© 2020 by the authors; licensee RonPub, Lübeck, Germany. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution license (http://creativecommons.org/licenses/by/4.0/).

Open Access

Open Journal of Cloud Computing (OJCC)
Volume 7, Issue 1, 2020
http://www.ronpub.com/ojcc
ISSN 2199-1987

Hardware Accelerating the Optimization of Transaction Schedules via Quantum Annealing by Avoiding Blocking

Tim Bittner, Sven Groppe

Institute of Information Systems (IFIS), University of Lübeck, Ratzeburger Allee 160, D-23562 Lübeck, Germany,
tim.bittner96@gmx.de, groppe@ifis.uni-luebeck.de

Abstract

The isolation property of database theory guarantees to avoid problems of not synchronized parallel execution of several transactions. In this paper we propose an algorithm for an optimal transaction schedule for the different cores of a multi-core CPU with minimal execution time ensuring the isolation property. Optimizing the transaction schedule is a combinatorial problem, which is ideal to be solved by quantum annealers as special form of quantum computers. In our contribution we show how to transform an instance of the transaction schedule problem into a formula that is accepted by quantum annealers including a proof of validity and optimality of the obtained result. Furthermore, we analyze the number of required qubits and the preprocessing time, and introduce an approach for caching formulas as result of preprocessing for the purpose of reducing the preprocessing time. In an experimental evaluation, the runtime on a quantum annealer outperforms the runtime of traditional algorithms to solve combinatorial problems like simulated annealing already for small problem sizes.

Type of Paper and Keywords

Regular research paper: *Quantum computing, quantum annealing, transaction processing, synchronization, 2-phase-locking, database, schedule, D-Wave*

1 Motivation

Transactions are a basic concept of databases: A transaction is a series of operations, carried out by a single user or application program, which reads or updates the contents of a given database. Transactions of several users and applications are typically processed in parallel for optimal performance. Database management systems apply sophisticated methods to avoid problems of not synchronized parallel execution of several transactions by guaranteeing the so called *isolation* property, which requires concurrently executed transactions to not influence each other.

To ensure the isolation property, conflicts between transactions must be dealt with. The most widely approach to dealing with the conflicts that arise is the use of locks: Each transaction acquires a lock for an object before access and releases it after access, which prevents concurrent access to an object. If a transaction wants to acquire a lock held by another transaction, then this transactions is blocked until the lock is released. Hence there is some delay of executing a set of transactions with conflicts due to blocking.

In this paper we deal with optimizing the transaction schedule, which determines the order of parallel executions of transactions on different cores of a multi-core CPU, such that delays are avoided by running only transactions in parallel, which do not block

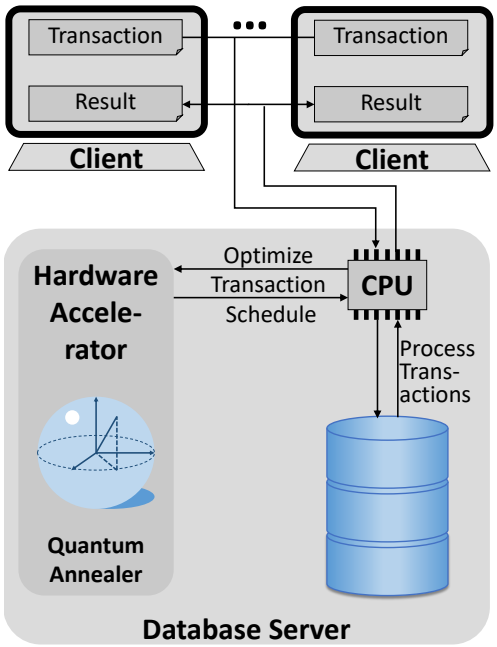


Figure 1: Hardware accelerating transaction schedules via quantum annealing

each other. The problem of finding an optimal transaction schedule is basically a job shop scheduling problem, which additionally needs to consider conflicts between transactions[1]. Note that the job shop scheduling problem is among the hardest combinatorial optimization problems [16], such that utilizing hardware acceleration [20] of the optimization of transaction schedules is a key technology for optimizing transaction schedules. Besides massive parallel hardware like GPUs and FPGAs, hardware acceleration includes quantum computing with special forms like quantum annealing solving quadratic unconstrained binary optimization (QUBO) problems [10, 17]. After preprocessing of a given problem like the optimization of transaction schedules by reformulating it to a QUBO problem, quantum annealers solve the problem in constant time in contrast to other hardware accelerators [19]. Please see Figure 1 for our proposed approach to utilize quantum annealing as hardware accelerator for optimizing transaction schedules.

