$ then A_{11} is a terminal.*
- *If $p > 1$ or $q > 1$ then each A_{ij} is a nonterminal.*

Proof. First, for each $a \in V_T$ figuring at a right-hand side of some production, introduce a new nonterminal N_a, add production $N_a \to a$ and replace all occurrences of a in production's right-hand sides containing two or more elements. Second, replace $A \in V_N$ in $S_0 \to A$ by right-hand sides of other productions reachable from A. Each of the two steps increases size of $\mathcal{G}$ by a maximum of $c(\mathcal{G})$. □

Lemma 6. *For $n \in \mathbb{N}$, let $\mathcal{G} = (V_N, \{0,1\}, \mathcal{P}, S_0)$ be a 2CFG such that $L(\mathcal{G}) \cap \Sigma^{n,n} = L_{\text{perim}} \cap \Sigma^{n,n}$. Then, $c(\mathcal{G}) = \Omega(n/\log n)$.*

Proof. W.l.o.g., $\mathcal{G}$ is in the form specified in Lemma 5 and $n \geq 2$. Let L be the set of all square pictures in L_{perim} of size n. It holds $|L| = 2^{2n-3}$. There is a production proving for at least $\lceil \frac{|L|}{|\mathcal{P}|} \rceil$ pictures from L that they belong to $L(\mathcal{G})$. Let $S_0 \to [A_{ij}]_{k\times\ell}$ be such a production. W.l.o.g., $k \geq \ell$ (implying $k \geq 2$).

Let P, Q be pictures in L such that $P \neq Q$, $P = \bigoplus [P_{ij}]_{k\times\ell}$, $Q = \bigoplus [Q_{ij}]_{k\times\ell}$ and each P_{ij} and Q_{ij} is in $L(\mathcal{G}, A_{ij})$. Observe that if all the pairs P_{1j}, Q_{1j} have the same size, then the first row of P and Q have to be necessarily identical. If not, $\mathcal{G}$ would generate a picture which is not in L_{perim}. Indeed, it would be possible to create P' from P by replacing subpictures P_{1j} by Q_{1j}. However, the first row in P' does not match its last row.

Sizes of pictures P_{1j}, $j = 1, \ldots, \ell$ are determined by one vertical and $\ell - 1$ horizontal coordinates in P. The number of such possibilities is certainly bounded by n^k. For a chosen picture sizes, all related pictures from L differ in content. Since we observed that their first rows must be the same, it is possible to choose symbols only for fields $2, \ldots, n-1$ in the first column. This results in the inequality

$$\frac{2^{2n-3}}{|\mathcal{P}|} \leq n^k 2^{n-2} \quad \Rightarrow \quad |\mathcal{P}| \geq \frac{2^{n-1}}{n^k}.$$

If $k \geq n/(2\log_2 n)$, then $c(\mathcal{G}) \geq c(S_0 \to [A_{ij}]_{k\times\ell}) \geq n/(2\log_2 n)$. If $k < n/(2\log_2 n)$, then

$$c(\mathcal{G}) \geq |\mathcal{P}| \geq \frac{2^{n-1}}{n^{\frac{n}{2\log_2 n}}} = \frac{2^{n-1}}{2^{\frac{n}{2\log_2 n}\log_2 n}} = 2^{\frac{n}{2}-1} = \Omega\left(\frac{n}{\log n}\right),$$

which completes the proof. □

Theorem 7. *Transforming 4DFA to 2CFG yields a non-recursive trade-off.*

Proof. An analogous approach to that one used in the proof of Theorem 3 is applicable here. It is possible to construct 4DFAs of small description size accepting finite subsets of L_{perim} containing large pictures. □

Our next goal is to design a sort of a Turing machine simulation performed with a two-dimensional context-free grammar. This is essential for proving that transforming some systems to 2CFGs also induces non-recursive trade-offs.

Proposition 8. *Let $\mathcal{T}$ be a one-dimensional deterministic Turing machine and w be an input to $\mathcal{T}$. There is a 2D context-free grammar $\mathcal{G}$ over $\{a\}$, computable uniformly in $\mathcal{T}$, such that $L(\mathcal{G})$ is empty if $\mathcal{T}$ does not halt on w, else if $\mathcal{T}$ halts in $t(w)$ steps, $L(\mathcal{G})$ is a one-element set containing a square picture of size at least $t(w)$. Moreover, size of $\mathcal{G}$ is recursive in size of $\mathcal{T}$ and $|w|$.*

Proof. Let $Q = \{q_1, \ldots, q_s\}$ be the set of states of $\mathcal{T}$ and $\Gamma = \{a_1, \ldots, a_r\}$ be the tape alphabet of $\mathcal{T}$, containing the blank symbol #. W.l.o.g, $\mathcal{T}$ is single-tape, the tape is infinite rightwards only and each instruction of $\mathcal{T}$ moves the head.

Let $\mathcal{C}_0, \mathcal{C}_1, \ldots$ be configurations of $\mathcal{T}$ when computing over w. We show how to construct a 2CFG $\mathcal{G} = (\{a\}, V_N, \mathcal{P}, S_0)$ such that

- there is nonterminal C in V_N such that $L(\mathcal{G}, C)$ consists of square pictures whose sizes encode configurations $\mathcal{C}_i$, and
- $L(\mathcal{G}, S_0)$ contains at most one picture – that one which encodes the final configuration.

For convenience, we treat integers also as binary strings and vice versa. Define $\text{code}(q_i) = 0^{i-1}10^{|Q|-i}$ and $\text{code}(a_i) = 0^{i-1}10^{|\Gamma|-i}$. For some $\mathcal{C}_i$, let $b_1 b_2 \ldots$ be the tape content and let the control unit of $\mathcal{T}$ be in a state q. We consider $\mathcal{C}_i$ to be represented by a binary string (integer)

$$\mathcal{C}_i = 1 B_1 B_2 \ldots B_{|w|+i+2}$$

where

$$B_j = \begin{cases} 1\,\text{code}(b_j)\,\text{code}(q) & \text{if } \mathcal{T} \text{ scans the } j\text{-th tape field,} \\ 1\,\text{code}(b_j)0^{|Q|} & \text{otherwise.} \end{cases}$$

Note that the suffix $B_{|w|+i+2}$ always equals $1\,\text{code}(\#)0^{|Q|}$, even if $|w| = 0$.

Our next step is to compare two subsequent configurations $\mathcal{C}_k$ and $\mathcal{C}_{k+1}$. The aim is to propose productions that generate the square picture of size $\mathcal{C}_{k+1} - \mathcal{C}_k$ so that we can combine it with the picture $\mathcal{C}_k \times \mathcal{C}_k$ to obtain the picture $\mathcal{C}_{k+1} \times \mathcal{C}_{k+1}$.

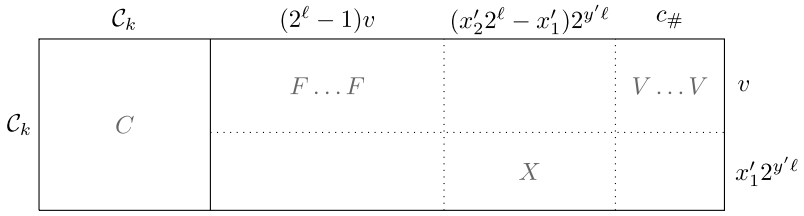


Fig. 6. Substitution of pictures to CJ-part of production (2), sizes of components are given along the border.

