Solving Hard Variants of Database Schema Matching on Quantum Computers

Kristin Fritsch
University of Passau
Passau, Germany
fritsc23@ads.uni-passau.de

Stefanie Scherzinger
University of Passau
Passau, Germany
stefanie.scherzinger@uni-passau.de

ABSTRACT

With quantum computers now available as cloud services, there is a global quest for applications where a quantum advantage can be shown. Naturally, data management is a candidate domain. Workable solutions require the design of hybrid quantum algorithms, where a quantum computing unit (a QPU) and classical computing (via CPUs) cooperate towards solving a problem. This demo illustrates such an end-to-end solution targeting NP-hard variants of database schema matching. Our demo is intended to be educational (and hopefully inspiring), allowing participants to explore the critical design decisions, such as the handover between phases of QPU- and CPU-based computation. It will also allow participants to experience hands-on – through playful interaction – how easily problem sizes exceed the limitations of today's QPUs.

PVLDB Reference Format:
Kristin Fritsch and Stefanie Scherzinger. Solving Hard Variants of Database Schema Matching on Quantum Computers. PVLDB, 16(12): 3990-3993, 2023.
doi:10.14778/3611540.3611603

1 INTRODUCTION

For decades, quantum computing has been considered a futuristic technology from the viewpoint of data management research. Yet with quantum hardware available as cloud-based services, there is now acute interest to explore quantum computing units (QPUs).

Already, a growing community is exploring quantum machine learning [11]. For database core tasks, early research targets problems where the solution space cannot be exhaustively explored using classical hardware. This includes query planning in multi-query optimization, join order optimization, and transaction scheduling. We refer to [2] for an overview of the state-of-the-art.

Coming from a traditional computer science background, data architects encounter a very steep learning curve with quantum software engineering: While programming libraries and tutorials are readily available (e.g., several MOOCs offered on platforms such as *edx*), engineers need to adopt an entirely new way of thinking.

Several of the pioneer works surveyed in [2] resort to solving QUBO models, a mathematical formula which is then solved on a QPU. However, it is not sufficient to just find *any* QUBO formulation: The major physical resource that limits quantum computers is the available number of *qubits*, so developers must find formulations that are parsimonious in this regard.

Conceptually, programming quantum computers is vastly different, which also holds for the operational aspects: Whereas we are used to executing a program (once) and then collecting the (usually deterministic) answer, hybrid quantum algorithms are executed in a stochastic process: A problem (*task*) is issued to the QPU repeatedly (e.g., in 10K *shots*). The QPU produces solutions nondeterministically, and the most frequent solutions must then be assessed for their viability: Non-viable solutions are discarded, and only the viable solutions are considered for further processing.

Goal of this Demo. In this demo proposal, we focus on the established data management challenge of *database schema matching*.

We walk our demo attendees through our system architecture, starting from two input tables, and ultimately, presenting a specific matching. Such an end-to-end discussion is indeed a contribution: Existing works on quantum computing in database research focus on the handover between CPU and QPU [3, 10, 12]. For instance, solutions to query optimization problems so far ignore the steps of query parsing, logical optimization, or query cost estimation. Strong assumptions as to the shapes of join trees or the independence of cost estimates for predicates make it difficult to assess the impact for real-world query workloads.

By presenting an end-to-end solution, we illustrate where exactly quantum computing comes into play. As we (as a research community) have learned from building machine learning pipelines, considerable work goes into classical data engineering. This effect will also be observable with the engineering of a hybrid quantum algorithm in our demo scenario.

We further discuss the interfaces between classical and quantum computations, the challenges in crafting suitable encodings, and common patterns in engineering hybrid quantum algorithms, as comprehensively surveyed in [13].

With the problem of schema matching, we address an important data integration challenge: To integrate two tables, pairs of attributes are weighted according to a similarity function. This part of our workflow is solved on classical hardware. The goal is then to identify the candidate pairs that form suitable matches. This part will be off-loaded to a QPU.