Our main contributions are

- proposing a formula runnable on quantum annealers for an optimal scheduling of transactions synchronized by 2-phase-locking protocol with preclaiming of all locks at begin of transaction and holding all locks until end of transaction,
- proof of validity and optimality of the proposed approach,
- proposing a cache of precomputed formulas in order to speed up the preprocessing time for generating the formulas for quantum annealing,
- a complexity analysis of the preprocessing time, the number of possible keys for a cache of precomputed formulas and the number of required qubits, and
- an extensive experimental evaluation comparing the runtimes of quantum annealers with the simulated annealing algorithm.

This paper is an extended paper of [6], which extends [6] by detailing more about the basics, providing the proofs of validity and optimality, discussing approaches to reduce the preprocessing times by caching parts of the formulas to be generated, analyzing the influence of the number of required variables in the formulas on the runtimes and discussing further related work.

We introduce the basics about transactions, transaction management and quantum computers in Section 2. Section 3 covers the main part of the work, the formal model for the problem and the transformation from this model into a formula suitable for a quantum annealer including proofs of validity and optimality and an approach for caching the generated formulas. We provide a short insight into the implementation and the used software in Section 4. We estimate the number of required qubits, and analyze the complexity of preprocessing, the number of possible keys for a cache of precomputed formulas and experiments on a quantum annealer in Section 5. We finally summarize and provide a small outlook for future work in Section 7.

2 Basics

This section is dedicated to the basics of transaction management (see Section 2.1) and quantum computers (see Section 2.2) with special focus on quantum annealing.

2.1 Transaction Management

A **transaction** $t =< s_1, ..., s_n >$ of length n is a series of operations s_i, carried out by a single user or application program, which reads or updates the contents of the database [8]. Each operation $s_i = a_i(e_i)$ consists of the type of access $a_i \in \{r, w\}$, where r represents a read access and w a write access, and the object e_i to be accessed.

For example, the transaction $t =< r(A), w(A), r(B), w(B) >$ is of length $|t| = 4$.

Since transactions often need to access a database at the same time and thereby often reference the same objects, transactions in a database system are required

[1] We can actually reduce the job shop scheduling problem to the transaction schedule problem by considering the jobs as transactions, which do not have any conflicts between each other.

to fulfill the so-called ACID properties [8]. The focus here is on the fulfillment of the (I)solation property, which guarantees to avoid problems of unsynchronized parallel execution of several transactions and requires concurrently executed transactions to not influence each other.

2.1.1 Conflict Management

To ensure the isolation property, conflicts between transactions must be dealt with. The simplest strategy to deal with conflicts would be to execute the transactions serially. Since this is too slow for operational systems and also many transactions do not conflict with each other at all, in practice transactions are executed in parallel. There are various approaches to dealing with the conflicts that arise, one of them is the use of locks. For this purpose, each transaction acquires a lock for an object before access and releases it after access, thus preventing concurrent access to an object.

The **two-phase-locking protocol** (see Figure 2) is a locking protocol that requires each transaction consisting of two subsequent phases, the locking phase and the release phase. During the locking phase, the transaction may acquire locks but not release them, whereas the release phase requires the release of previously required locks. If a transaction has released a lock, it may not acquire any new ones. There is a distinction between *read locks* (also called *shared locks*) and *write locks* (also called *exclusive locks*). Variants of the protocol include the conservative two-phase-locking protocol and the strict two-phase-locking protocol. The conservative variant (*preclaiming*) requires that all locks that are required during a transaction are acquired before the transaction is started.[2] The strict variant holds all locks until the end of a transaction, which avoids *cascading aborts* occurring in case of so called *dirty reads* of objects written by transactions, which are later aborted resulting in an abort of the current transaction as well. In this contribution, we consider the combination of the conservative and the strict two-phase-locking protocol in our transaction model, which results in a serial execution of all transactions that access the same objects.

