Discovery of Denial Constraints Using Specialized Hardware Processors

Sergio Luiz Marques Filho[1] **, Eduardo Cunha de Almeida**[1]

[1]Department of Informatics (Dinf) – Federal University of Paraná (UFPR)
Av. Cel. Francisco H. dos Santos, 100 – 81530-000 – Curitiba – PR – Brazil
{slmfilho,eduardo}@inf.ufpr.br

Level: Doctorate (Computer Science, Federal University of Parana, Brazil.)
Advisor: Eduardo Cunha de Almeida
Admission: 02/2021 - **Qualification:** 06/2023 - **Defense:** 12/2025
Completed activities: Mandatory credits; bibliographic review; problem statement; thesis qualifying exam; FPGA hardware prototype;
Future activities: Thesis writing and defense.
Publications: SIGMOD '23: Discovering Denial Constraints Using Boolean Patterns [Marques Filho 2023].

Abstract. *Denial constraints (DCs) are at the heart of maintaining data consistency. Automatically discovering DCs from the data is computationally expensive due to the large search space. In this ongoing work, we propose a hardware-accelerated design for the automatic discovery of DCs, implemented on FPGAs. Our design eliminates the need for intermediate data structures, thereby reducing memory requirements and boosting performance. In preliminary experiments on a real-world dataset, our hardware-accelerated design outperformed state-of-the-art software solutions by factors ranging from 7× to 110×. An earlier version of our hardware design earned 3rd place in the prestigious ACM SIGMOD Research Competition.*

Resumo. *Restrições de Negação (DCs) são fundamentais para manter a consistência dos dados. Descobrir DCs automaticamente a partir de dados é computacionalmente custoso devido ao grande espaço de busca. Neste trabalho em andamento, propomos um método acelerado por hardware para a descoberta automática de DCs, implementado em FPGAs, que elimina a necessidade de estruturas de dados intermediárias, reduzindo os requisitos de memória e aumentando o desempenho. Em experimentos preliminares com um conjunto de dados do mundo real, nosso método superou soluções de software de estado da arte por fatores entre 7x e 110x. Uma versão anterior do nosso método conquistou o 3º lugar na prestigiada Competição de Pesquisa ACM SIGMOD.*

1. Introduction

Businesses have generated and consumed large amounts of data. Consequently, organizations want to access and analyze these data as a consolidated whole, bringing the need to employ database management systems (DBMS) to manage and organize such data in a structured manner.

Thus, a database can be seen as a partial model of an external reality. It is of common interest that a database be in accurate correspondence with this external reality. Therefore, certain conditions are imposed to capture the meaning of the outside reality, and these conditions are named *Semantic Constraints* or *Integrity Constraints (ICs)*. ICs are used to validate the integrity and consistency of the external entities represented in the data, and ICs are valuable tools in several DBMS tasks, such as database design [Papenbrock and Naumann 2017], data integration [Wang et al. 2009], query optimization [Kossmann et al. 2021], data cleaning [Chu et al. 2013b] and many others.

Although integrity constraints find practical use in various applications, they also have different formalisms to define them, with their own level of expressiveness, such as *functional dependencies (FD)*, *conditional functional dependencies*, *unique column combinations*, and *order dependencies*, among others. In these independent formalisms, each type of IC has limited expressiveness and cannot encompass the high number of types of rules that can be observed in actual datasets. Moreover, the different ICs are logically isolated, making the interaction between ICs difficult [Bleifuß et al. 2017].

All the above-mentioned IC formalisms can be generalized by a more powerful constraint language such as *Denial Constraints (DC)* [Chu et al. 2013a, Pena et al. 2019]. The task of manually discovering DCs is difficult, as it requires domain expertise and database knowledge, and is known to be time-consuming and error-prone [Song et al. 2020]. Thus, automatically discovering DCs is desirable, but this is also a difficult task due to the large search space.

