Discovering Denial Constraints Using Boolean Patterns

Sergio Luiz Marques Filho
supervised by Eduardo Cunha de Almeida
Federal University of Paraná, Brazil
slmfilho@inf.ufpr.br

CCS CONCEPTS

• **Information systems → Integrity checking**.

KEYWORDS

denial constraints, relational database, FPGA accelerators

ACM Reference Format:
Sergio Luiz Marques Filho. 2023. Discovering Denial Constraints Using Boolean Patterns. In *Companion of the 2023 International Conference on Management of Data (SIGMOD-Companion '23), June 18–23, 2023, Seattle, WA, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3555041.3589392

1 PROBLEM AND MOTIVATION

Integrity Constraints (ICs) are fundamental rules used to avoid inconsistencies in database updates. Different types of ICs, such as key constraints, functional or order dependencies, can be generalized by denial constraints (DCs) [2, 9], a more powerful constraint language.

The discovery of DCs is a fundamental step in data cleaning pipelines. Manually discovering DCs is difficult because it requires domain expertise and database knowledge. It is known as a time-consuming and error-prone task [12]. Thus, automatic discovering DCs is desirable. However, the large search space exhibits exponential time complexity in the number of constraint predicates and requires large amounts of memory to process intermediate data that may hinder the execution of discovery algorithms [9].

We propose a new method named **Boolean Patterns (BP)** that automatically recognizes patterns within a set of distinct evidences identifying the existing DCs. The main appeal of **BP** is its simplicity, bringing the discovery of DCs from the land of elaborated data structures to the land of boolean signs. This opens many research opportunities. In particular, we are currently studying two opportunities. First, BP drastically reduces the amount of memory required to keep intermediates in the evidence set data structures. Second, the execution of boolean signs allows exploring the discovery of DCs in highly parallel emerging hardware, like GPUs/FPGAs and processing-in-memory (PIM), to offload the discovery execution and overcome performance bottlenecks in the CPU. We hypothesize that BP offers a more efficient discovery of DCs than the state-of-the-art algorithms, with better use of memory space and flexibility to offload the required computation in emerging hardware.

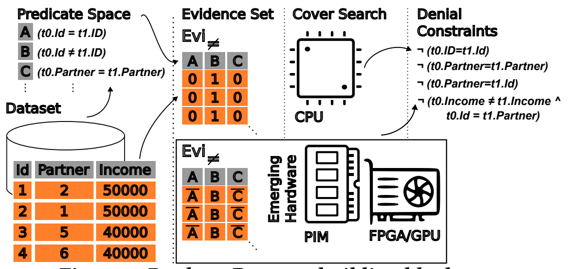

Figure 1: Boolean Patterns building blocks.

2 BACKGROUND AND RELATED WORK

The method presented in [2] shows that the problem of discovering DCs can be transformed into a problem of discovering minimal coverage sets (hereafter "*mincovers*") of an evidence set. Recent methods [1, 6, 8, 9, 14] proposed enhancements to improve efficiency in DC discovery. All these works keep an evidence set structure in memory. Not optimized evidence set construction can cause enormous problems such as a huge memory footprint, incorrect cache utilization, ineptitude for parallelism and other issues that can hinder DC discovery. Therefore, we seek efficient methods to deal with these problems and benefit from high-performance hardware.

To support the discovery of DCs, [13] maintains indexes even bigger than the original dataset. FACET [10] also implements indexes to discover DC violations keeping diverse and intricate data structures in memory. Our approach builds sole distinct evidence set in memory, keeping memory footprint minimal.

All previous works use complex algorithms and data structures that cannot benefit from emerging hardware architectures. In contrast, the **BP** method employs simple algorithms and minimum bitwise data structures more suitable for hardware acceleration.

Discovering *mincovers* is equivalent to hitting set enumeration [3, 5]. According to their structure, the hitting set enumeration algorithms are categorized in iterative, such as BMR [11], RS and the state-of-the-art algorithm MMCS [7], and hill-climbing, such as MTminer [4] and now **BP**. In the DC search, the number of predicates is large, causing a huge search space. To our best understanding, this is the first study comparing *mincovers* algorithms to assess how they handle large inputs in DC discovery.