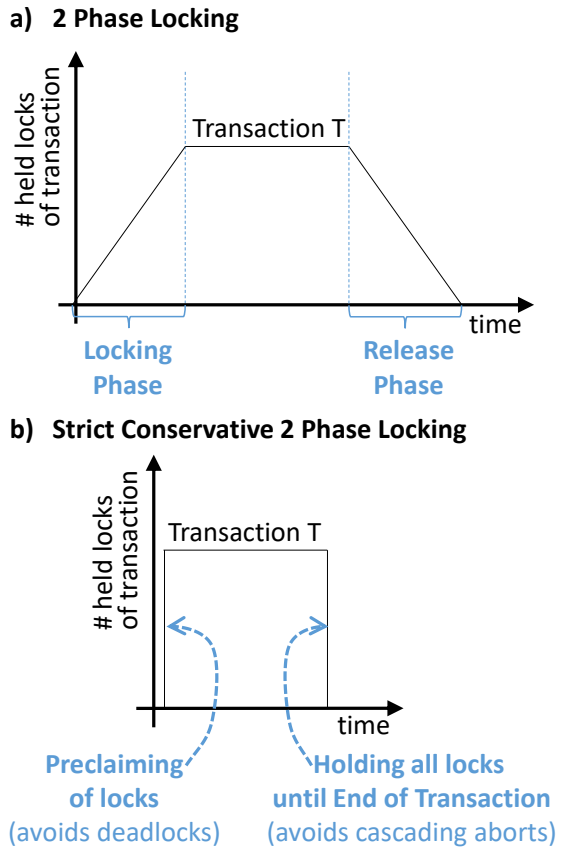


Figure 2: a) Two-phase-locking protocol with locking phase and release phase, and b) the strict conservative two-phase locking protocol

2.1.2 Conflicts

Let T be the set of transactions, D be the set of data objects. Two transactions $i \in T$ and $j \in T$ are in conflict with each other if there exists two operations of these transactions being in conflict with each other. Two operations $a_i(e) \in i$ and $a_j(e) \in j$ of the transactions i and j are in conflict with each other if they access the same object $e \in D$ and at least one of the operations is writing the object:

$$a_i(e) \text{ in conflict with } a'_j(e) \text{ if}$$

$$\exists i, j \in T, e \in D, a_i(e) \in i, a'_j(e) \in j :$$

$$i \neq j \wedge (a_i = w \vee a'_j = w)$$

We assume that accesses to the same object $e \in D$ are serialized and we use the notation $a_i(e) \rightarrow a'_j(e)$ in order to denote that the operation $a_i(e)$ is executed before $a'_j(e)$. The isolation property is therefore fulfilled if all conflict operations of conflicting transactions $i \in T$ and $j \in T$ $(i \neq j)$ are processed in the same order of transactions:

$$\Big(\forall a_i(e) \in i, a'_j(e) \in j :$$

$$(a_i = w \vee a'_j = w) \Rightarrow a_i(e) \rightarrow a'_j(e)\Big)$$

[2] Please note that for those transactions, for which the required locks are not known before processing, the required locks can be determined by an additional phase before transaction processing. The contribution in [41] describes such an approach which can be also applied in our scenario.

$$\vee$$

$$\Big(\forall a_i(e) \in i, a'_j(e) \in j :$$

$$(a_i = w \vee a'_j = w) \Rightarrow a'_j(e) \rightarrow a_i(e)\Big)$$

The two-phase-locking protocol fulfills the isolation property by guaranteeing that each object is continuously locked from its first to last access and during the entire processing. An operation $a_i(e)$ of a transaction i is suspended if another transaction j ($i \neq j$) already holds a lock to the object e (and the held lock or the lock to be acquired is an exclusive lock). In this way it is guaranteed that all conflicting operations of transaction j are executed before those operations of transaction i. When using preclaiming of locks, according to [41] there are no real limits (by determining the required locks before transaction processing in an additional phase), while the occurrence of deadlocks is eliminated, where transactions wait for releasing the locks of each other.

2.2 Quantum Computer

Quantum computers are computers that are not based on classical mechanics, instead they exploit the effects of quantum mechanics.

2.2.1 Basic Idea

Quantum mechanics describes the states and behavior of particles that are smaller than the size of an atom and do not follow the laws of classical physics. At this scale, there occur effects that the quantum computer makes use of, especially the principle of superposition and that of quantum entanglement. The quantum computer uses qubits, which can take on 2 states simultaneously due to the principle of superposition. While a bit can assume the state 0 or 1, a qubit assumes the states 0 and 1 simultaneously. If a measurement of the state is made, the qubit changes to one of the two states. Both states have relative probabilities with which they are assumed in the measurement. The principle of quantum entanglement enables the mutual influence of qubits, since entangled qubits mutually influence their probabilities. Imagining the principle of superposition as a special form of parallel computing opens up a new world of computation beyond polynomial time and allows in theory an exponential speedup compared to classical computers.

2.2.2 Quantum Computing

Quantum computing finds desired solutions of a problem by clever manipulation of single qubits as well as entangled qubits. For the purpose of manipulating qubits, gates provide elementary operations on one or two qubits. For example, the Hadamard-Gate puts one qubit into superposition and a Controlled-NOT(CNOT)-Gate inverts a second qubit depending on the first [3]. A quantum computer is thus able to execute quantum algorithms like Shor's algorithm for factorizing large numbers [38] or Grover's algorithm for searching in huge unsorted databases [22, 23].