This document focuses on the automatic discovery of DCs, which express a set of predicates that cannot be true together for any combination of tuples in a relationship. We walk through the example dataset shown in Table 1 to demonstrate the expressiveness of a DC. We observe that flights with the same origin airport cannot have a different origin city market, a rule expressed by the denial constraint $\varphi : \forall t_x, t_y \in \texttt{Flights}, \neg(t_x.OriginCityMarketID \neq t_y.OriginCityMarketID \wedge t_x.OriginAirportID = t_y.OriginAirportID)$.

Table 1. A glance of Flights dataset.

	FlightNum	Origin AirportID	AirlineID	OriginCity MarketID
t_1	305	12953	19805	31703
t_2	1527	13204	20355	31454
t_3	4352	11618	20366	31703
t_4	5358	12892	20304	32575
t_5	4602	12892	19393	32575

As DCs are expressive enough to subsume many other dependencies, to fully take advantage of the benefits of DCs, the set of valid DCs is expected to be known. To support DC discovery, current state-of-the-art algorithms follow the process of constructing intermediate data structures from the data and subsequently deriving DCs from these intermediates. However, these algorithms suffer from inefficiencies in intermediate computation [Pena et al. 2022]. More-

over, the utilization of high-performance hardware may be hindered if complex intermediate data structures are stored in memory during the implementation of such algorithms.

The usage of specialized hardware in data processing has a long tradition [Mueller et al. 2009]. To overcome the current difficulties presented by software-based methods, we propose specialized hardware to discover DCs implemented on top of Field-Programmable Gate Arrays (FPGAs). FPGAs offer greater flexibility, intrinsic parallelism, and distributed on-chip memory, enabling the design and evaluation of customized hardware solutions, and as their adoption in data centers grows, the integration of specialized hardware into data processing systems has become increasingly practical and efficient [Jiang et al. 2023].

2. Related Work

In software-based DC discovery, algorithms based on sets of evidence have become the standard approach [Chu et al. 2013a, Bleifuß et al. 2017, Pena and de Almeida 2018, Pena et al. 2019, Livshits et al. 2021, Xiao et al. 2022, Pena et al. 2022]. FastDC [Chu et al. 2013a] uses a depth-first search to evaluate all possible DC candidates in a lattice of column combinations and identifies minimal coverages through intersections in an evidence set. However, its approach demands enormous memory to build the lattice, which can take days to process, rendering it impractical [Papenbrock et al. 2015].

BFastDC [Pena and de Almeida 2018] and DCFinder [Pena et al. 2019] improve evidence set building by processing chunks of evidence at a time to benefit from hardware caches and using indices. However, these algorithms require a computationally expensive procedure to revisit the chunks of evidences for building the evidence set, which can be quadratic in the number of pieces of evidence. Hydra [Bleifuß et al. 2017] discovers exact DCs using a sampling-based approach to construct intermediate DC sets, reducing the computational cost of evidence set construction, and finally extracts the final DCs from the complete evidence set.

The Evidence Context Pipeline (ECP) [Pena et al. 2022] introduces new intermediate representations, indexes, and algorithms to build a parallel pipeline to generate evidence sets. ECP incorporates DC enumeration techniques leveraging inverted indices and pruning strategies to reduce the search space and enable parallelism.

Current software-based methods discover DCs from the evidence set intermediate structures, and all these algorithms keep such a 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 problems that can hinder DC discovery. In contrast, we use simple structures carrying boolean signals appropriate to hardware acceleration.

2.1. Presentation of the Problem

Discovering DCs requires detecting all minimal DCs that hold in a given relation instance, but the high expressive power of DCs allows them to express a variety of predicates, resulting in a huge search space. The number of DCs that potentially hold in a database instance is $2^{|P|}$, where $|P|$ is the number of predicates in the predicate space.

Software-based algorithms are plagued by inefficiencies in computing intermediates, resulting in significant overhead. In certain instances, the structures produced by

these algorithms are larger than the initial dataset [Pena et al. 2022]. Moreover, current DC discovery methods are fine-tuned to run on a CPU, with none designed to leverage the advantages of high-performance hardware.

Given all these problems, we have some questions to answer. 1. Can specialized hardware make better use of memory by eliminating the intermediate data structures maintained by software versions during the DC discovery process? 2. How does specialized hardware performance compare to software methods for discovering DCs? 3. How does specialized hardware scale in relation to the number of attributes and the number of rows in the dataset?