Let the head of $\mathcal{T}$ scan the m_1-th and m_2-th tape field in $\mathcal{C}_k$ and $\mathcal{C}_{k+1}$, respectively. Since $\mathcal{T}$ always moves the head, $m_1 \neq m_2$. Denote $m = \min\{m_1, m_2\}$. We can write

$$\mathcal{C}_k = 1B_1B_2 \ldots B_{|w|+k+1}B_{|w|+k+2},$$
$$\mathcal{C}_{k+1} = 1B_1 \ldots B_{m-1}B'_mB'_{m+1}B_{m+2} \ldots B_{|w|+k+2}B_{|w|+k+3}.$$

When comparing $\mathcal{C}_{k+1}$ to $\mathcal{C}_k$, we see it differs in B'_m, B'_{m+1} and it is prolonged by the suffix $B_{|w|+k+3} = 1\,\mathrm{code}(\#)0^{|Q|}$. Denote $x_1 = B_mB_{m+1}$, $x_2 = B'_mB'_{m+1}$ (codes of local changes performed by the applied instruction of $\mathcal{T}$, x_2 is being determined by x_1), $\ell = |\Gamma| + |Q| + 1$ (the length of each block B_i), $y = |w| + k + 1 - m$ (the number of blocks following B_{m+1} in $\mathcal{C}_k$), $c_\# = 1\,\mathrm{code}(\#)0^{|Q|}$ (the constant suffix of each $\mathcal{C}_i$). Then, $\mathcal{C}_{k+1}$ is expressed as

$$\mathcal{C}_{k+1} = (\mathcal{C}_k + (x_2 - x_1)2^{y\ell})2^\ell + c_\#.$$

We are ready to list productions that generate encodings of configurations. All nonterminals and productions from Example 1 are included in $\mathcal{G}$. Next, the following productions are defined.

$$C \to \overbrace{\begin{matrix} a & \cdots & a \\ \vdots & \ddots & \vdots \\ a & \cdots & a \end{matrix}}^{\mathcal{C}_0} \Bigg\} \mathcal{C}_0 \qquad (1) \qquad\qquad C \to \begin{matrix} C & J \\ A & R \end{matrix} \qquad (2)$$

$$J \to \begin{matrix} \overbrace{F \quad \cdots \quad F}^{2^\ell - 1} & A & \overbrace{V \quad \cdots \quad V}^{c_\#} \\ A \quad \cdots \quad A & X & V \quad \cdots \quad V \end{matrix} \qquad (3)$$

Production (1) ensures that C generates the square picture of size $\mathcal{C}_0$. Recall languages generated by A, V and R (see Example 1). Assume that X generates exactly all pictures of size $x'_1 2^{y'\ell} \times (x'_2 2^\ell - x'_1)2^{y'\ell}$ where the pair x'_1, x'_2 encodes a local change allowed by the instruction set of $\mathcal{T}$ and $y' \in \mathbb{N}$. Moreover, assume that F generates all square pictures of size $1B_1 \ldots B_{p-1}0^\ell 0^\ell B_{p+2} \ldots B_q$, where $q \geq 2$, $1 \leq p < q$ and each $B_i = 1\,\mathrm{code}(c_i)0^{|Q|}$ for some $c_i \in \Gamma$. This means $L(\mathcal{G}, F)$ consists of pictures whose sizes encode all configurations $\mathcal{C}_i$ with two neighboring blocks erased (filled by zeros) where one of the erased blocks encodes the head position and the state of the control unit in $\mathcal{C}_i$.

Let us examine which pictures generated by J can be substituted to the right-hand side of production (2) provided that the square picture $\mathcal{C}_k \times \mathcal{C}_k$ is substituted to C. The situation is illustrated in Fig. 6. The picture substituted for J is decomposed by the right-hand side of production (3) there. The number of rows enforces the equality $\mathcal{C}_k = v + x'_1 2^{y'\ell}$. Observe that it determines parameters x'_1, y' and v uniquely: Comparing to $\mathcal{C}_k$, value v, which is the size of a square picture in $L(\mathcal{G}, F)$, has two blocks erased and misses the representation of the state of $\mathcal{T}$ in $\mathcal{C}_k$. This needs to be supplied by adding $x'_1 2^{y'\ell}$. Since all blocks start by bit 1, there is only one possibility at which position to add x'_1, it holds $y' = y$ and v has the same number of bits as $\mathcal{C}_k$. Further, $\mathcal{C}_k$ determines if the head position is encoded in the first or second block of x'_1 ($\mathcal{T}$ is deterministic), hence x'_1 has to be equal to x_1. Value of parameter x'_2 is determined by x'_1, thus $x'_2 = x_2$. Moreover, it is also obvious that the non-erased blocks of v have to match the corresponding blocks in $\mathcal{C}_k$. Summing the counts of columns in Fig. 6 gives

$$\mathcal{C}_k + (2^\ell - 1)(\mathcal{C}_k - x_1 2^{y\ell}) + (x_2 2^\ell - x_1)2^{y\ell} + c_\# = \mathcal{C}_{k+1}.$$

The second row of the right-hand side of production (2) takes care for completing a square picture.

To finish the construction of $\mathcal{G}$, we have to list productions generating pictures in $L(\mathcal{G}, X)$ and $L(\mathcal{G}, F)$. This is mainly a technical issue. Let us start by productions related to X.

$$U \to a \qquad (4)$$

$$U \to \overbrace{\begin{matrix} U & \cdots & U \\ \vdots & \ddots & \vdots \\ U & \cdots & U \end{matrix}}^{2^\ell} \Bigg\} 2^\ell \qquad (5)$$

$$X \to \overbrace{\begin{matrix} U & \cdots & U \\ \vdots & \ddots & \vdots \\ U & \cdots & U \end{matrix}}^{x_2 2^\ell - x_1} \Bigg\} x_1 \qquad (6)$$

Production (6) is added for each possible pair x_1, x_2. Pictures in $L(\mathcal{G}, F)$ are generated in three phases. For $q \geq 2$, $1 \leq p < q$ and arbitrary blocks $B_i = 1\,\mathrm{code}(c_i)0^{|Q|}$, $c_i \in \Gamma$,

1. D generates square pictures of sizes $1B_1 \dots B_{p-1}$,
2. E generates square pictures of sizes $1B_1 \dots B_{p-1}0^\ell 0^\ell$, and
3. F generates square pictures of sizes $1B_1 \dots B_{p-1}0^\ell 0^\ell B_{p+2} \dots B_q$.

Productions related to D are as follows.

$$D \to a \qquad (7)$$

For each $i = 1, \dots, |\Gamma|$, add productions

$$D \to D_{i,\ell} \qquad (8)$$

$$D_{i,i+1} \to \begin{matrix} D_{i,i} & D_{i,i} & V \\ D_{i,i} & D_{i,i} & V \\ H & H & a \end{matrix} \qquad (9) \qquad\qquad D_{i,1} \to \begin{matrix} D & D & V \\ D & D & V \\ H & H & a \end{matrix} \qquad (10)$$

$$D_{i,j} \to \begin{matrix} D_{i,j-1} & D_{i,j-1} \\ D_{i,j-1} & D_{i,j-1} \end{matrix} \quad \text{for each } j \in \{2, \dots, \ell\} \setminus \{i+1\} \qquad (11)$$

Nonterminals H, V come from Example 1. $L(\mathcal{G}, D_{i,\ell})$ consists of square pictures of size $1B_1 \dots B_{p-2}\,\mathrm{code}(a_i)0^{|Q|}$. Production (9) or (10) can be interpreted as appending bit 1 to size of a square picture in $L(\mathcal{G}, D_{i,i})$ or $L(\mathcal{G}, D)$, respectively. Analogously, production (11) appends bit 0 to size of a square picture in $L(\mathcal{G}, D_{i,j-1})$. Productions related to E and F follow similar patterns.

$$E \to E_\ell \qquad (12)$$

$$E_i \to \begin{matrix} E_{i-1} & E_{i-1} \\ E_{i-1} & E_{i-1} \end{matrix} \quad \text{for each } i = 2, 3, \dots, \ell \qquad (13)$$

$$E_1 \to \begin{matrix} D & D \\ D & D \end{matrix} \qquad (14) \qquad\qquad F \to E \qquad (15)$$

For each $i = 1, \dots, |\Gamma|$, add productions

$$F \to F_{i,\ell} \qquad (16)$$

$$F_{i,i+1} \to \begin{matrix} F_{i,i} & F_{i,i} & V \\ F_{i,i} & F_{i,i} & V \\ H & H & a \end{matrix} \qquad (17) \qquad\qquad F_{i,1} \to \begin{matrix} F & F & V \\ F & F & V \\ H & H & a \end{matrix} \qquad (18)$$

$$F_{i,j} \to \begin{matrix} F_{i,j-1} & F_{i,j-1} \\ F_{i,j-1} & F_{i,j-1} \end{matrix} \quad \text{for each } j \in \{2, \dots, \ell\} \setminus \{i+1\} \qquad (19)$$

It remains to add productions generating the final configuration provided that $\mathcal{T}$ halts. We make clones of productions (2), (3) and (6). Production (20) will be applicable only to the configuration preceding the final configuration.

$$S_0 \to \begin{matrix} C & J_0 \\ A & R \end{matrix} \qquad (20)$$

$$J_0 \to \begin{matrix} \overbrace{F \quad \cdots \quad F}^{2^\ell - 1} & A & \overbrace{V \quad \cdots \quad V}^{c_\#} \\ A \quad \cdots \quad A & X_0 & V \quad \cdots \quad V \end{matrix} \qquad (21)$$

$$X_0 \to \left.\begin{matrix} \overbrace{U \quad \cdots \quad U}^{x_2 2^\ell - x_1} \\ \vdots \quad \ddots \quad \vdots \\ U \quad \cdots \quad U \end{matrix}\right\} x_1 \qquad (22)$$

Pattern (22) applies to those pairs x_1, x_2, where x_2 encodes a final state of $\mathcal{T}$. □

Theorem 9. *Transforming 2CFG to Kolam grammar yields a non-recursive trade-off.*

Proof. For $k \in \mathbb{N}$, construct a 2CFG $\mathcal{G}_k$ over $\{a, 0, 1\}$ as follows. There is the initial nonterminal S_0 and two distinguished nonterminals C_k and D. Productions and other nonterminals are added so that:

- D generates L_{cross},
- C_k generates all square pictures encoding a final configuration of some binary k-state Turing machine started over a blank tape (productions achieving this can be designed thanks to Proposition 8).