3 UNIQUENESS AND APPROACH

Consider a relational instance r with schema $\mathbb{R}(A_1, ..., A_n)$, a tuple $t \subset r$, and a set $O = \{=, <, >, \neq, \leq, \geq\}$ operators. A predicate is a comparison between two attributes and an o operator that takes the form $t_x.A_i o t_y.A_j$, where A_i, A_j are different attributes in schema $\mathbb{R}$, t_x, t_y are different tuples in the relational instance r and $o \in O$. A predicate space $\mathbb{P}$ from a relational schema $\mathbb{R}$ is a set of all predicates with which DCs can be formed. A DC φ over an instance r is an expression in the form $\varphi : \forall t_x, t_y \in r, \neg(p_i \wedge ... \wedge p_n)$ in which p_i is a predicate. A DC expresses a set of predicates that cannot be true together for any combination of tuples in a relationship. For example, DC $\neg(t0.Id = t1.Id)$ means no tuple pairs can have the same Id. A structure called *evidence set* (*Evi*) is used to guide the search for DCs and validate DCs candidates. *Evi* is formed by comparing the tuples in the dataset, checking the predicates each tuple pair satisfies.

We define $Evi_{\neq}$ as a subset of Evi. As only distinct values are allowed in $Evi_{\neq}$, then $|Evi_{\neq}| \leq |Evi|$. The usage of $Evi_{\neq}$ is capable of considerably reducing the size of the evidence set.

A sample of $Evi_{\neq}$ generated over the dataset shown in Figure 1 for $\mathbb{P}$ predicates is $Evi_{\neq} = \{\{0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0\}, ...\}$, where 0 means no match, and 1 indicates a match in the predicate.

3.1 Boolean Patterns

The search for *mincovers* requires processing expensive scans on evidence sets. As no single boolean algebra gate can deal with all possible combinations of evidences alone, we introduce **Boolean Patterns**, a method that works by evaluating values on $Evi_{\neq}$ looking for patterns that ensure whether a rule is a *mincover* and consequently forms a DC. **BP** can detect *mincovers* or discard them even without having to go through all the involved records, as will be seen in the sequel.

$Evi_{\neq}$ can be seen as boolean signs suitable for hardware acceleration. We set up a function named $FullCover(\Sigma)$ that takes a set of predicates as input and returns $Evi_{\neq}$ equivalent boolean signs, *e.g.*, calling $FullCover(\mathbb{P})$ returns $\Psi = \{\{\overline{A}B\overline{C}DE\overline{FG}H\overline{I}J\overline{K}LM\overline{N}\}, \{\overline{A}B\overline{C}D\overline{E}F\overline{G}HI\overline{JK}LMN\}, \{\overline{A}B\overline{C}D\overline{E}FG\overline{HI}J\overline{KL}MN\}, \{\overline{A}B\overline{C}DE\overline{FG}H\overline{I}J\overline{KL}MN\}\}$

We set two theorems. The **Viability** is a *mincover* indicator pattern which states that *assuming a set of rules* Σ *and* $\Psi = FullCover(\Sigma)$, *if the values* Ψ *match with the pattern* Π, *and* $\Phi \not\subset \Psi$, *then the group* Σ *of rules is a mincover of* $Evi_{\neq}$ (see examples 1, 2, 3 and 6 in Table 2). The **Unviability** is the discard indicator pattern which says that *considering a set of rules* Σ *and* $\Psi = FullCover(\Sigma)$, *if* $\Phi \subset \Psi$ *then* Σ *is not a mincover of* $Evi_{\neq}$ (see examples 4 and 5 in Table 2). Depicting example 6 in table 2, calling $FullCover(\Sigma = \{EN\})$ will generate $\Psi = \{\{E\overline{N}\}, \{\overline{E}N\}, \{EN\}\}$ showing that rules $\{EN\}$ form a *mincover* due the existence of pattern $\Pi = \{\{E*\}, \{*N\}\}$.

N	Rule (Σ)	Viability (Π)	Unviability (Φ)
1	$\{P\}$	$\{P\}$	$\{\overline{P}\}$
2	$\{PQ\}$	$\{\{P*\}, \{*Q\}\}$	$\{\overline{PQ}\}$
3	$\{PQR\}$	$\{\{P**\}, \{*Q*\}, \{**R\}\}$	$\{\overline{PQR}\}$
...	...	...	...
$\lvert\mathbb{P}\rvert$	$\{PQRS...\}$	$\{\{P*...\}, \{*Q*...\}, \{**R*...\}, \{***S*...\}, ...\}$	$\{\overline{PQRS...}\}$
$P, Q, R, S, ...$ are boolean variables that can represent any predicate in the predicate space $\mathbb{P}$. The $*$ character is a wildcard in the domain $\{P, \overline{P}\}$, $\{Q, \overline{Q}\}$, $\{R, \overline{R}\}$, ... depending on its column.			