Considering that current DC discovery algorithms follow the process of first building an intermediate data structure and then enumerating the DCs based on these complex intermediates, we hypothesize that the proposed DC Specialized Hardware offers a more efficient discovery of DCs than the state-of-the-art algorithms, with better use of memory space with accelerated computation in highly parallel hardware.

The main objective of this work is to propose a new method for discovering denial constraints using reconfigurable hardware, such as an FPGA, and to evaluate the performance gains provided by hardware acceleration.

2.2. Methodology

The Denial Constraints Specialized Hardware processor is a hardware accelerator built to assess attribute values in a dataset and identify the exact DCs that hold. Its goal is to efficiently traverse the search space looking for DCs while keeping a small memory structure inside the FPGA board that eliminates the need for an intermediate set of evidence structures, which is a major bottleneck in state-of-the-art software-based solutions. Figure 1 illustrates the interaction between a CPU and the DC specialized hardware processor implemented on top of an FPGA board.

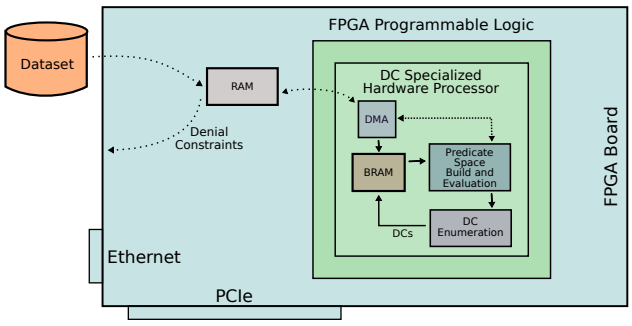

Figure 1. Architecture of the Denial Constraint Specialized Hardware Processor.

In the first step of DC discovery, we perform the *Predicate Space Building* in which we compare attribute values to derive and evaluate DC predicates. A pair of tuples t_x, t_y is the fundamental processing unit in the DC discovery, where the building of the predicate space requires iterating over all tuple pairs [Chu et al. 2013b], with predicates on categorical attributes using the operators $\{=, \neq\}$, and those on numerical attributes also including $\{<, \leq, >, \geq\}$. Our FPGA-based acceleration employs dedicated parallel circuits that efficiently execute multiple value comparisons simultaneously.

Attribute values are structured in pipelines and stored locally in Block RAMs (BRAMs), interfacing with Direct Memory Access (DMA) via the Advanced eXtensible Interface (AXI) protocol, which feeds the pipeline during comparison execution.

Next, we perform the *DC enumeration*, which is the process of generating DCs that hold in the dataset. This process handles the search for minimal DCs, which are DCs that cannot be derived from any other. This minimality property is important for eliminating redundant DCs that reduce the search space [Chu et al. 2013a].

We organize the predicate space as a set enumeration tree (SE-Tree) representing all possible irredundant combinations of predicates. The SE-tree node can be designed using boolean algebra symbols, *e.g.*, we use the boolean algebra symbol $A_{=}$ to denote an equality predicate on attribute A, with other comparison operators following the same notation. Thus, a combination such as $A_{=} + B_{\neq}$ represents the disjunction of the equality predicate on attribute A and the non-equality predicate on attribute B.

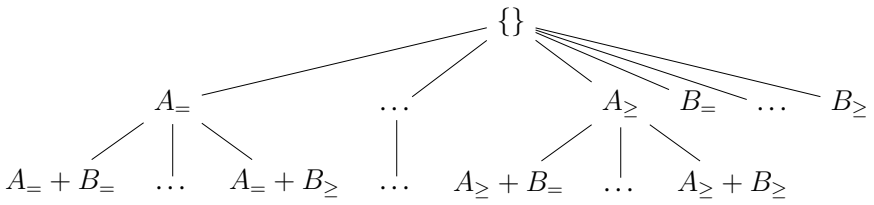

Figure 2. The SE-Tree with the possible combinations of two attributes.

We apply the forward path sharing technique for predicate prefix merging [Maschi et al. 2020], in which combinations with fewer predicates are placed at higher levels of the SE-Tree, while those with more predicates appear at lower levels.