Schema matching can be reduced to stable matching problems (a.k.a. stable marriage problems), a family of problems with NP-hard variants. Specifically, we focus on finding a matching of maximum cardinality, with ties and incomplete preference lists, and leverage a QUBO model proposed by Roch et al [9].

Contributions. Our demo makes the following contributions:

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611603

- We target hard variants of database schema matching. We present a solution implemented as a hybrid quantum algorithm, where pre- and post-processing are realized on classical CPUs, while solving the hard variants of the stable matching/marriage problem is delegated to the QPU.
- We point out the challenges in designing such a hybrid quantum algorithm, specifically the handover between phases of CPU- and QPU-based computing.
- We illustrate the QPU properties limiting the scalability of our solution, most notably the scarcity of qubits.
- We integrate different quantum architectures in our workflow. We work with gate-based and annealing-based QPUs, as well as software-based simulators.
- We present our solution end-to-end, starting from two database schemas to be matched. Although solutions to other data management problems will of course require careful adaption, our workflow can serve as a blueprint.

2 PRELIMINARIES

We structure our introduction of the necessary preliminaries along the workflow sketched in Figure 2. We proceed top-down.

Schema Matching. Schema matching is a well-known data integration challenge: The task is to find matching pairs of attributes between a source and a target schema. We refer to [1] for a systematic overview of the various approaches and to the Valentine framework for a collection of implementations [6]. As input, we assume relational schemas, so matching is based only on attributes. We employ a linguistic and schema-based matching algorithm such as Cupid [7]. The output is a similarity matrix, assigning each pair of attributes from the source and target schema a similarity score.

Stable Matching Problems. The next step is to compute a *global matching*. We reduce this task to the problem of finding a solution to the *stable matching problem* (also known as "stable marriage problem"). Our input is the similarity matrix, from which we derive an ordering of preferences for each element. A solution to the stable matching problem is then a specific assignment, where each individual is assigned at most one partner.

The matching is *stable* when there does not exist any pair in which the partners prefer each other to their current partner under the matching. If such pairs exist, they are called *blocking* in the terminology of the mathematical problem [9].

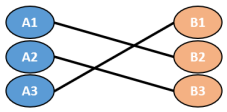
Table 1: Instance of stable matching problem w/ ranked preferences. Brackets denote a *tie*, i.e., equally preferred partners.

Set A	Preferences of Ai	Set B	Preferences of Bi
A1	B2, B1	B1	A2, A1, A3
A2	B3, [B1, B2]	B2	A3, A2, A1
A3	B1, B3, B2	B3	A1, A3, A2

Consider Table 1, for matching sets A and B with attributes $A1, A2, A3$ and $B1, B2, B3$ respectively. Individual $A1$ has the preferences $B2, B1$ (in this order), so $A1$ prefers $B2$ over $B1$. As $B3$ does not appear in the preference list of $A1$, we speak of *incomplete preference lists*.

If we consider the preference list of $A2$, we notice that $B1$ and $B2$ are set in brackets. This expresses the presence of a *tie*, i.e., $A2$ prefers $B1$ and $B2$ equally. Yet $A2$ still prefers $B3$ over $B1$ or $B2$.

One possible stable matching is shown below:



Every element of set A gets its first choice and every element of set B gets its third choice. The matching is stable since every pair is happy (it is not the case that *both* partners prefer another partner) and the pairing is monogamous (no one is matched twice).

Although there are variants of this problem that are computable in $O(n^2)$, there are also variants that are NP-hard, such as finding stable matchings of maximum cardinality with ties and incomplete preference lists [5]. This is the family of problems considered here.

QUBO formulations. A QUBO model is a quadratic unconstrained binary optimization formula. This formula is a sum that is based on binary variables and products of pairs of these binary variables. The products are restricted to quadratic relationships between the binary variables. Further, each term carries a coefficient. A solution to a QUBO model is an assignment of the variables with zero or one such that the overall sum is minimized.