Finally, there is production $S_0 \to C_k\, D$.

Each picture language $L(\mathcal{G}_k)$ is finite, thus it is generated by a Kolam grammar. However, $L(\mathcal{G}_k)$ inherits the complexity of finite subsets of L_{cross} (Lemma 4 can be easily generalized to it), thus the size of the considered Kolam grammars is not bounded from above by any function recursive in k. □

For the purposes of our next theorem we need a suitable unary picture language which can be easily generated by a 2CFG, but cannot be recognized by any 4NFA. Let us consider the language of unary pictures from [12] where the number of columns is the square of the number of rows. Define $L_{\text{rect}} = \{P \in \{a\}^{*,*} \mid \text{cols}(P) = \text{rows}^2(P)\}$.

Lemma 10. *(See [12].) For $n \in \mathbb{N}$, let $\mathcal{A}$ be a 4NFA such that $L(\mathcal{A}) \cap \Sigma^{n,*} = L_{\text{rect}} \cap \Sigma^{n,*}$. Then, $c(\mathcal{A}) = \Omega(n)$.*

Theorem 11. *Transforming 2CFG generating finite unary picture languages to 4NFA yields a non-recursive trade-off.*

Proof. For $k \in \mathbb{N}$, construct a 2CFG grammar $\mathcal{G}_k$ over $\{a\}$, containing nonterminal C_k that generates all square pictures encoding a final configuration of some binary k-state Turing machine started over a blank tape (use Proposition 8). Let S_0 be the initial nonterminal of $\mathcal{G}_k$. Add additional productions and nonterminals so that $\mathcal{G}_k$ translates each square picture $P \in L(\mathcal{G}_k, C_k)$ to the rectangle of size $\text{rows}(P) \times \text{rows}^2(P)$.

- Include all nonterminals (H, V, A, R) and productions from Example 1.
- Insert nonterminal Y and productions

$$Y \to \begin{matrix} a & a \\ a & a \end{matrix}, \qquad Y \to \begin{matrix} Y & R & R \\ H & H & H \end{matrix}, \qquad S_0 \to C_k\ Y\ .$$

It is easy to verify (by induction on n) that Y generates every picture of size $n \times (n^2 - n)$ where $n \geq 2$. Thus, $L(\mathcal{G}_k, S_0)$ is formed of rectangular pictures of the desired size.

Finite unary picture languages $\{L(\mathcal{G}_k)\}_{k=1}^{\infty}$ are accepted by some 4NFAs, however, sizes of such 4NFAs are not bounded from above by any function recursive in k. □

Theorem 12. *Transforming 2CFG generating finite picture languages to 2OTA yields a non-recursive trade-off.*

Proof. Define L_{pal} as the picture language over $\{b, c\}$ consisting of pictures whose all rows are palindromes. L_{pal} can be generated by a 2CFG $\mathcal{G}$ with an initial nonterminal S (generate a column of nonterminals S, generate a one-row palindrome from each S). On the other hand, L_{pal} is not in REC and thus not accepted by any 2OTA. A counting argument based technique presented in [4] applies here.

Extend L_{pal} to the picture language L'_{pal} over $\{a, b, c\}$ by extending each P of size n to square P' of size $2n$ such that P is the top-left subpicture of P' and the remaining fields of P' store symbol a. The technique from [4] still proves that L'_{pal} is not in REC.

For $k \in \mathbb{N}$, let $\mathcal{G}_k$ over $\{a\}$ be again a 2CFG containing nonterminal C_k that generates all square pictures encoding a final configuration of some binary k-state Turing machine started over a blank tape.

Combine grammars $\mathcal{G}$ and $\mathcal{G}_k$ to obtain a 2CFG $\mathcal{G}'_k$ that generates finitely many square pictures over $\{a, b, c\}$ containing large pictures from L'_{pal}. Introduce a new initial nonterminal S_0, nonterminals S_1, S_2 and the following productions.

$$S_0 \to \begin{matrix} S_1 \\ S_2 \end{matrix}, \qquad S_1 \to \begin{matrix} S \\ C_k \end{matrix}, \qquad S_2 \to \begin{matrix} C_k \\ C_k \end{matrix}.$$

Let n_k be the size of the largest (square) picture in $L(\mathcal{G}_k)$. The picture language $L(\mathcal{G}'_k)$ is finite and it contains all square picture of size $2n_k$ from L'_{pal}. This means that 2OTA accepting $L(\mathcal{G}'_k)$ needs a large number of states. □

5. Conclusion

We have showed that the non-recursive trade-offs are a common phenomenon in the world of picture languages. We considered the most important models of two-dimensional finite automata as well as context-free grammars and demonstrated how the separability results on the classes of picture languages described by the systems can be turned into results on non-recursive trade-offs.

The ability of four-way automata and 2D context-free grammars to simulate Turing machines has been utilized. It allowed us to come up with constructive witnessing sequences of systems describing finite picture languages.

Besides the proven non-recursive trade-offs, the simulation of a Turing machine by a 2CFG brings additional consequences. It reveals that the emptiness problem is not decidable for the 2D context-free grammars as well as for the tile regional grammars. Moreover, 2CFG can generate one-element picture languages containing a square picture whose size can be arbitrarily (non-recursively) large with respect to the size of the grammar. This means that there is no general analogy to the pumping lemma which is counted as one of the basic properties of one-dimensional context-free grammars.

There remain some interesting open problems. The presented simulation of a Turing machine by a 2CFG benefited from productions with right-hand sides of size at least 2×2. It is not clear whether it can be performed using only productions of Kolam grammar. This raises the question of how succinct are Kolam grammars generating finite picture languages comparing to 4DFAs. It is not also known whether 2CFG is stronger than Kolam grammar for a unary alphabet. Showing that Kolam grammar cannot simulate Turing machine would resolve this problem positively.

Another open problem is whether Theorem 12 is valid for unary picture languages. We did not answer this since it is not know if there is a unary picture language generated by a 2CFG, but not accepted by any 2OTA.

Acknowledgement

This work was supported by the Czech Science Foundation under the project 15-04960S.

