Tabular Constraint Learning

Name1 Surname1 and **Name2 Surname2** and **Name3 Surname3** [1]

Abstract. abstract

1 Introduction

SERGEY: bullet points for luc to start introduction

Key question:

Can we discover or reconstruct structural relations in flat tabular spreadsheet data? [in a general way that allows declarative specification of constraints to discover]

Motivation:

- File generated from model, model got lost, need to reconstruct
- Constraint programming is hard - is Excel hard?
- Avoid manual analysis, provide selection of constraints
- Error checking
- Completion, gain speed and insights (Complicated constraints, also complicated to verify, too much output)

Novelty:

- Unsupervised setting (contrary to flashfill, etc)
- Numeric, different constraints (contrary to single textual function solution in flashfill, etc)
- Data format (2D) – data is no longer in rows like a classic ML or DM settings
- Declarative, general / modular, stacking of constraint problems

SERGEY: need to elaborate the example here: like in a story

2 Formalization

Algorithm 1 Tabular constraint learning

Input: G – set of groups
Output: S – learned constraints with their satisfaction assignments
$S \leftarrow \emptyset$ ▷ The set of solutions
for $c \in \mathcal{C}$ **do** ▷ $\mathcal{C}$– partially ordered set of Excel constraints; Table 1
  $v_1, ..., v_n$ = variables of c
  for $v_1\colon G_1, \ldots, v_n\colon G_n \in$ *generateAssignments*(c, G, S) **do**
    $S \leftarrow S \cup$ *findSolutions*$(c, v_1\colon G_1, \ldots, v_n\colon G_n, S)$
return S

2.1 Groups, Type-consistency and Constraints

In this work we distinguish the following types of data: numeric and textual. The numeric type has two subtypes: integers and floats. We also consider the special element called *None*, which has two types: numeric and textual. A set is called *type-consistent* iff all elements are either numeric or textual. Certain constraints, such as *rank* or *series*, make use of the numeric sub-types by requiring its arguments to be integers.

A *vector* is either a column or a row that is type-consistent. If a vector is a row (column), we say that it has a *row* (*column*) orientation. A *group* is a subrange of vectors with the same orientation in a table. We use the following notation to refer to a row (column) group G in a table T with rows (columns) ranging from a to b: $G = T[a{:}b, :]$ ($G = T[:, a{:}b]$), where a, b are natural numbers. We denote as the *length* of a row (column) group G, written as *length(G)*, the number of its columns (rows). We call a group G *numeric* (*textual*, etc), written as *numeric(G)*, if its vectors contain numeric (textual, etc) elements.

An *Excel constraint* is a triple *(Name, Signature, Function)*. Let us elaborate on each of them. *Name* is the textual name of the constraint together with its variable names. *Signature* is a set of constraints specifying the properties of the group assignments, corresponding to the variables in Name, such as their types, e.g., requiring them to be integers or constraining the sizes, e.g. the length of vectors in the arguments must be equal. These are constraints on the group meta-information not on the actual group content. *Function* is a set of constraints specifying that the data in the subgroups satisfies the function. For example, an excel constraint *rank* has Name *Y = RANK(X)*, where X and Y are the variables; its Signature is: the group G_X (associated with X) is numeric, G_Y is integer and the length of vectors in G_X is the same as in G_Y; its Function is the following constraint: a pair of vectors X, Y is a solution iff $X \in G_X, Y \in G_Y$ and each value in X has the rank (possibly with ties) specified in Y.

3 Problem Statement

In the previous section we introduced the problem of tabular constraint learning informally using the example in Figure 1. Here we formalize the statement in terms of Excel constraints and group assignments.