2.2.3 Quantum Annealers

Quantum annealers are a special form of quantum computers [14, 33], which are designed to solve hard optimization problems, but cannot execute quantum algorithms like Shor's algorithm. Quantum annealers find the global minimum of a given objective function by transforming a simplified objective function with known global minimum into the objective function of interest, so that the quantum annealer always remains in the state of the global minimum, which represents the desired result. The most widely-known quantum annealers are manufactured by the canadian company D-Wave [9], which produced the first commercial quantum computers.

Input format: Quantum annealers have very limited input formats, such that all problems have to be transformed into this input format before being solved and transformed back after annealing. Quantum annealers solve quadratic unconstrained binary optimization (QUBO) problems [10, 17], which belong to the class of NP-hard problems. A QUBO-problem is defined by N weighted binary variables $X_1, ..., X_N \in \{0, 1\}$, either as linear or quadratic term:

$$\sum_{0<i\leq N} w_i X_i + \sum_{i\leq j\leq N} w_{ij} X_i X_j, \text{ where } w_i, w_{ij} \in \mathbb{R}$$

The quantum annealer transforms these weightings into energy levels of single qubits or between two qubits, and by minimizing the energy level the quantum annealer finds a minimization of the objective function, in other words a variable assignment for the QUBO-problem. The idea of quantum annealing is very close to the simulated annealing approach [36] on traditional computers (see Figure 3).

3 MODELLING

In this section, we describe the formal model of instances of the considered scheduling problem (see Section 3.1) and afterwards formulate the scheduling problem as QUBO-problems (see Section 3.2) for valid solutions (see Section 3.3) and optimal solutions (see

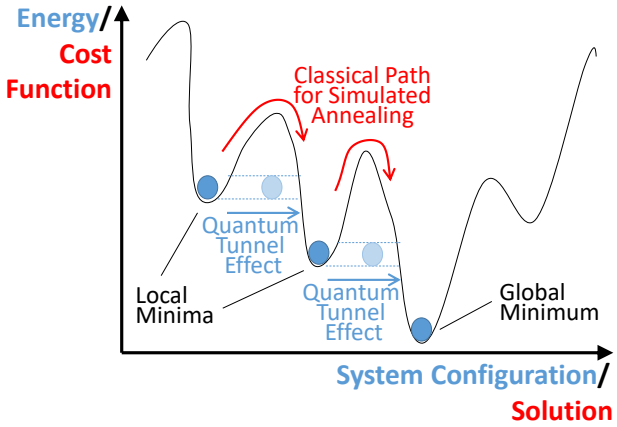


Figure 3: Simulated annealing (sa) [36] versus quantum annealing (qa) to reach a global minimum (simplified discussion): Quantum tunneling driven by an external magnetic field "passes" high energy/cost function "peaks" instead of climbing them as in sa. Larger tunnels can be passed (qa) and bigger peaks climbed (sa) with a high magnetic field (qa) and a high temperature (sa). Then lower magnetic fields (qa) and lower temperatures (sa) hinders to get out of a valley of the energy (qa) and cost function (sa) respectively with a global minimum. As long as the function fits into the system configuration size, qa determines the minimum in constant time in contrast to sa, which takes longer times for more complex functions.

Section 3.3.1), and optimize the total solution further (see Section 3.3.2). Furthermore, we prove our proposed formulation of optimizing transaction schedules as QUBO-problems in Section 3.4. We finally propose a cache for the formulas to be generated in order to speed up preprocessing time in Section 3.5

3.1 Formal Model

The goal of this optimization is to reduce, or in the best case completely eliminate, the waiting times of transactions that use the two-phase-locking protocol. A transaction must wait whenever it requests a lock, but this lock is held by another transaction. In this case, one transaction blocks the other and only one of the two can be executed at a time. For simplicity of presentation, we consider the conservative and strict two-phase-locking protocol[3]. We assume that k machines are available for processing n transactions. Thus k transactions can run parallel to each other, if they do not block each other. We hence consider the scheduling problem, where n transactions have to be distributed to k machines, taking into account that two transactions blocking each other are never executed simultaneously. Thereby the maximum execution time R (see Section 3.3.3) should be minimized over all machines. An instance of this scheduling problem consists of a set T of transactions with $|T| = n$, a set M of machines with $|M| = k$ (see Figure 7) and a set $O \subseteq T \times T$ of blocking transactions (see Figure 6). Each transaction $t_i \in T$ has a certain length l_i (see Figure 4) and thus an upper bound $r_i = R - l_i$ for its start time (see Figure 5).

3.1.1 Running Examples

This paper contains two running examples:

Example E_1: The first running example E_1 deals with a rather complex scenario and contains 8 transactions for illustrating the model in Figure 4, 5, 6, 7, 8 and 10.