References

[1] M. Blum, C. Hewitt, Automata on a 2-dimensional tape, in: Proceedings of the 8th Annual Symposium on Switching and Automata Theory, SWAT 1967 (FOCS '67), IEEE Computer Society, Washington, DC, USA, 1967, pp. 155–160.
[2] K. Inoue, A. Nakamura, Some properties of two-dimensional on-line tessellation acceptors, Inform. Sci. 13 (2) (1977) 95–121.
[3] K. Inoue, I. Takanami, A characterization of recognizable picture languages, in: A. Nakamura, M. Nivat, A. Saoudi, P.S. Wang, K. Inoue (Eds.), Parallel Image Analysis, in: Lecture Notes in Computer Science, vol. 654, Springer, Berlin, Heidelberg, 1992, pp. 133–143.
[4] D. Giammarresi, A. Restivo, Recognizable picture languages, Int. J. Pattern Recognit. Artif. Intell. 6 (2–3) (1992) 32–45.
[5] G. Siromoney, R. Siromoney, K. Krithivasan, Abstract families of matrices and picture languages, Comput. Graph. Image Process. 1 (3) (1972) 284–307, http://dx.doi.org/10.1016/S0146-664X(72)80019-4.
[6] G. Siromoney, R. Siromoney, K. Krithivasan, Picture languages with array rewriting rules, Inf. Control 22 (5) (1973) 447–470, http://dx.doi.org/10.1016/S0019-9958(73)90573-1.
[7] O. Matz, Regular expressions and context-free grammars for picture languages, in: 14th Annual Symposium on Theoretical Aspects of Computer Science, Springer-Verlag, 1997, pp. 283–294.
[8] M.I. Schlesinger, Matematiceskie Sredstva Obrabotki Izobrazenij (Mathematic Tools for Image Processing), Naukova Dumka, Kiev, 1989, in Russian.
[9] M.I. Schlesinger, V. Hlaváč, Ten Lectures on Statistical and Structural Pattern Recognition (Computational Imaging and Vision), 1st edition, Springer, 2012.
[10] D. Průša, Two-dimensional languages, Ph.D. thesis, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic, 2004.
[11] M. Pradella, A. Cherubini, S.C. Reghizzi, A unifying approach to picture grammars, Inform. and Comput. 209 (9) (2011) 1246–1267, http://dx.doi.org/10.1016/j.ic.2011.07.001.
[12] J. Kari, C. Moore, New results on alternating and non-deterministic two-dimensional finite-state automata, in: A. Ferreira, H. Reichel (Eds.), STACS 2001, in: LNCS, vol. 2010, Springer, Berlin, Heidelberg, 2001, pp. 396–406.
[13] A.R. Meyer, M.J. Fischer, Economy of description by automata, grammars, and formal systems, in: SWAT (FOCS), IEEE Computer Society, 1971, pp. 188–191.
[14] L.G. Valiant, A note on the succinctness of descriptions of deterministic languages, Inf. Control 32 (2) (1976) 139–145, http://dx.doi.org/10.1016/S0019-9958(76)90173-X.
[15] C. Herzog, Pushdown automata with bounded nondeterminism and bounded ambiguity, Theoret. Comput. Sci. 181 (1) (1997) 141–157, http://dx.doi.org/10.1016/S0304-3975(96)00267-8.
[16] C.A. Kapoutsis, From $k+1$ to k heads the descriptive trade-off is non-recursive, in: L. Ilie, D. Wotschke (Eds.), 6th International Workshop on Descriptional Complexity of Formal Systems, vol. 619, The University of Western Ontario, Canada, 2004, pp. 213–224.

[17] M. Holzer, M. Kutrib, J. Reimann, Non-recursive trade-offs for deterministic restarting automata, J. Autom. Lang. Comb. 12 (1–2) (2007) 195–213.
[18] M. Kutrib, F. Otto, On the descriptional complexity of the window size for deleting restarting automata, Internat. J. Found. Comput. Sci. 24 (06) (2013) 831–846, http://dx.doi.org/10.1142/S0129054113400212.
[19] M. Kutrib, The phenomenon of non-recursive trade-offs, Internat. J. Found. Comput. Sci. 16 (05) (2005) 957–973.
[20] H. Gruber, M. Holzer, M. Kutrib, On measuring non-recursive trade-offs, J. Autom. Lang. Comb. 15 (1/2) (2010) 107–120.
[21] D. Giammarresi, A. Restivo, Two-dimensional languages, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, vol. 3, Springer, New York, 1997, pp. 215–267.
[22] J. Kari, C. Moore, Rectangles and squares recognized by two-dimensional automata, in: J. Karhumki, H.A. Maurer, G. Paun, G. Rozenberg (Eds.), Theory is Forever, in: Lecture Notes in Computer Science, vol. 3113, Springer, 2004, pp. 134–144.
[23] M.L. Minsky, Computation: Finite and Infinite Machines, Prentice Hall, Inc., 1967.
[24] T. Radó, On non-computable functions, Bell Syst. Tech. J. 41 (3) (1962) 877–884.
[25] A. Nakamura, Two-dimensional connected pictures are not recognizable by finite-state acceptors, Inform. Sci. 69 (1–2) (1993) 55–64, http://dx.doi.org/10.1016/0020-0255(93)90039-O.
[26] A. Ito, K. Inoue, A note on three-way two-dimensional alternating Turing machines, Inform. Sci. 45 (1) (1988) 1–22, http://dx.doi.org/10.1016/0020-0255(88)90005-9.
[27] G. Pighizzini, J. Shallit, M. Wang, Unary context-free grammars and pushdown automata, descriptional complexity and auxiliary space lower bounds, J. Comput. System Sci. 65 (2) (2002) 393–414, http://dx.doi.org/10.1006/jcss.2002.1855.

2017 14th IAPR International Conference on Document Analysis and Recognition

Rank-reducing Two-dimensional Grammars for Document Layout Analysis

Daniel Průša
Faculty of Electrical Engineering, Czech Technical University
Karlovo náměstí 13, 121 35 Prague 2, Czech Republic
Email: prusapa1@fel.cvut.cz

Akio Fujiyoshi
Faculty of Engineering, Ibaraki University
4-12-1 Nakanarusawa, Hitachi, Ibaraki, 316-8511, Japan
Email: akio.fujiyoshi.cs@vc.ibaraki.ac.jp

***Abstract*—We study the task of document layout analysis based on two-dimensional context-free grammars. We first identify a subclass of the grammars sufficient for a document structure description where productions follow a mechanism inducing regular languages in the case of one-dimensional productions. We then show that properties of such grammars can be conveniently utilized to implement a very fast top-down parser. Experimental results are reported for PDF documents, which are chosen as a test domain since we are motivated by a development of digital document access methods for people with disabilities in which a retrieval of structural information plays an important role.**

I. Introduction

Two-dimensional (2D) grammars are a powerful tool for document structure analysis. Variants with context-free productions have been studied in this context already from 60's. The grammars are still popular nowadays and serve as a base for recognition of many domains [1], [2], including the logical layout of documents [3]. The grammars naturally support nondeterminism and ambiguity whose absence is considered as a weakness of other methods for document layout analysis [4].

A challenge regarding the usage of 2D grammars is computational complexity. Especially in the case of complex structures like mathematical formulas, 2D topology and grammar ambiguity might result in parsing time exponential in the number of terminal elements, hence techniques limiting the set of hypotheses examined and tested are required for acceleration [5]. Parsing of document layouts is relatively easier scenario which can adopt classical bottom-up or top-down algorithms like CKY [6] or Earley parsing [7]. Their 2D counterparts have been described for regular grids of terminal elements [8], [9]. But still, time complexity of these algorithms is a high degree polynomial and they get further complicated if the layout of terminal elements is irregular. To deal with this, some authors proposed to parse linear projections of the input by 1D grammars [10], [11]. This approach results in a good performance, but has drawbacks such as a weaker expressive power caused by the simplified parsing or a complicated system of 1D productions not easy to be created by the user.

We contribute to the portfolio of 2D grammars by defining a subclass of the 2D context-free grammars where productions have a nonterminal rank reducing property. This property induces grammars generating regular languages in the case of 1D productions. We show that the grammar is sufficient for modeling document layouts. Next, we pursue the idea of using 1D grammars to speed up parsing. We apply a simple top-down, recursive descent parsing algorithm strengthen by strongly discriminative rules for finding branching points. These decision rules are based on 1D regular grammars describing sequences of terminal elements that can form borders of areas representing nonterminals. These 1D grammars are extracted fully automatically from the 2D grammar, together with non-deterministic finite-state automata used to recognize the sequences of terminals. We show that with this method the number of backtracks occurring during the parsing is minimal.

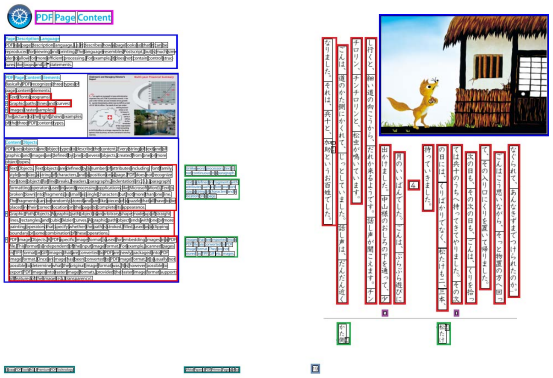


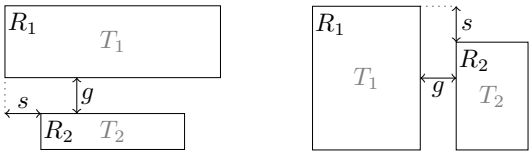
Fig. 1. Parsed PDF documents. Selected sections of interest are highlighted.