Tabular Constraint Learning Problem

Given:	The set of all groups $\mathcal{G}$ and the set of Excel constraints $\mathcal{C}$ with its dependencies DAG $\mathcal{D}$
Find:	text

4 Approach to Tabular Constraint Learning

Let us describe the key steps and functions in Algorithm 1. Essentially, the algorithm has two steps: candidate group generation and

[1] KU Leuven, Belgium, email: firstname.lastname@kuleuven

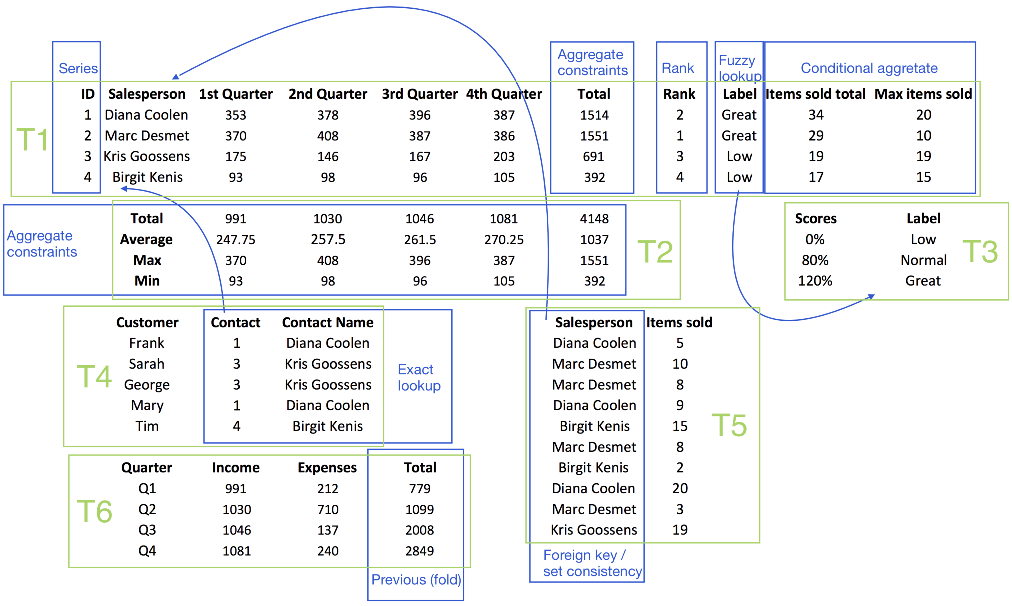

Figure 1. An example of constraint reconstruction (in blue) with indicated groups (in green)

subgroup satisfaction search, which is in line with the "generate-and-test" paradigm that is well-known in AI [5]. Let us elaborate on each step in detail.

Candidate group generation *generateAssignments*(*Constraint,GroupSet,Solutions*) is the function generating tuples of groups that are legitimate solution candidates e.g., X, Y are variables of Y = *RANK(X)* and X, Y satisfy the constraints from Signature above. Essentially, the group generation step is a constraint satisfaction problem associated with the specific constraint. However, many constraints have the same candidate generation procedures, e.g., sum, min, max, avg, count, etc.

Subgroup satisfaction search *findSolutions*(*Constraint,Candidates,Solutions*) is the function looking for the subsets of vectors in the candidates satisfying the constraint. If multiple subsets satisfy the constraint, a maximal is selected. If G_1, G_2, associated with the variables X, Y in the constraint Y = *RANK(X)*, are candidates, then *findSolutions* selects a single vector y in G_2 and a single vector x in G_1 such that x is ranked by y. For example, in Figure 1 the group $G = T_1[:, 3{:}8]$ serves as X and Y and x can be picked as $T_1[:, 7]$ and y would be $T_1[:, 8]$.

4.1 Constraints

The set of Excel constraints has a partial order in which they should be learned. For certain constraints this order does not matter, such as rank or product. For many others they have to be learned in the order, which is specified as a DAG in Figure 2 (if a constraint is not in the graph, it is independent). This figure specifies a potential parallelization of the computation, since each connected component is independent of the others.

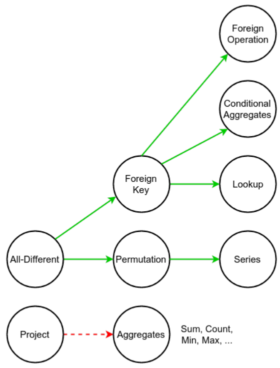

Figure 2. Constraint Learning Order. The solid arrows indicate a construction order, i.e., learning one constraint is necessary to build the other. The dashed arrows indicate a pruning order, the former constraint is used to prune the latter.