The formulation of QUBO models for optimization problems is an established approach, and we can resort to mature software-based solvers [8]. However, QUBO models are also popular in quantum software engineering. We refer to [4] for a comprehensive tutorial on how to embed a range of well-known computer science problems into the NP-hard QUBO problem.

We can provide a very basic introduction only. QUBO models use binary variables and in our use case, each variable represents a candidate pair of attributes to be matched:

$$x_{ab} = \begin{cases} 1 & \text{if attributes } a \text{ and } b \text{ are matched,} \\ 0 & \text{otherwise.} \end{cases}$$

The QUBO formula in Figure 1 targets our family of matching problems and was proposed (as a standalone solution) by Roch et al. in [9]: A solution to the stable matching problem (i.e., an assignment of the variables) minimizes the objective function. We cannot go into the details regarding the design of this specific function, but we introduce the high-level idea. By p_1, p_2, p_3, we denote penalties that developers assign to weight the objective function (first part) and the constraints (second and third part) of the optimization.

The first part represents our main goal: to minimize the objective function and to find as many pairs as possible. The second part is declared for all candidate attribute pairs a, b. It adds the penalty $p2$ to the solution if a pair is not stable, thereby discouraging unstable pairs. In the first half, unstable pairs are identified, in the second half, pairs that are not likely to be blocking are promoted. Finally, the third part imposes the penalty $p3$ if candidates are matched more than once, which would result in nonmonogamous matchings.

Solving QUBO on QPUs. QUBO formulas can be solved on QPUs. There are two kinds of architectures, namely gate-based QPUs and quantum annealers. Both are able to run and solve our QUBO formula. However, gate-based QPUs need a quantum approximate

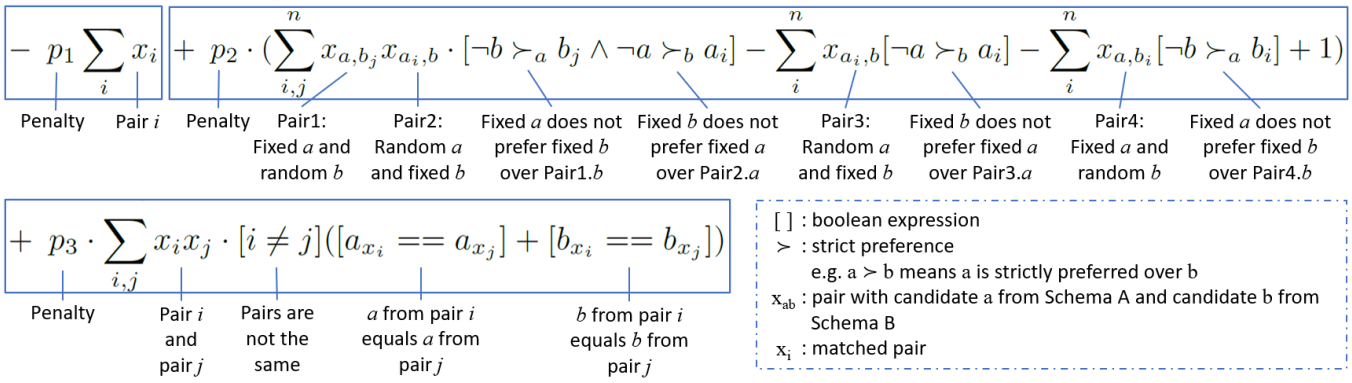


Figure 1: The QUBO formula proposed by [9] for the stable matching problem, annotated for better comprehensibility.

optimization algorithm, short QAOA, in between. The QUBO formula acts as an input to the QAOA that can then be run by the QPU. The challenge with gate-based QPUs is that there are currently only a few hundred qubits available, while we would need as many qubits as there are possible matching pairs. Ultimately, the execution time of gate-based QPUs depends on the depth of the resulting QAOA circuit.