We evaluate our proposal on PDF documents (see Fig. 1). There is a practical motivation for this. Legislation has evolved in many countries to improve the access to documents for people with disabilities, hence the accessibility of digital documents is one of the biggest concerns for publishers in the world. To guarantee the accessibility for the blind and print disabled, the structural information of a document needed by screen readers must be embedded. There are mainly two ways to guarantee the accessibility of PDF, which is the most widely used format. One way is to directly embed structural information. The international standard for accessible PDF technology has been introduced as PDF/UA (ISO14289-1) [12]. The other way is to bundle an equivalent document in an accessible digital format (DAISY [13], EPUB3 [14]), converted from a PDF document. The necessity of the structural information of original PDFs is common to the two ways.

2379-2140/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDAR.2017.185

(a) (b)

Fig. 2. (a) Example of boxes extracted by Apache PDFBox. (b) A recognized word assembled from adjoining characters.



(a) (b)

Fig. 3. (a) A horizontal or (b) vertical production assigned by a spatial constraint $(g_{\min}, g_{\max}, s_{\min}, s_{\max})$ is applicable to divide a set of terminal elements $T = T_1 \cup T_2$ to T_1 and T_2 if and only if *gap* g and *shift* s fulfill $g_{\min} \leq g \leq g_{\max}$ and $s_{\min} \leq s \leq s_{\max}$, respectively. Rectangle R_1 and R_2 is the bounding box of terminal elements in T_1 and T_2, respectively.

```
Document -> Tests / PageNumber
PageNumber -> <number>
Tests -> Test / Tests
Tests -> Test
Test -> <image> / Text
Text -> Assignment /(-,-,50,-) Explanation
Assignment -> Question /(-,-,20,-) Answers
Question -> Rows
Answers -> Answer / Answers
Answers -> Answer
Answer -> <number> | Row
Explanation -> <image> | Rows
Rows -> Row /(-,-,-,8) Rows
Rows -> Row
Row -> <word> | Row
Row -> <word>
```

Fig. 4. A representation of a 2D grammar in a text file. Terminal names are enclosed in angle brackets, right-hand side variables of horizontal and vertical productions are separated by '/' and '|', respectively. These separators are optionally followed by four values in brackets determining the spatial constraint which overrides the default constraint $(0, -, -, -)$. Each component of this quadruple is either an integer or $-$ which stands for "no limit". The initial nonterminal appears at the left-hand side of the first production.

II. Proposed Method

The whole method is presented in this section, structured as follows. The process of terminal elements extraction from a PDF is described in Subsection II-A. A general 2D context-free grammar with a specific spatial constraints is defined in Subsection II-B. The top-down parsing algorithm building a derivation tree over a set of terminal elements is stated in Subsection II-C. The special rank-reducing 2D grammar is defined in Subsection II-D. Finally, it is shown in Subsection II-E how to utilize this grammar to implement a fast and highly discriminative procedure to select branching points for the top-down parsing algorithm.

A. Terminal Elements Extraction

We use Apache PDFBox [15] to extract content from PDF documents. Fig. 2(a) shows two types of bounding boxes that can be obtained for characters. The red ones represent the reserved area to render glyphs, the blue ones represent the exact area of the rendered glyphs. We choose the blue boxes to recognize *words*. We define a word as a group of adjoining characters of the same font and size (see Fig. 2(b) and Fig. 1). Neighboring characters are grouped based on the gap between their boxes and the "drawing order" of elements. If there are n characters, words are recognized in time $\mathcal{O}(n \log n)$ as we need to sort the boxes by their coordinates. For each recognized word, we create a *terminal element* consisting of a bounding box and a terminal symbol derived from the word content. We distinguish terminal symbols like a number, bullet, text word, etc. A terminal element is also created for each detected image. The used length unit equals 1 millimeter. The origin $(0, 0)$ is associated with the top-left corner of a PDF page and the coordinates grow rightwards and downwards.

B. Two-dimensional Context-free Grammar

Definition 1. *A* 2D context-free grammar *(*2CFG*) is a tuple* $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \mathcal{P}, S)$*, where* $\mathcal{N}$ *is a set of nonterminals,* $\mathcal{T}$ *is a set of terminals,* $S \in \mathcal{N}$ *is the initial nonterminal and* $\mathcal{P}$ *is a set of productions in one of the following forms:*

$$(1)\ N \to A, \quad (2)\ N \to A\,B, \quad (3)\ N \to \begin{matrix} A \\ B \end{matrix}$$

where $N \in \mathcal{N}$ *and* $A, B \in \mathcal{T} \cup \mathcal{N}$*.*

The productions follow the Chomsky normal form. We say that a production of type (2) is *horizontal* as it generates a horizontal line composed of A and B. Analogously, a production of type (3) is *vertical*.

Let T be a set of terminal elements. The goal of the structural analysis controlled by $\mathcal{G}$ is to build a derivation tree, also known as X-Y tree [4], over elements of T. Speaking in words of the top-down parsing, the whole set T is firstly assigned by the initial nonterminal S. A production $S \to A\,B$ or $S \to \begin{matrix} A \\ B \end{matrix}$ is applied to divide the set vertically or horizontally, respectively, into two parts, assigned by variables A and B. Alternatively, a production $S \to A$ is applied to rename S to A where renaming to a terminal $A \in \mathcal{T}$ is allowed only if S is assigned to a one-element set containing a terminal element of type A. The process continues recursively. An application of productions which ends up with one-element groups assigned by terminals is sought.

To increase the expressive power of the grammar, we restrict applicability of a vertical or horizontal production by a spatial constrained defined as a tuple $(g_{\min}, g_{\max}, s_{\min}, s_{\max})$. The constraint is explained in Fig. 3.

We illustrate the described notions by giving a grammar in Fig. 4 and a derivation tree built by the grammar in Fig. 5.

C. Top-down Parsing Algorithm

An execution of the top-down parsing is captured by Alg. 1. The first call of PARSE passes to it a grammar

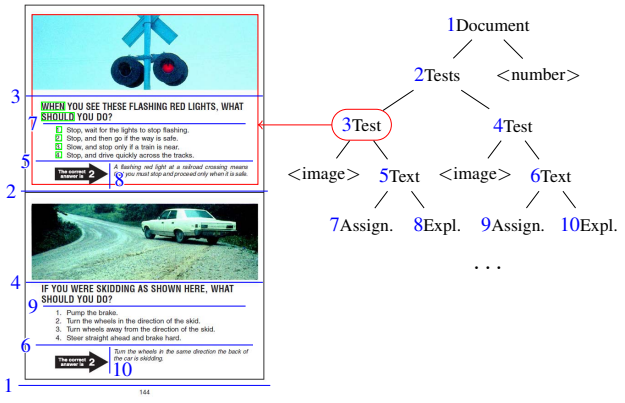


Fig. 5. The first 5 levels of a derivation tree built over a PDF page. Each Inner node is assigned by a nonterminal (black text) and a vertical or horizontal cut line (blue number, blue line). For simplicity, usages of productions of the form $N \to A$ are omitted. A part of the document represented by the subtree rooted in node '3Test' is highlighted in red. The leftmost terminal elements of the top 'Assignment' area (node '7Assign.') are highlighted in green.

$\mathcal{G} = (\mathcal{N}, \mathcal{T}, \mathcal{P}, S)$, the initial nonterminal S and the set of terminal elements 'Terms'. A global variable 'backtracks' is used to count the number of encountered dead branches. Procedure MATCHES(Terms, V) returns true iff the passed set of terminal elements 'Terms' contains the only terminal element assigned by V. Procedure FINDSPLITPOINTS(Terms, P) detects all vertical/horizontal cuts compatible with the spatial constraint of production P. Given a vertical/horizontal cut (determined by a point s), SPLIT(Terms, s, P) divides terminal elements of 'Terms' into two resulting subsets. For the pseudocode simplicity, PARSE returns only a boolean indicating whether the process succeeded (not a tree).

Algorithm 1 Top-down parsing

Input: $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \mathcal{P}, S)$, $V \in \mathcal{N} \cup \mathcal{T}$, Terms

```
1: procedure PARSE(G, V, Terms)
2:     if V ∈ T and MATCHES(Terms, V) then
3:         return true
4:     for each P ∈ P do
5:         if P = V → A and PARSE(G, A, Terms) then
6:             return true
7:         else if P = V → A B or P = V → [A; B] then
8:             SplitPoints = FINDSPLITPOINTS(Terms, P)
9:             for each s ∈ SplitPoints do
10:                [TermsA, TermsB] = SPLIT(Terms, s, P)
11:                if PARSE(G, A, TermsA) and PARSE(G, B,
   TermsB) then
12:                    return true
13:    backtracks ← backtracks + 1
14:    return false
```

D. Rank-reducing 2D Context-free Grammar

Let $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \mathcal{P}, S)$ be a 2CFG. Considering productions of $\mathcal{G}$ without their spatial constraints, we can construct 1D grammars $\mathcal{G}_\mathrm{T}$ and $\mathcal{G}_\mathrm{L}$, generating strings over $\mathcal{T}$, that characterize sequences of terminal elements appearing at the top and left border, respectively, in documents described by $\mathcal{G}$.