Table 1: Boolen Patterns.

#	Rule	$FullCover(\Sigma)$	Pattern found		DC
	Σ	Ψ	Π	Φ	φ
1	$\{B\}$	$\{B\}$	$\{B\}$		$\neg(t0.Id = t1.Id)$
2	$\{D\}$	$\{D\}$	$\{D\}$		$\neg(t0.Partner = t1.Partner)$
3	$\{M\}$	$\{M\}$	$\{M\}$		$\neg(t0.Id = t0.Partner)$
4	$\{F\}$	$\{\{F\}, \{\overline{F}\}\}$		$\{\overline{F}\}$	
5	$\{EL\}$	$\{\{EL\}, \{\overline{EL}\}, \{E\overline{L}\}\}$		$\{\overline{EL}\}$	
6	$\{EN\}$	$\{\{E\overline{N}\}, \{\overline{E}N\}, \{EN\}\}$	$\{\{E*\}, \{*N\}\}$		$\neg(t0.Income \neq t1.Income \wedge t0.Id = t1.Partner)$

Table 2: Examples of Boolean Patterns usage.

Rules formed with the possible combinations of $\mathbb{P}$ create a set enumeration tree in which a breadth-first search is carried out. For each node composed of groups of rules Σ, a call to $FullCover(\Sigma)$ is performed, and the result is compared to Π and Φ patterns to determine whether these rules form a *mincover*. **BP** uses efficient dynamic programming techniques to store previous results, reducing the number of comparisons between rules.

4 RESULTS

Our preliminary experiments compared **BP** with the related work algorithms that deal with the *mincover* problems. We set up a first round of experiments using the *BMS-WebWiew-2* [7] dataset commonly used for evaluating *mincover* problems and a second round using real-world datasets used in DCs experiments *Hospital*, *SPStock* and *Airport* [1, 2, 9].

We used the method stated in [2] to prepare the evidence sets. For all datasets, we built $Evi \approx 90k$ rows to have a hard comparison in the worst-case scenario.

Experiments were carried out in a CPU. The environment runs on a LMDE 5 (elsie) machine with AMD EPYC 7401 24-Core Processor 2.0GHz CPU and 200GB RAM.

	800	400	200	100	50	30	20
BMR	1.990	9.070	67.640	304.250	767.210	1316.450	1884.150
RS	0.030	0.100	1.030	13.530	108.770	498.300	1186.850
MMCS	0.030	0.120	1.210	14.520	143.740	626.770	1329.150
MTminer	0.044	0.153	0.535	1.896	5.127	11.409	19.011
BP	0.020	0.070	0.280	1.188	3.714	8.283	15.571

Table 3: Execution time in seconds for BMS-WebWiew-2 dataset.

	Hospital		SPStock		Airport	
	time(s)	Evi(Kb)	time(s)	Evi(Kb)	time(s)	Evi(Kb)
BMR	0,274	5220	0,398	5916	0,265	1137
RS	0,319	5220	1,088	6264	0,184	4555
MMCS	0,414	5220	1,189	6264	0,164	4555
MTminer	0,264	696	5,488	696	0,324	700
BP	0,021	0,75	0,561	1,66	0,018	0,67

Table 4: Real-world datasets execution time in seconds and size in Kbytes of evidence set kept in memory.

In the preliminary results, **BP** revealed superior performance, with a fraction of memory required by its counterparts to hold evidence sets in memory while allowing hardware acceleration.

Representing $Evi_{\neq}$ as boolean signs reduces the number of intermediate results when using **BP**, as demonstrated in our preliminary results, and it facilitates hardware acceleration, which is our next goal. Our future work involves utilizing reprogrammable hardware, such as an FPGA equipped with private memory that can be accessed via a PCIe connection. DMA controllers will transfer $Evi_{\neq}$ from the CPU memory to the accelerator's private memory. The process of DC discovery will occur in specialized hardware developed within the FPGA, where the low coupling between DC candidates can take advantage of the FPGA's high parallel architecture. The results will then be returned to the CPU memory. We will compare the accelerated version of **BP** with current CPU-based methods.