Quantum annealers do not need an interim representation for executing the QUBO formula, since they are designed to solve optimization problems. They offer thousands of qubits.

A solution is obtained by issuing multiple shots to the QPU (up to several thousands shots). A shot yields one computed result, which can differ between shots, due to quantum mechanical effects.

When all shots are completed, the candidate solutions are returned, each with a candidate matching, a probability value capturing how often the solution was returned among all shots, and the result of evaluating the QUBO. The candidate solutions must be validated on a CPU, and invalid solutions discarded. Valid solutions with a low objective value constitute the best solutions found.

3 WORKFLOW AND DEMO OUTLINE

Our demo is set up as an interactive python notebook, following the workflow in Figure 2. Figure 3 shows screenshots. Our demo attendees may execute the workflow step-by-step and can inspect all interim results (e.g., the milestones ❶-❹ in the screenshot):

(1) The user enters the database schemas to be matched. In the demo, we provide a set of input schemas that are hand-crafted to fall into the category of NP-hard matching problems. Demo attendees can freely edit these inputs.
(2) The user confirms the database schema matcher, such as Cupid. Matching produces a similarity matrix and assigns to each candidate match a similarity score. Next, the preference lists are derived from the similarity matrix.
(3) The user confirms the global matching algorithm, in our case "stable matching".
(4) Our tool automatically translates the similarity matrix to the QUBO model, which the user can inspect.
(5) The user chooses the target platform: A gate-based quantum computer, a quantum annealer, or a simulation software.

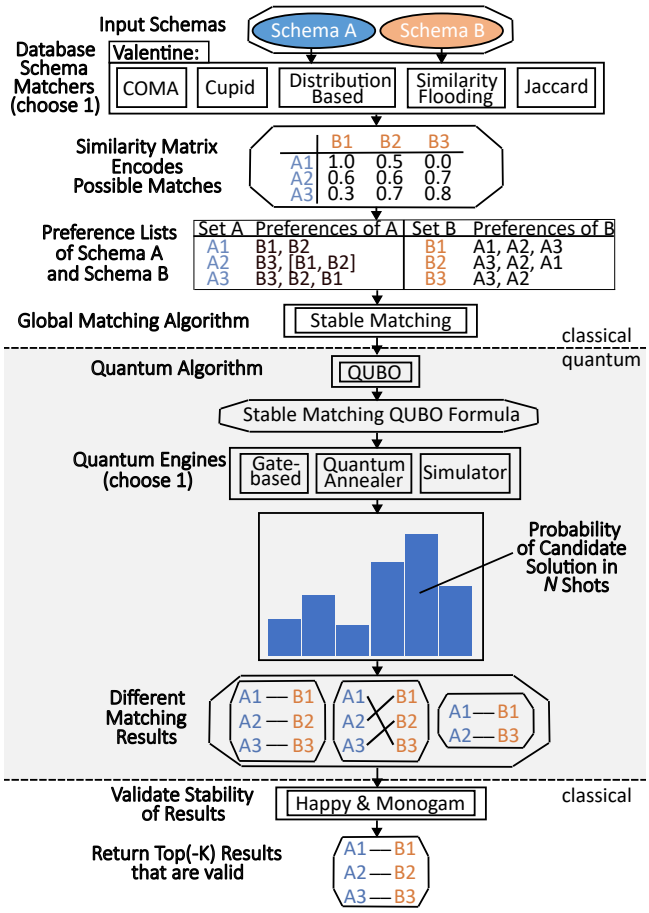


Figure 2: Workflow[1] with classical and quantum phases.

(6) The QUBO model is issued to the target platform in a sequence of N shots. The results are aggregated as a probability distribution (see blue barchart in the illustration).
(7) In post-processing, our tool automatically validates the candidate solutions produced by the N shots, to discard

[1]Visualization inspired by https://de.wikipedia.org/wiki/Globales_Matching.

matchings that are unstable or not monogamous. Validation is again solved on classical hardware.