Evidence is data satisfying one or more predicates, and a cover is a set of predicates that overlap with such evidence. Instead of using intermediate data structures, we use components inside the FPGA responsible for performing the cover verification aligned with the evaluation of predicates to determine whether predicates hold. To test the *minimal cover*, we verify if a subset of the tested predicates also covers the evidence. The internal FPGA components are tied together in a chain, allowing the data to traverse the tree, evaluating whether the predicates are minimal.

After all data in the dataset have been processed, the DCs found remain available in the FPGA and can be obtained with read operations using the AXI protocol.

2.3. Preliminary Results

We carried out experiments aimed at answering the research questions stated in Section 2.1, comparing the DC Specialized Hardware with the state-of-the-art software-based algorithms that detect exact DCs: Hydra [Bleifuß et al. 2017] and DCFinder [Pena et al. 2019]. We use the real-world dataset *Flights*, which consists of data on the departure and arrival times of flights from different data sources, increasing the number of tuples from 1k to 8k, and the number of attributes from 1 to 4.

To perform the DC Specialized Hardware experiments, we synthesize the hardware in the FPGA part `xc7a100tcsg324-1` with a clock of 90MHz with a dual-core Cortex-A9 650MHz processor and 512MB DDR3 RAM. The software experiments to run the Hydra and DCFinder methods were carried out on an LMDE 5 (elsie) machine with AMD EPYC 7401 24-Core Processor 2.0GHz CPU and 200GB DDR4 RAM.

Table 2 presents the DC discovery execution time (in seconds), as reported by Hydra, DCFinder, and DC Specialized Hardware, while Table 3 shows the speedup of

Table 2. Hydra, DCFinder and the DC Specialized Hardware execution time (s) to detect DCs on the Flights dataset.

Attrib.	Hydra Dataset size				DCFinder Dataset size				**DC Specialized Hardware** Dataset size			
	1024	2048	4096	8192	1024	2048	4096	8192	1024	2048	4096	8192
1	0,1623	0,1730	0,2090	0,3063	0,3170	0,5200	1,1500	2,8300	0,0029	0,0054	0,0124	0,0361
2	0,1940	0,2157	0,2890	0,3857	0,3673	0,6100	1,4300	3,5400	0,0044	0,0079	0,0174	0,0456
3	0,2237	0,2543	0,3350	0,4373	0,3987	0,6293	1,6600	4,0100	0,0058	0,0113	0,0244	0,0615
4	0,2577	0,3497	0,4303	0,5450	0,4487	0,7600	1,8200	4,5000	0,0067	0,0138	0,0302	0,0721

Table 3. DC Specialized Hardware speedup over Hydra and DCFinder to Detect DCs on the Flights dataset.

Attrib.	Hydra Dataset size				DCFinder Dataset size			
	1024	2048	4096	8192	1024	2048	4096	8192
1	56,5941	32,1046	16,8244	8,4776	110,5722	96,4995	92,6014	78,2168
2	44,3330	27,4650	16,6463	8,4576	83,9432	77,6830	82,5788	77,5369
3	38,8759	22,5748	13,7469	7,1070	69,2930	55,8600	68,1598	65,2417
4	38,4732	25,3316	14,2323	7,5570	66,9922	55,0581	60,2805	62,3280

DC Specialized Hardware over Hydra and DCFinder.

Considering the results of the *Flights* dataset for a single attribute, as the number of tuples varies from 1024 to 8192, the DC Specialized Hardware outperforms DCFinder from 78× to 110× and Hydra from 8× to 57×. These values emphasize the impact of the creation of the intermediary data structures, such as the evidence set on the discovery of DCs done by Hydra and DCFinder, in contrast with the design of our method that eliminates the evidence set, outperforming all software-based algorithms in different configurations of tuple numbers and attribute numbers.

2.4. Conclusion

We have shown DC Specialized Hardware, a new approach to deal with the problem of detecting denial constraints that uses high-parallel hardware. Compared to its software counterparts, it demonstrated superior performance, from 7x to 110x.

