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]

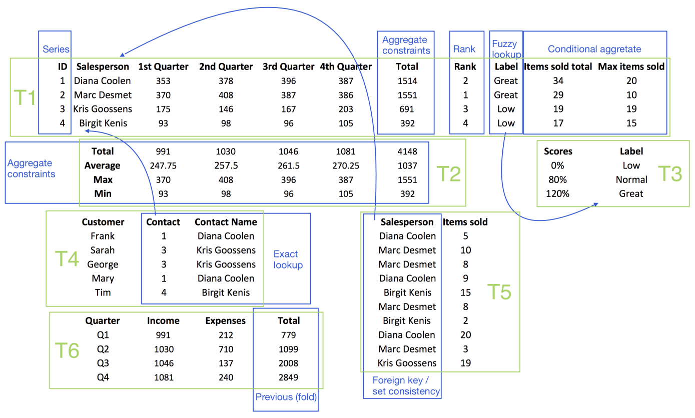
Motivation:

- 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: to himself we need structure here



ID	Salesperson	1st Quarter	2nd Quarter	3rd Quarter	4th Quarter	Total	Rank	Label	Items sold total	Max items sold
1	Diana Coolen	353	378	396	387	1514	2	Great	34	20
2	Marc Desmet	370	408	387	386	1551	1	Great	29	10
3	Kris Goossens	175	146	167	203	691	3	Low	19	19
4	Birgit Kenis	93	98	96	105	392	4	Low	17	15

Total	991	1030	1046	1081	4148
Average	247.75	257.5	261.5	270.25	1037
Max	370	408	396	387	1551
Min	93	98	96	105	392

Scores	Label
0%	Low
80%	Normal
120%	Great

Customer	Contact	Contact Name
Frank	1	Diana Coolen
Sarah	3	Kris Goossens
George	3	Kris Goossens
Mary	1	Diana Coolen
Tim	4	Birgit Kenis

Salesperson	Items sold
Diana Coolen	5
Marc Desmet	10
Marc Desmet	8
Diana Coolen	9
Birgit Kenis	15
Marc Desmet	8
Birgit Kenis	2
Diana Coolen	20
Marc Desmet	3
Kris Goossens	19

Quarter	Income	Expenses	Total
Q1	991	212	779
Q2	1030	710	1099
Q3	1046	137	2008
Q4	1081	240	2849

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

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 $v_1\colon G_1, \dots, v_n\colon G_n \in$ generateAssignments$(c, G, S)$ do
        $S \leftarrow S \cup$ findSolutions$(c, v_1\colon G_1, \dots, v_n\colon G_n, S)$
return $S$
```

2 Formalization

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 a subrange of a column or of a row that is type-consistent. If a vector is a subrange of 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 group G in the N-th table with rows ranging from a to b: $G = T_N[a{:}b, :]$, where N, a, b are natural numbers. Similarly for a column group G ranging from a to b columns in the N-th table we write $G = T_N[:, a{:}b]$.

A *constraint* is a triple of *(Name,Signature,Function)*, where Name is a textual name of the constraint; Signature is the signature of the constraint consisting of its arity and properties of the arguments, such as their types, sizes (properties might constrain sizes of two arguments as well, by for example, require them to be equal), etc; Function is the function over arguments specified in Signature that represents constraint satisfaction. For example, a column-wise sum constraint has Name equal to *Y = SUM(X, col)*, its Function sets arity to 2 and both arguments must be numeric and SERGEY: Samuel, could you put the actual properties and Function here?

2.2 Constraint learning algorithm description

Let us describe the key steps and functions in Algorithm 1. Essentially, the algorithm has two steps: candidate group generation and subgroup satisfaction search, which is in line with the well-known in AI paradigm of "generate-and-test" [5]. Let us elaborate on each step in detail.

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

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 = SUM(X)* and X, Y satisfy the following constraints SERGEY: Samuel, can you add here, what is actually used for the sum now?. 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., 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 = SUM(X)*, are candidates, then *findSolutions* selects a single vector y in G_2 and consecutive subset of vectors x in G_1 such that the sum over x is equal to y. For example, in Figure 1 the group $G = T_1[:, 3{:}7]$ can be used for both X and Y in the row-wise sum constraint, then as x would be selected $T_1[:, 3{:}6]$ and as y would be $T_1[:, 7]$.

2.3 Constraints

SERGEY: we need to explain that constraints order is a partial order, which is a DAG

Name	Signature	Function

Table 1. Tabular Constraints

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 $S$
```

3 Case Study aka Experiments

Approach

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

Experimental questions

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

4 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).