(8) The top-k valid results are chosen and translated to human-consumable matchings, which are returned to the user.

Implementation. Our tool demo is realized as a Jupyter notebook, written in *python*. We use the libraries *pandas* for importing database schemas, the library *valentine*, which implements several matchers, *docplex* for encoding the QUBO formula within python, *qiskit* for access to IBMQ quantum engines, and *qiskit_optimization* for converting the docplex QUBO model into a valid input for the quantum engine, creating an QAOA optimizer with the quantum engine and solving the problem.

We use *ibmq_qasm_simulator* for simulation, and further the *ibmq_lima* QPU as gate-based QPU, both from IBMQ[2]. For quantum annealing, we use the *Advantage_System4.1* QPU by D-Wave[3].

Note that only small problems can be solved live in our demo, where we use simulators. For larger experiments, we can only show the results of prerecorded runs, as simulations easily take hours.

We also present pre-recorded results from real quantum hardware. Unfortunately, running these experiments live is not feasible, due to the waiting times in the job queues of the cloud providers.

Scalability. Our chosen QPU vendors currently support a maximum of 433 (exploratory gate-based QPU of IBM), 5,000 (simulator of IBM) or 5,627 (quantum annealer of D-Wave) physical qubits. The QUBO formulation we use requires as many logical qubits as possible pairs, so up to $|A| \times |B|$ logical qubits.

In embedding the QUBO model onto the QPU, several physical qubits may be required to represent one logical qubit. This will ultimately increase the total number of qubits required on the QPU. We point this effect out to our demo attendees.

Discussion items. In presenting this demo, our goal is to familiarize the VLDB community with the methodology of engineering hybrid quantum algorithms. Throughout our demo, we emphasize on the aspects of quantum software engineering for data management tasks, rather than tuning the embedding of the QUBO onto QPUs, or comparative benchmarking of QPUs from different vendors. These are all promising angles for future work, as well as the design of a custom QUBO formula for schema matching that directly integrates the similarity scores from the similarity matrix.

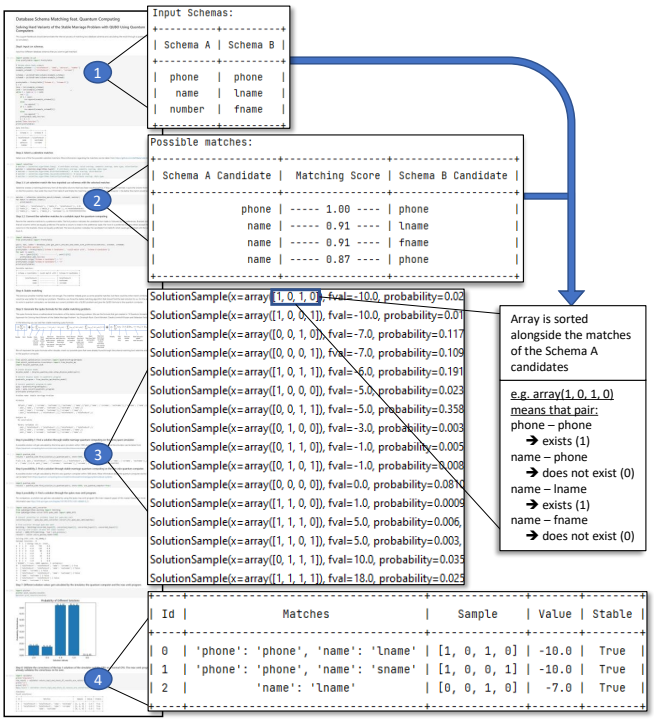


Figure 3: Screenshots of our interactive notebook.

ACKNOWLEDGMENTS

We thank Meike Klettke, Uta Störl, and Markus Zajac for their feedback on earlier versions of this article.