$\mathcal{G}_\mathrm{T} = (\mathcal{N}, \mathcal{T}, \mathcal{P}_\mathrm{T}, S)$ where $\mathcal{P}_\mathrm{T}$ contains each $P \in \mathcal{P}$ of the form $N \to A$ and $N \to A\,B$. In addition, instead of each production $N \to \begin{matrix} A \\ B \end{matrix}$, it contains $N \to A$ (B on the right-hand side is deleted, it cannot generate a part of the top border). Analogously, $\mathcal{G}_\mathrm{L} = (\mathcal{N}, \mathcal{T}, \mathcal{P}_\mathrm{L}, S)$ where $\mathcal{P}_\mathrm{L}$ contains all productions in $\mathcal{P}$ of the form $N \to A$ and $N \to \begin{matrix} A \\ B \end{matrix}$. In addition, instead of each $N \to A\,B$, it contains $N \to A$.

For $\mathcal{G}' \in \{\mathcal{G}_\mathrm{T}, \mathcal{G}_\mathrm{L}\}$, let $L(\mathcal{G}', N)$ be the 1D language generated by G' when $N \in \mathcal{N}$ is taken as the initial nonterminal.

Example 1. Let $\mathcal{G}$ be the grammar in Fig. 4. Then, $L(\mathcal{G}_\mathrm{L}, \texttt{Assignment})$ generates columns whose content is given by the regular expression `<word>+<number>+`, hence the sequence of left border terminal elements of each Assignment area have to match this pattern (see Fig. 5 for an example). This is utilized in the improved version of procedure FINDSPLITPOINTS presented in Subsection II-E.

Next, we formulate general conditions under which $\mathcal{G}_\mathrm{T}$ and $\mathcal{G}_\mathrm{L}$ generate regular languages, which are very easy to parse.

Definition 2. *Let $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \mathcal{P}, S)$ be a* 2CFG *grammar. A function $\sigma : \mathcal{N} \cup \mathcal{T} \to \mathbb{N}$ is called a* rank function *of $\mathcal{G}$ if it fulfills $\forall a \in \mathcal{T} : \sigma(a) = 0$ and $\forall N \in \mathcal{N} : \sigma(N) > 0$. For $V \in \mathcal{N} \cup \mathcal{T}$, $\sigma(V)$ is called a* rank *of V.*

Definition 3. *We say that a* 2CFG *$\mathcal{G} = (\mathcal{N}, \mathcal{T}, \mathcal{P}, S)$ is* rank-reducing *iff there is a rank function $\sigma : \mathcal{N} \cup \mathcal{T} \to \mathbb{N}$ such that*

1) *$\sigma(N) \geq \sigma(A)$ for each production $N \to A \in \mathcal{P}$,*
2) *$\sigma(N) \geq \sigma(B) > \sigma(A)$ for each production $N \to A\,B \in \mathcal{P}$ or $N \to \begin{matrix} A \\ B \end{matrix} \in \mathcal{P}$.*

In this case, σ is called a rank-reducing function *of $\mathcal{G}$.*

The next definition says which productions can be used in derivations applied recursively to a nonterminal N.

Definition 4. *Let $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \mathcal{P}, S)$ be a* 2CFG. *For each $N \in \mathcal{N}$, define $\mathrm{pcl}(N) \subseteq \mathcal{P}$ as the smallest subset fulfilling*

- *$P = N \to A \in \mathcal{P} \Rightarrow P \in \mathrm{pcl}(N)$ and $\mathrm{pcl}(A) \subseteq \mathrm{pcl}(N)$,*
- *$P = N \to A\,B$ or $P = N \to \begin{matrix} A \\ B \end{matrix}$ is in $\mathcal{P} \Rightarrow P \in \mathrm{pcl}(N)$, $\mathrm{pcl}(A) \subseteq \mathrm{pcl}(N)$ and $\mathrm{pcl}(B) \subseteq \mathrm{pcl}(N)$.*

Lemma 1. *Let $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \mathcal{P}, S)$ be a rank-reducing* 2CFG *and σ be a rank-reducing function of $\mathcal{G}$. Then, for each $N \in \mathcal{N}$, each production in $\mathrm{pcl}(N)$ contains nonterminals whose rank is not greater than $\sigma(N)$.*

Proof. Follows directly from Definition 3. □

Theorem 2. *Let $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \mathcal{P}, S)$ be a rank-reducing* 2CFG. *Then, $\mathcal{G}_T$ and $\mathcal{G}_L$ generate regular languages.*

Proof. Let σ be a rank-reducing function of $\mathcal{G}$. We show $L(\mathcal{G}_\mathrm{T}, N)$ is a regular language by induction on the rank of

nonterminal N (the proof for $\mathcal{G}_{\mathrm{L}}$ is analogous). Observe that σ is a rank-reducing function of $\mathcal{G}_{\mathrm{T}}$ too. For $n \in \mathbb{N}$, let $\mathcal{R}_n = \{V \in \mathcal{N} \cup \mathcal{T} \mid \sigma(V) = n\}$. Note that $\mathcal{R}_0 = \mathcal{T}$. Let n_1 be the smallest positive integer such that $\mathcal{R}_{n_1} \neq \emptyset$. Consider any $N \in \mathcal{R}_{n_1}$. If $\mathrm{pcl}(N)$ contains a production $M \to \begin{matrix} A \\ B \end{matrix}$ or $M \to A\,B$, then, by Lemma 1, $\sigma(B) \leq \sigma(N)$, and, by Definition 3, $\sigma(A) < \sigma(B)$, hence the production is regular and $L(\mathcal{G}_{\mathrm{T}}, N)$ is a regular language.

Next, consider $n > n_1$ for which $\mathcal{R}_n \neq \emptyset$. Denote $\mathcal{S}_n = \bigcup_{i=0}^{n-1} \mathcal{R}_n$. For each $N \in \mathcal{R}_n$, productions of $\mathrm{pcl}(N)$ contain only symbols of $\mathcal{R}_n$ and $\mathcal{S}_n$. In analogy to the previous paragraph, N generates a regular language over $\mathcal{S}_n$. By the induction hypothesis, each nonterminal in $\mathcal{S}_n$ generates a regular language over $\mathcal{T}$. The language generated by N over $\mathcal{T}$ is obtained by the operation of regular substitution (each nonterminal $M \in \mathcal{S}_n$ is substituted by the regular language $L(\mathcal{G}_{\mathrm{T}}, M)$). Since the regular languages are closed under the regular substitution [16], $L(\mathcal{G}_{\mathrm{T}}, N)$ is a regular language. □

Theorem 3. *There is an algorithm deciding in time $\mathcal{O}(|\mathcal{N}| + |\mathcal{P}|)$ whether a* 2CFG $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \mathcal{P}, S)$ *is rank-reducing.*

Proof. Given a 2CFG $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \mathcal{P}, S)$, construct a directed *dependency* graph $G = (V, E)$ where the set of vertices V equals $\mathcal{N}$ and (A, B) is a directed edge iff $A \neq B$ and $\mathcal{P}$ contains a production $A \to C$, $A \to C\,D$ or $A \to \begin{matrix} C \\ D \end{matrix}$ such that $C = B$ or $D = B$. Distinguish two types of edges. An edge (A, B) is *strictly reducing* if there is a production $A \to B\,D$ or $A \to \begin{matrix} B \\ D \end{matrix}$, otherwise it is *weakly reducing*.

Apply Tarjan's algorithm [17] to find strongly connected components of graph G. Grammar $\mathcal{G}$ is rank-reducing iff each strictly reducing edge connects vertices from different components. Indeed, if both vertices of a strictly reducing edge are in the same strongly connected component, the edge is a part of a cycle and we cannot define σ for vertices along the cycle as the descendants are expected to have smaller rank than their parents. Otherwise, we can construct a rank-reducing function σ of $\mathcal{G}$. Let $C_1, \ldots, C_m$ be the detected components written topologically ordered (note that Tarjan's algorithm produces such ordering). Define $\sigma : \mathcal{N} \cup \mathcal{T} \to \{0, \ldots, m\}$ as a function assigning to each nonterminal $N \in C_i$ value $m - i + 1$. Moreover, define $\sigma(a) = 0$ for all $a \in \mathcal{T}$.