Example E_2: The second running example E_2 contains a simpler configuration of transactions, but is especially designed for illustrating the generated formula to be minimized by a quantum annealer. In E_2, three transactions are to be distributed on two machines. The first transaction has a length of 2, the other two a length of 1. Furthermore, the second and third transactions are blocking each other in their execution. We use $R = 2$ for the maximum execution time[4].

Overall we have the following configuration for the running example E_2:

$$
\begin{aligned}
&R = 2\\
&T = \{t_1, t_2, t_3\}\ with\ |T| = 3\\
&M = \{m_1, m_2\}\ with\ |M| = 2\\
&O = \{(t_2, t_3)\}\\
&l_1 = 2, l_2 = 1, l_3 = 1\\
&r_1 = 0, r_2 = 1, r_3 = 1
\end{aligned}
$$

3.2 Formulation as QUBO-Problem

A QUBO-problem consists of binary variables that occur weighted in linear or quadratic terms. The binary variables

$$X_{i,j,s} \text{ for } 1 \leq i \leq n, 1 \leq j \leq k, 0 \leq s \leq r_i$$

contain the value 1 if transaction t_i is started at time s on machine m_j, otherwise 0. For a valid schedule n of the variables must hold the value 1 and all others 0, so that

[3] As discussed in Section 2.1.1 and by using the approach in [41], transactions can be also processed, which do not have a fixed set of required locks, by introducing an additional processing phase before transaction start.

[4] We have to use a reasonable maximum execution time for minimizing the formulas proposed in the following sections and describe an approach for a clever determination of R in Section 3.3.3.

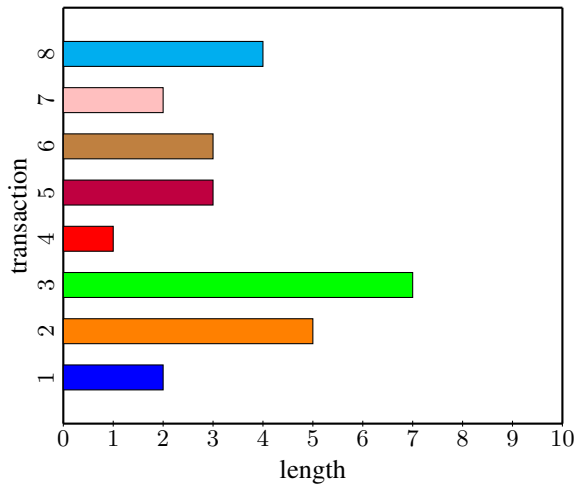


Figure 4: $n = 8$ **transactions with their respective lengths (Example** E_1**)**

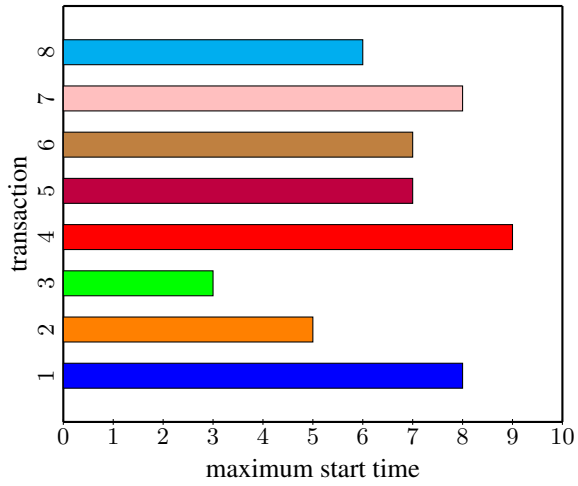


Figure 5: $n = 8$ **transactions with their respective maximum start times (Example** E_1**)**