Algorithm 2 Workflow

Input: D – dataset, (optional: tables T, groups G)
Output: S – learned constraints with their satisfaction assignment
if T is **not** provided **then**
  $T \leftarrow extractTables(D)$
if G is **not** provided **then**
  $G \leftarrow extractGroups(D, T)$
$S \leftarrow learnConstraints(G)$
$S \leftarrow pruneRedundant(S)$
return S

4.2 Workflow

5 Case Study aka Experiments

Approach

- Notation
- Algorithm (select constraints, find assignments, find solutions)

Experimental questions

- How accurate are we? (Accuracy / recall)

Name	Signature	Function
Y = RANK(X)	$\mathit{numeric}(G_X) \wedge \mathit{integer}(G_Y) \wedge \mathit{length}(G_X) = \mathit{length}(G_Y)$	$X \in G_X, Y \in G_Y : \forall i, j \in \mathcal{N} : (Y_i \leq Y_j \rightarrow X_{Y_i} \leq X_{Y_j})$ $\wedge\, 1 \leq Y_i \leq \mathit{length}(G_Y)$
ALLDIFFERENT(X)	$\mathit{textual}(G_X) \vee \mathit{integer}(G_X)$	$X \in G_X : \forall i, j \in \mathcal{N} : i \neq j \rightarrow X[i] \neq X[j]$

Table 1. Excel Tabular Constraints SERGEY: Luc, we need your comments here on the notation of X, Y and G_X, G_Y

- How fast are we and which factors affect the runtime (how)?
- How general is our approach, what limitations are there?

6 Related Work

SERGEY: key bullet points for Luc and possibly Samuel and me to make related work section

SERGEY: ECAI reference style file ignores their guideline and their guideline ignores what is written in the guidelines! flashfill, flashextract, flashmeta [3, 4, 6]

- their supervised vs our unsupervised approach
- they look for a single "smallest" solution, we enumerate them all
- they are looking for a function, we solve constraint satisfaction problems
- we do not assume classic row based data layout, we work in the tabular setting

sketch [8]

- look for a constant that would fill in the gap in a program
- tailored for programming languages
- similar to model checking
- looks for a single solution
- similar to constraint satisfaction and sat, where one is interested in a single assignment that works for any potential input

tabular [2]

- language based on the excel tables that specify probabilistic models
- a system for probabilistic inference and similarity mostly in the usage of excel
- probabilistic constraint satisfaction (?) and graphical models
- single solution again

modelseeker [1] SERGEY: Samuel, Luc, probably you would need elaborate here more in details

- not designed for excel-like data representation (type consistency, groups, etc)
- not designed for excel-like constraints (lookups, conditional ifs, etc)
- does not support user extensions (?)

claudien [7] SERGEY: Samuel, Luc, you would need to help with this one

REFERENCES

[1] Nicolas Beldiceanu and Helmut Simonis, *Principles and Practice of Constraint Programming: 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, chapter A Model Seeker: Extracting Global Constraint Models from Positive Examples, 141–157, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[2] Andrew D. Gordon, Thore Graepel, Nicolas Rolland, Claudio Russo, Johannes Borgstrom, and John Guiver, 'Tabular: A schema-driven probabilistic programming language', Technical Report MSR-TR-2013-118, (December 2013).

[3] Sumit Gulwani, 'Automating string processing in spreadsheets using input-output examples', in *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, pp. 317–330, New York, NY, USA, (2011). ACM.

[4] Vu Le and Sumit Gulwani, 'Flashextract: a framework for data extraction by examples', in *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014*, p. 55, (2014).

[5] Vladimir Lifschitz, 'What is answer set programming?', in *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pp. 1594–1597, (2008).

[6] Oleksandr Polozov and Sumit Gulwani, 'Flashmeta: a framework for inductive program synthesis', in *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2015, part of SLASH 2015, Pittsburgh, PA, USA, October 25-30, 2015*, pp. 107–126, (2015).

[7] Luc De Raedt and Luc Dehaspe, 'Clausal discovery', *Machine Learning*, **26**(2-3), 99–146, (1997).

[8] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat, 'Combinatorial sketching for finite programs', *SIGOPS Oper. Syst. Rev.*, **40**(5), 404–415, (October 2006).