Time complexity of Tarjan's algorithm is $\mathcal{O}(|V| + |E|)$. It holds $|E| \leq |\mathcal{P}|$, hence the proposed algorithm runs in time $\mathcal{O}(|\mathcal{N}| + |\mathcal{P}|)$. □

Example 2. Let $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \mathcal{P}, S)$ be a 2CFG where $\mathcal{N} = \{B, H, L, M, N, P, S\}$, $\mathcal{T} = \{h, w\}$ and $\mathcal{P}$ consists of

$$S \to B, \quad S \to B\,S, \quad S \to \begin{matrix} B \\ S \end{matrix},$$
$$B \to \begin{matrix} H \\ P \end{matrix}, \quad P \to L, \quad P \to \begin{matrix} L \\ P \end{matrix},$$
$$H \to h\,M, \quad M \to h\,N, \quad M \to h, \quad N \to L\,H,$$
$$L \to w, \quad L \to w\,L.$$

$\mathcal{G}$ generates documents S formed of blocks B. Each block is composed of a header H and a body P. The header format is given by the regular expression $hh\,(w^{+}hh)^{*}$, the body is filled by w's. Fig. 6 depicts the dependency graph for $\mathcal{G}$.

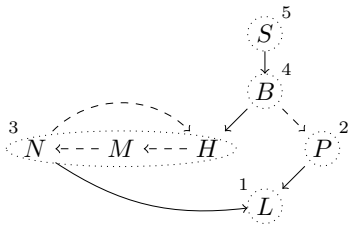


Fig. 6. An example of the dependency graph. The solid edges are strictly reducing, while the dashed edges are weakly reducing. The dotted circles/ellipse denote the strongly connected components. As there are no strongly reducing edges inside the components, the grammar is rank-reducing. The rank of each nonterminal can be defined as the number assigned to its component.

Remark 1. The grammar in Fig. 4 is rank-reducing. We verified for more documents that a usage of rank-reducing productions is sufficient for modeling their structure. Usually, a general 2CFG expressing a document layout can be transformed to a rank-reducing grammar by using more nonterminals. For example, production

```
Document -> Tests / <number>
```

cannot be included in a rank-reducing grammar, however, this problem is solved by introducing a nonterminal `PageNumber` and replacing the production above by productions

```
Document -> Tests / PageNumber
PageNumber -> <number>
```

E. Finding Split Point Candidates

Here we describe an efficient implementation of procedure FINDSPLITPOINTS in Alg. 1. It is based on spatial constraints of productions and sequences of leftmost or topmost terminal elements in 'Terms' described by the regular language generated by $\mathcal{G}_{\mathrm{L}}$ or $\mathcal{G}_{\mathrm{T}}$, respectively.

Let 'Terms' consist of terminal elements $R_i = (x_i, y_i, w_i, h_i, t_i)$, $i = 1, \ldots, n$, where (x_i, y_i), w_i, h_i and t_i is the top-left corner coordinate, width, height and the assigned terminal symbol of R_i, respectively. Without loss of generality, consider that FINDSPLITPOINTS has to detect split point candidates for a vertical production $P = N \to \begin{matrix} A \\ B \end{matrix}$. For each $i = 1, \ldots, n$, define interval $I_i = [y_i, y_i + h_i - 1]$.

As the first step, the procedure calculates the union of projections of elements R_i to y-axis. The result is a set of intervals depicted in Fig. 7(a). Moreover, it also detects all leftmost elements. Formally, an element R_i is *leftmost* iff for each $j \neq i$, either I_i and I_j are disjoint or $x_i < x_j$. For an example, see Fig. 7(b). To achieve time complexity $\mathcal{O}(n \log n)$, elements R_i are sorted by component x_i in ascending order. Assume that $R_1, \ldots, R_n$ is a sorted sequence. Taking the elements in this order, they are inserted to a red-black tree T with key values I_i. Let R_i be an element that is about to be inserted

into T. If I_i does not intersect any interval (key) in T, R_i is a leftmost element and it is inserted to T. Otherwise, R_i is not a leftmost element and the intervals intersecting I_i are removed from T, united with I_i into a new interval I_i' which is inserted to T.

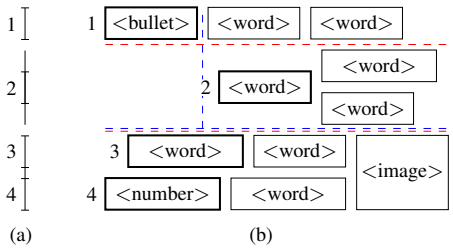


Fig. 7. (a) A projection of terminal elements (b) to y-axis. The leftmost elements are highlighted and numbered. Their projections are numbered too. The leftmost element with number 2 need not be a part of a column generated by $\mathcal{G}_\mathrm{L}$ as it becomes hidden if the blue dashed cuts are applied to build a derivation tree. On the other hand, it is a part of a column generated by $\mathcal{G}_\mathrm{L}$ if the red dashed cuts are applied. As there are these two possibilities, the leftmost element with number 2 is optional. This is not the case of the leftmost elements with numbers 1, 3 and 4, which are not optional.

Let $L_1, \ldots, L_m$ be a sequence of the leftmost terminal elements, sorted by their y-coordinate. Denote by s_i the terminal symbol assigned to L_i. In the projection in Fig. 7(a), projections of two neighboring elements L_j and L_{j+1} belong either to adjacent line segments separated by a gap of size g_j, or to one line segment (define $g_j = 0$). A leftmost element L_j is *optional* iff it can be separated from another leftmost element by a vertical cut as it is explained in Fig. 7(b).

For a non-terminal M, let $\mathcal{A}(M)$ denote a nondeterministic finite-state automaton (NFA) accepting the regular language $L(\mathcal{G}, M)$. We construct such automaton for each $M \in \mathcal{N}$. The construction is incremental and follows the constructive proof of Theorem 2. It starts with nonterminals of rank 1 and proceeds to nonterminals of higher ranks. We denote by $\mathcal{A}^R(M)$ an NFA which accepts the reversion of language $L(\mathcal{G}, M)$. This automaton is easily derived from $\mathcal{A}(M)$.

A position $j \in \{1, \ldots, m-1\}$ corresponds to a split point candidate returned by procedure FINDSPLITPOINTS for production P and set of terminal elements 'Terms' iff

- $g_j > 0$ and the subsets 'TermsA' and 'TermsB' obtained after cutting 'Terms' by a horizontal line separating L_j and L_{j+1} fulfill the spatial constraint of P (to relax $g_j > 0$ requirement, it is advisable to shrink the border of R_i's by a constant size before computing the projections).
- $\mathcal{A}(A)$ accepts a string obtained from $s_1 \ldots s_j$ by deleting an arbitrary number of terminals s_k such that L_k is optional. And similarly, $\mathcal{A}^R(B)$ accepts a string obtained from $s_m \ldots s_{j+1}$ by deleting some (or none) optional terminals.

NFAs are simulated by a technique which represents transitions and sets of active states by bit vectors [18]. It applies bitwise operations to the vectors to compute the next set of active states. The nondeterminism is favorably used to implement a skipping of optional leftmost elements. When the automaton scans s_k where L_k is optional, it simultaneously performs two actions: (i) ignores (skips) s_k, (ii) reads s_k and performs transitions. To find all positions j fulfilling the second requirement, it suffices to simulate $\mathcal{A}(A)$ and $\mathcal{A}^R(B)$ over $w = s_1 \ldots s_m$ only once. If J_1 and J_2 is the set of positions in w at which $\mathcal{A}(A)$ and $\mathcal{A}^R(B)$ reaches an accepting state when reading w from left to right and from right to left, respectively, then the test is passed by each j such that $j \in J_1$ and $j+1 \in J_2$.

III. EXPERIMENTS

The algorithms in Section II were implemented in Java 1.8. The experiments reported here were carried out on a notebook with Intel Core i5-4300M 2.6 GHz, 12 GB RAM and 64-bit Windows 7. Our application can be downloaded at [19].

We have created rank-reducing grammars for four types of documents: (i) Driving licence test [20], (ii) Whitepaper "PDF Primer" [21], (iii) Japanese elementary school textbook [22], (iv) "Answers to problems" in a book on chemistry [23]. We have excluded pages whose layout differs from the main layout (e.g. a table of contents) or we selected only pages from one section (dataset (iv)). PDF pages in Fig. 1 are samples from (i) and (iii), the page in Fig. 5 is a sample from (ii).