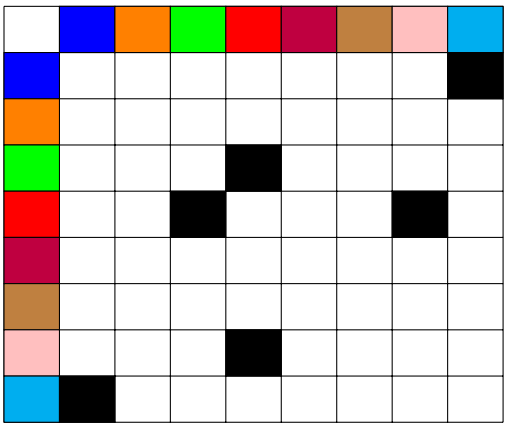


Figure 6: Black fields indicate blocking transactions (Example E_1**)**

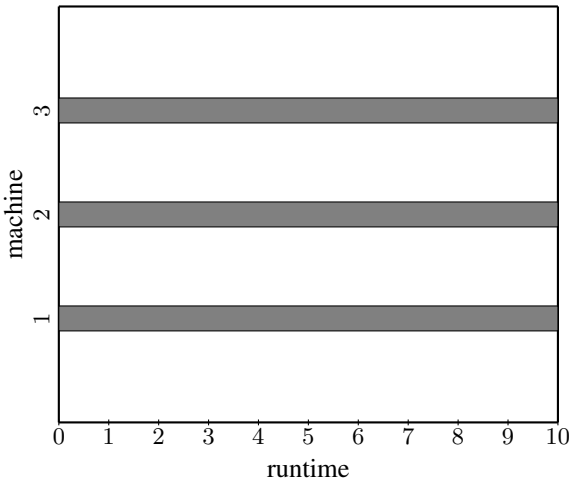


Figure 7: Transactions are scheduled on three machines, the maximum runtime here is $R = 10$ **(Example** E_1**)**

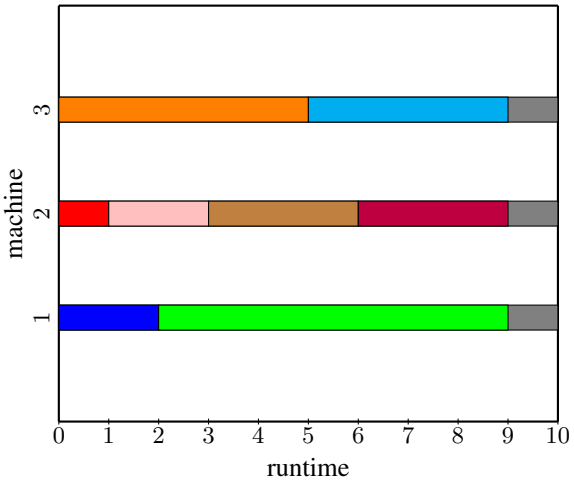


Figure 8: Binary variables set for this schedule: $X_{110}, X_{312}, X_{420}, X_{721}, X_{623}, X_{526}, X_{230}, X_{835}$ **(Example** E_1**)**

for each transaction the respective start time is expressed. The solution to this problem is the distribution of the transactions to the different machines (see Figure 8).

3.3 Valid Solution

Constraints for a valid schedule are now formulated in such a way that the resulting formula takes on high values whenever the constraints for validity are not satisfied and low values whenever they are satisfied. Three constraints must be formulated so that minimizing the formula guarantees a valid schedule:

A: Each transaction starts exactly once,

B: two or more transactions cannot be executed at the same time on the same machine, and
C: transactions that block each other cannot be executed at the same time.

To ensure that each transaction starts exactly once, it may only start once across all machines and possible start times:

$$A = \underbrace{\sum_{i=1}^{n}}_{\text{transactions}} \Big(\underbrace{\sum_{j=1}^{k}}_{\text{machines}} \underbrace{\sum_{s=0}^{r_i}}_{\text{start times}} X_{i,j,s} - 1 \Big)^2$$

For the example configuration E_2 of Section 3.1.1:

$$\begin{aligned} A = &(X_{1,1,0} + X_{1,2,0} - 1)^2 \\ &+ (X_{2,1,0} + X_{2,1,1} + X_{2,2,0} + X_{2,2,1} - 1)^2 \\ &+ (X_{3,1,0} + X_{3,1,1} + X_{3,2,0} + X_{3,2,1} - 1)^2 \end{aligned}$$

A holds the value 0 if each transaction starts exactly once, a higher one otherwise. If for a transaction the rear term in brackets is multiplied with $k \cdot r_i$ variables, only terms with variables appear and offsets of $1 = -1 \cdot -1$. Hence we obtain n offsets for n variables. Since a QUBO-problem consists solely of linear and quadratic terms, the offset has no place there and is ignored. Hence during running the problem on a quantum annealer, A finally retrieves only the value $-n$ and the offset of n is ignored. After the final formula has been minimized and a variable assignment has been found, the offset can of course be added again, but it does not play any role in solving the problem.