Our major contributions are threefold: 1) We present a new method to deal with the discovery of DCs that uses memory efficiently by eliminating the intermediate data structures kept by the state-of-the-art software versions. 2) We present a specialized hardware processor using reconfigurable hardware on top of an FPGA, a new solution to the problem of discovering DCs. To our best understanding, this is the first method that uses high-parallel hardware to detect DCs. 3) We evaluate the specialized hardware processor and its software counterparts on a real-world dataset commonly used to discover DCs.

2.5. Acknowledgments

This work was partially supported by CNPq (grants 302909/2022-2 and 444192/2024-7) and by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Brazil – Finance Code 001.

References

Bleifuß, T., Kruse, S., and Naumann, F. (2017). Efficient denial constraint discovery with hydra. *Proceedings of the VLDB Endowment*, 11:311–323.

Chu, X., Ilyas, I. F., and Papotti, P. (2013a). Discovering denial constraints. *Proc. VLDB Endow.*, 6(13):1498–1509.

Chu, X., Ilyas, I. F., and Papotti, P. (2013b). Holistic data cleaning: Putting violations into context. In *2013 IEEE 29th International Conference on Data Engineering*.

Jiang, W., Parvanov, M., and Alonso, G. (2023). Swiftspatial: Spatial joins on modern hardware.

Kossmann, J., Papenbrock, T., and Naumann, F. (2021). Data dependencies for query optimization: a survey. *The VLDB Journal*, 31.

Livshits, E., Heidari, A., Ilyas, I. F., and Kimelfeld, B. (2021). Approximate denial constraints. *Proc. VLDB Endow.*, 13(10):1682–1695.

Marques Filho, S. L. (2023). Discovering denial constraints using boolean patterns. In *Companion of the 2023 International Conference on Management of Data*, SIGMOD '23, page 281–283, New York, NY, USA. Association for Computing Machinery.

Maschi, F., Owaida, M., Alonso, G., Casalino, M., and Hock-Koon, A. (2020). Making search engines faster by lowering the cost of querying business rules through fpgas. In *SIGMOD Conference 2020 [USA], June 14-19, 2020*, pages 2255–2270. ACM.

Mueller, R., Teubner, J., and Alonso, G. (2009). Data processing on fpgas. *Proc. VLDB Endow.*, 2(1):910–921.

Papenbrock, T., Ehrlich, J., Marten, J., Neubert, T., Rudolph, J.-P., Schönberg, M., Zwiener, J., and Naumann, F. (2015). Functional dependency discovery: An experimental evaluation of seven algorithms. *Proc. VLDB Endow.*, 8(10):1082–1093.

Papenbrock, T. and Naumann, F. (2017). Data-driven schema normalization. In Markl, V., Orlando, S., Mitschang, B., Andritsos, P., Sattler, K., and Breß, S., editors, *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017*, pages 342–353. OpenProceedings.org.

Pena, E. H. M. and de Almeida, E. C. (2018). BFASTDC: A bitwise algorithm for mining denial constraints. In *Database and Expert Systems Applications - DEXA 2018, Regensburg, Germany, September, 2018, Proceedings, Part I*, volume 11029 of *Lecture Notes in Computer Science*, pages 53–68, Regensburg, Germany. Springer.

Pena, E. H. M., de Almeida, E. C., and Naumann, F. (2019). Discovery of approximate (and exact) denial constraints. *Proc. VLDB Endow.*, 13(3):266–278.

Pena, E. H. M., Porto, F., and Naumann, F. (2022). Fast algorithms for denial constraint discovery. *Proc. VLDB Endow.*, 16(4):684–696.

Song, S., Gao, F., Huang, R., and Wang, C. (2020). Data dependencies over big data: A family tree. *IEEE Transactions on Knowledge and Data Engineering*, 34(10):1–1.

Wang, D., Dong, X., Das Sarma, A., Franklin, M., and Halevy, A. (2009). Functional dependency generation and applications in pay-as-you-go data integration systems.

Xiao, R., Tan, Z., Wang, H., and Ma, S. (2022). Fast approximate denial constraint discovery. *Proc. VLDB Endow.*, 16(2):269–281.