5 CONTRIBUTIONS

Our major contributions: 1) We present **Boolean Patterns**, a new algorithm to deal with *mincover* problems that can be used to discover DCs while keeping tiny structures in memory and is suitable for hardware acceleration. 2) We evaluate **BP** and its counterparts on real-world datasets used in discovering DCs, linking algorithms to deal with *mincovers* and the problem of DC discovery. 3) We created and validated a CPU version of **BP** assuring its correctness.

REFERENCES

[1] Tobias Bleifuß, Sebastian Kruse, and Felix Naumann. 2017. Efficient denial constraint discovery with hydra. *Proceedings of the VLDB Endowment* 11 (11 2017), 311–323. https://doi.org/10.14778/3157794.3157800

[2] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Discovering Denial Constraints. *Proc. VLDB Endow.* 6, 13 (2013), 1498–1509. https://doi.org/10.14778/2536258.2536262

[3] Andrew Gainer-Dewar and Paola Vera-Licona. 2017. The Minimal Hitting Set Generation Problem: Algorithms and Computation. *SIAM Journal on Discrete Mathematics* 31, 1 (2017), 63–100. https://doi.org/10.1137/15M1055024 arXiv:https://doi.org/10.1137/15M1055024

[4] Céline Hébert, Alain Bretto, and Bruno Crémilleux. 2007. A Data Mining Formalization to Improve Hypergraph Minimal Transversal Computation. *Fundam. Informaticae* 80 (2007), 415–433.

[5] Li Lin and Yunfei Jiang. 2003. The computation of hitting sets: Review and new algorithms. *Inform. Process. Lett.* 86, 4 (2003), 177–184. https://doi.org/10.1016/S0020-0190(02)00506-9

[6] Ester Livshits, Alireza Heidari, Ihab F. Ilyas, and Benny Kimelfeld. 2021. Approximate Denial Constraints. *Proc. VLDB Endow.* 13, 10 (mar 2021), 1682–1695. https://doi.org/10.14778/3401960.3401966

[7] Keisuke Murakami and Takeaki Uno. 2014. Efficient algorithms for dualizing large-scale hypergraphs. *Discrete Applied Mathematics* 170 (2014), 83–94. https://doi.org/10.1016/j.dam.2014.01.012

[8] Eduardo H. M. Pena and Eduardo Cunha de Almeida. 2018. BFASTDC: A Bitwise Algorithm for Mining Denial Constraints. In *Database and Expert Systems Applications - 29th International Conference, DEXA 2018, Regensburg, Germany, September 3-6, 2018, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11029)*, Sven Hartmann, Hui Ma, Abdelkader Hameurlain, Günther Pernul, and Roland R. Wagner (Eds.). Springer, Regensburg, Bavaria Land, Germany, 53–68. https://doi.org/10.1007/978-3-319-98809-2_4

[9] Eduardo H. M. Pena, Eduardo C. de Almeida, and Felix Naumann. 2019. Discovery of Approximate (and Exact) Denial Constraints. *Proc. VLDB Endow.* 13, 3 (nov 2019), 266–278. https://doi.org/10.14778/3368289.3368293

[10] Eduardo H. M. Pena, Eduardo C. de Almeida, and Felix Naumann. 2022. Fast Detection of Denial Constraint Violations. *Proc. VLDB Endow.* 15, 4 (apr 2022), 859–871. https://doi.org/10.14778/3503585.3503595

[11] K. Ramamohanarao, J. Bailey, and T. Manoukian. 2003. A Fast Algorithm for Computing Hypergraph Transversals and its Application in Mining Emerging Patterns. In *2013 IEEE 13th International Conference on Data Mining*. IEEE Computer Society, Los Alamitos, CA, USA, 485. https://doi.org/10.1109/ICDM.2003.1250958

[12] Shaoxu Song, Fei Gao, Ruihong Huang, and Chaokun Wang. 2020. Data Dependencies over Big Data: A Family Tree. *IEEE Transactions on Knowledge and Data Engineering* 34, 10 (2020), 1–1. https://doi.org/10.1109/TKDE.2020.3046443

[13] Zijing Tan, Ai Ran, Shuai Ma, and Sheng Qin. 2021. Fast Incremental Discovery of Pointwise Order Dependencies. *Proc. VLDB Endow.* 13, 10 (mar 2021), 1669–1681. https://doi.org/10.14778/3401960.3401965

[14] Renjie Xiao, Zijing Tan, Haojin Wang, and Shuai Ma. 2022. Fast Approximate Denial Constraint Discovery. *Proc. VLDB Endow.* 16, 2 (nov 2022), 269–281. https://doi.org/10.14778/3565816.3565828