To ensure that transactions do not run at the same time on the same machine, the runtime is calculated in the following way using the start time and the length of the transactions:

$$B = \underbrace{\sum_{j=1}^{k}}_{\text{machines}} \overbrace{\sum_{i_1=1}^{n-1}}^{\text{transactions without } t_n} \underbrace{\sum_{s_1=0}^{r_{i_1}}}_{\text{start times}} \overbrace{\sum_{i_2=i_1+1}^{n}}^{\text{remaining transactions}} \underbrace{\sum_{s_2=q}^{p}}_{\text{invalid start times}} X_{i_1,j,s_1} X_{i_2,j,s_2}$$

$$\text{for } q = \max\{0, s_1 - l_{i_2} + 1\}, p = \min\{s_1 + l_{i_1}, r_{i_2}\}$$

For the example configuration E_2 of Section 3.1.1:

$$\begin{aligned} B = &X_{1,1,0}X_{2,1,0} + X_{1,1,0}X_{2,1,1} + X_{1,1,0}X_{3,1,0} \\ &+ X_{1,1,0}X_{3,1,1} + X_{2,1,0}X_{3,1,0} + X_{2,1,1}X_{3,1,1} \\ &+ X_{1,2,0}X_{2,2,0} + X_{1,2,0}X_{2,2,1} + X_{1,2,0}X_{3,2,0} \\ &+ X_{1,2,0}X_{3,2,1} + X_{2,2,0}X_{3,2,0} + X_{2,2,1}X_{3,2,1} \end{aligned}$$

If no transactions are running at the same time on the same machine, at most one of the variables pairwise takes the value 1 and B overall takes the value 0. For each machine, transaction and associated start time, all invalid start times of all remaining transactions are calculated and this combination is added to the formula. B reaches the maximum, i.e., exactly the value of all sums of the formula, in the case of pairwise both variables take the value 1. The values q and p delimit the range in which two transactions overlap in their runtimes for certain start times (see Figure 9).

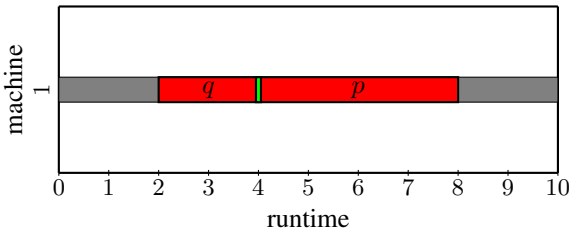


Figure 9: The green line represents a starting time 4 of a transaction of length 4 on a machine. The value q indicates that a second transaction of length 3 must not be started in the red area in front of the green line, because otherwise the runtimes will overlap. The value p expresses the time until when the first transaction runs, so that no other transactions may be started in the second red area either.

To avoid transactions that block each other being executed at the same time, similar constraints are established as for B:

$$C = \underbrace{\sum_{\{t_{i_1}, t_{i_2}\} \in O}}_{\text{blocking transactions}} \overbrace{\sum_{j_1=1}^{k}}^{\text{machines}} \underbrace{\sum_{s_1=0}^{r_{i_1}}}_{\text{start times}} \overbrace{\sum_{j_2 \in J}}^{\text{remaining machines}} \underbrace{\sum_{s_2=q}^{p}}_{\text{invalid start times}} X_{i_1,j_1,s_1} X_{i_2,j_2,s_2}$$

$$\text{for } J = \{1, ..., k\} \setminus \{j_1\},$$

$$q = \max\{0, s_1 - l_{i_2} + 1\}, p = \min\{s_1 + l_{i_1}, r_{i_2}\}$$

For the example configuration E_2 of Section 3.1.1:

$$\begin{aligned} C = &X_{2,1,0}X_{3,2,0} + X_{2,1,1}X_{3,2,1} \\ &+ X_{2,2,0}X_{3,1,0} + X_{2,2,1}X_{3,1,1} \end{aligned}$$

Similar to B, C takes the value 0, if no transactions that block each other are executed at the same time and so again only one of the variables pairwise takes the value 1. For all pairs of blocking transactions, the different invalid start combinations are determined and added to the formula. C reaches the maximum, i.e., exactly the value of all sums of the formula, in the case of pairwise both variables take the value 1.

3.3.1 Optimal Solution

For an optimal solution, the variables are now weighted so that minimizing the formula requires the earliest possible start time for each transaction. Therefore the calculated end time is weighted so that its weight exceeds the sum of all weights of all machines for each smaller end time. This is necessary because otherwise many short transactions that are started early would keep the weight lower than long transactions. As a result, long transactions with a different weight distribution would only be started at the end and thus the actual goal of minimizing the maximum execution time would not be achieved. The weight for an end time $s + l_i$ looks like this (see Figure 10):

$$w_{s+l_i} = \frac{(k+1)^{s+l_i-1}}{(k+1)^R}$$

By dividing by $(k+1)^R$, $w_{s+l_i} \in (0,1)$ applies, which requires that a valid solution is preferred to an optimal one, since the weights in A, B and C always assume the value $|1|$ or higher. This results in the following formula:

$$D = \sum_{i=1}^{n} \sum_{j=1}^{k} \sum_{s=0}^{r_i} w_{s+l_i} X_{i,j,s}$$

For the example configuration E_2 of Section 3.1.1:

$$\begin{aligned} D = &\frac{3}{9}X_{1,1,0} + \frac{3}{9}X_{1,2,0} \\ &+ \frac{1}{9}X_{2,1,0} + \frac{3}{9}X_{2,1,1} + \frac{1}{9}X_{2,2,0} + \frac{3}{9}X_{2,2,1} \\ &+ \frac{1}{9}X_{3,1,0} + \frac{3}{9}X_{3,1,1} + \frac{1}{9}X_{3,2,0} + \frac{3}{9}X_{3,2,1} \end{aligned}$$