Table I summarizes statistics on grammars and automata sizes and time of automata creation. Note that four automata are created for each nonterminal N ($\mathcal{A}(N)$ and $\mathcal{A}^R(N)$ for $\mathcal{G}_\mathrm{L}$ as well as for $\mathcal{G}_\mathrm{T}$). We can observe that the average number of states per automaton is small (not greater than 16 when the grammar has about 35 productions and 22 nonterminals).

Table II reports performance of terminal elements extraction and two parsing algorithms. The first algorithm ('parsing I') utilizes spatial constraints and finite automata to select split point candidates (Subsection II-E), while the second algorithm ('parsing II'), applied to the same grammars, utilizes only spatial constraints of productions. It does not need to spend time to simulate automata, but the number dead branches it encounters is higher. Column 'pages' gives the number of pages per each dataset. The other columns give an average value per a page. The average time of parsing is followed by the average number of backtracks performed during it. We can see that the automata-based parsing is 3 to 4 times faster than the simple parsing. The number of backtracks is kept small, with the exception of dataset (iv), however, it is still substantially smaller when compared to the simple parsing. The difference between the two parsing algorithms is even more amplified in the case of not successfully parsed pages (due to an unexpected layout). For example, grammar (ii) does not support figure captions. A page of this type was rejected by 'parsing I' in 31 [ms] and 552 backtracks occurred. In contrast, 'parsing II' rejected the page in 917 [ms] as it was forced to perform 254, 324 backtracks. A similar behavior was observed in more situations when a page had to be rejected. This suggests that the automata-based parsing is a suitable choice for a document layout classification done via a successive parsing of the document by a collection of grammars.

Document	productions	automata	avg. states	max. states	build time [ms]
(i) Test	18	52	4.4	16	3
(ii) Whitepaper	35	92	10.5	57	17
(iii) School book	34	88	10.6	53	14
(iv) Chem. book	31	88	15.4	77	15

TABLE I
STATISTICS ON A GRAMMAR AND FINITE AUTOMATA CREATED FOR EACH DATASET.

Document	pages	term. elements	extraction [ms]	parsing I [ms]	backtracks	parsing II [ms]	backtracks
(i) Test	12	115.8	3.9	**11.5**	**23.3**	49.0	4191.1
(ii) Whitepaper	7	409.1	16.3	**20.1**	**13.7**	76.1	7690.7
(iii) School book	10	127.4	3.9	**7.6**	**16.8**	19.9	1032.5
(iv) Chem. book	4	698.5	23.5	**24.5**	**235.5**	86.5	3669.5

TABLE II
PERFORMANCE OF TERMINAL ELEMENTS EXTRACTION, AUTOMATA-BASED PARSING AND SIMPLE PARSING.

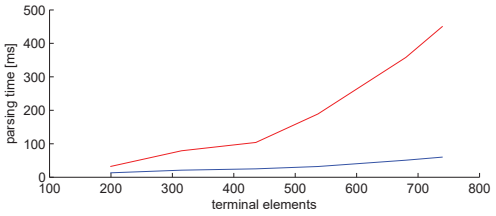


Fig. 8. Time of 'parsing I' (in blue) and 'parsing II' (in red) in dependence on the number of terminal elements.

Fig. 8 shows how parsing time depends on the number of terminal elements. To plot the graph, we created own PDF pages of type (ii) with varying number of characters. We see that 'parsing I' scales much more better than 'parsing II'.

IV. CONCLUSION

We have established a subclass of 2D context-free grammars and verified that the grammar is sufficiently expressive towards logical layouts of documents. We have proposed a top-down parsing algorithm over freely-spaced terminal elements. Although we have not presented any polynomial time guarantees on its time complexity, we have demonstrated that it is very stable and efficient in practice as it benefits from strongly discriminative rules for finding branching points, implemented via nondeterministic finite-state automata. As a future work, the grammar can be evaluated in a context of other domains or extended to a stochastic version to suit needs of noisy inputs.

ACKNOWLEDGMENTS

D. Průša was supported by the Czech Science Foundation under grant number 16-05872S. A. Fujiyoshi was supported by JSPS KAKENHI Grant Number JP26282044.

REFERENCES

[1] A. Lemaitre, H. Mouchère, J. Camillerapp, and B. Coüasnon, "Interest of syntactic knowledge for on-line flowchart recognition," in *9th IAPR International Workshop on Graphics Recognition, 2011. GREC 2011*, 2011, pp. 85–88.

[2] F. Álvaro, J.-A. Sánchez, and J.-M. Benedí, "Recognition of on-line handwritten mathematical expressions using 2D stochastic context-free grammars and hidden Markov models," *Pattern Recognition Letters*, vol. 35, no. 0, pp. 58 – 67, 2014, frontiers in Handwriting Processing.

[3] F. Álvaro, F. Cruz, J.-A. Sánchez, O. Ramos Terrades, and J.-M. Benedí, "Structure detection and segmentation of documents using 2D stochastic context-free grammars," *Neurocomputing*, vol. 150, no. PA, pp. 147–154, Feb. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.neucom.2014.08.076

[4] S. Mao, A. Rosenfeld, and T. Kanungo, "Document structure analysis algorithms: a literature survey," in *Proc. SPIE*, vol. 5010, 2003, pp. 197–207. [Online]. Available: http://dx.doi.org/10.1117/12.476326

[5] P. Liang, M. Narasimhan, M. Shilman, and P. Viola, "Efficient geometric algorithms for parsing in two dimensions," *International Conference on Document Analysis and Recognition*, pp. 1172–1177, 2005.

[6] D. Younger, "Recognition of context-free languages in time n^3," *Information and Control*, vol. 10, pp. 189–208, 1967.

[7] J. Earley, "An efficient context-free parsing algorithm," *Commun. ACM*, vol. 13, no. 2, pp. 94–102, Feb. 1970. [Online]. Available: http://doi.acm.org/10.1145/362007.362035

[8] M. Schlesinger and V. Hlaváč, *Ten lectures on statistical and structural pattern recognition*, ser. Computational Imaging and Vision. Dordrecht, The Netherlands: Kluwer Academic Publishers, 2002, vol. 24.

[9] M. Tomita, *Parsing 2-Dimensional Language*. Boston, MA: Springer US, 1991, pp. 277–289. [Online]. Available: https://doi.org/10.1007/978-1-4615-3986-5_18

[10] M. Krishnamoorthy, G. Nagy, S. Seth, and M. Viswanathan, "Syntactic segmentation and labeling of digitized pages from technical journals," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 7, pp. 737–747, Jul 1993.

[11] T. A. Tokuyasu and P. A. Chou, "Turbo recognition: a statistical approach to layout analysis," in *Proceedings of SPIE Conference on Document Recognition and Retrieval*, San Jose, CA, USA, Jan 2001.

[12] "PDF/UA Competence Center," https://www.pdfa.org/working-group/pdfua-competence-center/.

[13] "DAISY consortium," http://www.daisy.org/.

[14] "EPUB," http://idpf.org/epub/.

[15] "PDFBox," https://pdfbox.apache.org/.

[16] A. Meduna, *Automata and Languages: Theory and Applications*. Springer Verlag, 2005.

[17] R. Tarjan, "Depth first search and linear graph algorithms," *SIAM Journal on Computing*, vol. 1, no. 2, 1972.

[18] J. Holub and B. Melichar, "Implementation of nondeterministic finite automata for approximate pattern matching," in *Revised Papers from the Third International Workshop on Automata Implementation*, ser. WIA '98. London, UK: Springer-Verlag, 1999, pp. 92–99.

[19] "PDF parser web page," http://cmp.felk.cvut.cz/~prusapa1/pdf-parser.html.

[20] "Driving licence test, State of Hawaii, Department of Transportation," http://hidot.hawaii.gov/highways/files/2013/01/mvso-Practice-Test-Q-A-Part-1.pdf.

[21] "PDF Primer - Whitepaper, PDF-tools.com," https://www.pdf-tools.com/public/downloads/whitepapers/Whitepaper-PDF-Primer-EN.pdf.

[22] *Elementary school textbooks in Japanese language*. Mitsumura Tosho Publishing Co., Ltd., 2015.

[23] "T. Poulsen: Introduction to Chemistry, CK-12 Foundation," http://openedgroup.org/books/Chemistry.pdf.