REFERENCES

[1] Zohra Bellahsene, Angela Bonifati, and Erhard Rahm (Eds.). 2011. *Schema Matching and Mapping*. Springer. https://doi.org/10.1007/978-3-642-16518-4

[2] Umut Çalikyilmaz, Sven Groppe, Jinghua Groppe, Tobias Winker, Stefan Prestel, Farida Shagieva, Daanish Arya, Florian Preis, and Le Gruenwald. 2023. Opportunities for Quantum Acceleration of Databases: Optimization of Queries and Transaction Schedules. *Proc. VLDB Endow.* 16, 9 (2023), 2344–2353. https://www.vldb.org/pvldb/vol16/p2344-calikyilmaz.pdf

[3] Tobias Fankhauser, Marc E. Solèr, Rudolf M. Füchslin, and Kurt Stockinger. 2021. Multiple Query Optimization using a Hybrid Approach of Classical and Quantum Computing. *CoRR* abs/2107.10508 (2021). arXiv:2107.10508

[4] Fred W. Glover, Gary A. Kochenberger, Rick Hennig, and Yu Du. 2022. Quantum bridge analytics I: A tutorial on formulating and using QUBO models. *Ann. Oper. Res.* 314, 1 (2022), 141–183. https://doi.org/10.1007/s10479-022-04634-2

[5] Robert W. Irving, David F. Manlove, and Sandy Scott. 2008. The Stable Marriage Problem with Master Preference Lists. *Discrete Appl. Math.* 156, 15 (2008), 2959–2977. https://doi.org/10.1016/j.dam.2008.01.002

[6] Christos Koutras, George Siachamis, Andra Ionescu, Kyriakos Psarakis, Jerry Brons, Marios Fragkoulis, Christoph Lofi, Angela Bonifati, and Asterios Katsifodimos. 2021. Valentine: Evaluating Matching Techniques for Dataset Discovery. In *Proc. ICDE*. 468–479. https://doi.org/10.1109/ICDE51399.2021.00047

[7] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. 2001. Generic Schema Matching with Cupid. In *Proc. VLDB*. 49–58. http://www.vldb.org/conf/2001/P049.pdf

[8] Abraham P. Punnen (Ed.). 2022. *The Quadratic Unconstrained Binary Optimization Problem*. Springer International Publishing. https://doi.org/10.1007/978-3-031-04520-2

[9] Christoph Roch, David Winderl, Claudia Linnhoff-Popien, and Sebastian Feld. 2022. A Quantum Annealing Approach for Solving Hard Variants of the Stable Marriage Problem. In *Proc. Innovations for Community Services (Communications in Computer and Information Science)*, Vol. 1585. Springer, 294–307. https://doi.org/10.1007/978-3-031-06668-9_21

[10] Manuel Schönberger, Stefanie Scherzinger, and Wolfgang Mauerer. 2023. Ready to Leap (by Co-Design)? Join Order Optimisation on Quantum Hardware. *Proc. ACM Manag. Data* 1, 1 (2023), 92:1–92:27. https://doi.org/10.1145/3588946

[11] Maria Schuld and Francesco Petruccione. 2017. Quantum Machine Learning. In *Encyclopedia of Machine Learning and Data Mining*. Springer, 1034–1043. https://doi.org/10.1007/978-1-4899-7687-1_913

[12] Immanuel Trummer and Christoph Koch. 2016. Multiple Query Optimization on the D-Wave 2X Adiabatic Quantum Computer. *Proc. VLDB Endow.* 9, 9 (2016), 648–659. https://doi.org/10.14778/2947618.2947621

[13] Manuela Weigold, Johanna Barzen, Frank Leymann, and Daniel Vietz. 2021. Patterns for Hybrid Quantum Algorithms. In *Proc. SummerSOC*, Vol. 1429. 34–51. https://doi.org/10.1007/978-3-030-87568-8_2

[2] https://quantum-computing.ibm.com/

[3] https://www.dwaves.com/leap/