3.3.2 Total Solution

Together A, B, C and D form the actual formula, the QUBO-problem, the solution of which is the solution of the actual problem, a valid and optimal distribution of the transactions to the different machines, so that if possible there are no idle times and if possible each transaction has no waiting time:

$$P = A + B + C + D$$

A valid solution always assumes the value $-n$ for $A + B + C$ and is increased by the value of D whenever an optimal solution is reached.

For the example configuration E_2 of Section 3.1.1, the formula P is minimized if all three constraints for a valid schedule are satisfied and the weights across all variables are minimal. In this example, the formula is minimized by four different variable assignments, all of

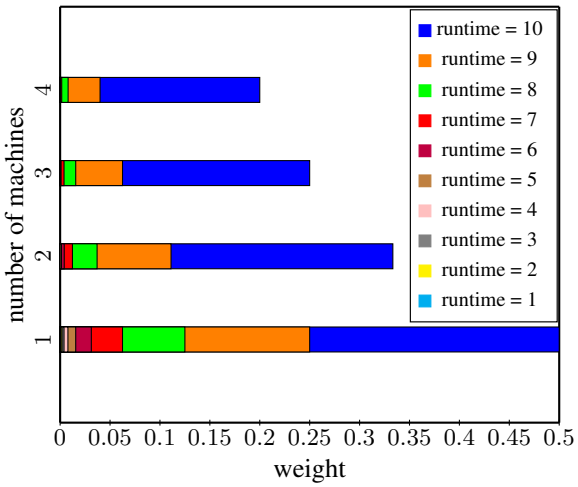


Figure 10: Display of weights for various numbers of machines and end times (Example E_1)

which represent a valid schedule (here, only the variables with the value 1 are specified, the rest take the value 0 accordingly):

$$\begin{aligned} &X_{1,1,0}, X_{2,2,0}, X_{3,2,1} \\ &X_{1,1,0}, X_{2,2,1}, X_{3,2,0} \\ &X_{1,2,0}, X_{2,1,0}, X_{3,1,1} \\ &X_{1,2,0}, X_{2,1,1}, X_{3,1,0} \end{aligned}$$

The solution represents which transaction is started when and on which machine. P assumes the following value for all four different variable assignments:

$$P = A + B + C + D = -3 + 0 + 0 + \frac{7}{9} = -2\frac{2}{9}$$

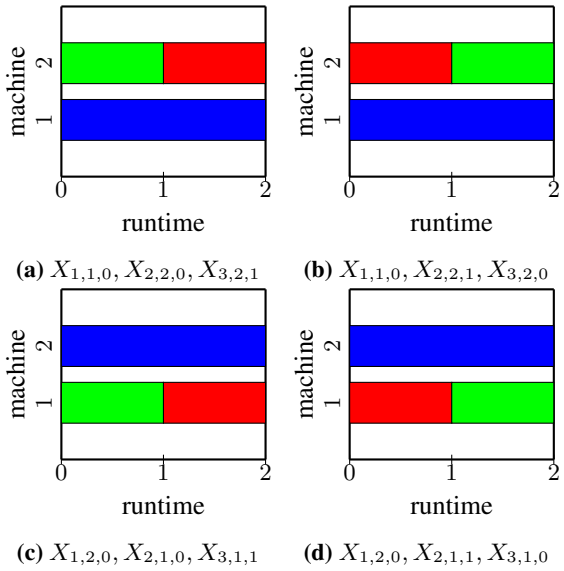
If the offset is added (optional), then the result is:

$$P = A + B + C + D = -3 + 0 + 0 + \frac{7}{9} + 3 = \frac{7}{9}$$

Both solutions are correct, for the purpose of simplicity the offset is omitted here. The solution is illustrated in Figure 11.

3.3.3 Minimization of Execution Time

The (maximal) execution time R is a critical parameter during generation of the formula and its solving: If R is too large, then we need too many unnecessary variables, which increases the problem size and hence also the time of preprocessing (but we still retrieve the correct solution). If R is too small, then we do not obtain any solution. In the latter case, we need to rerun the formula generation and solving it on a quantum annealer with an increased R.



(a) $X_{1,1,0}, X_{2,2,0}, X_{3,2,1}$ **(b)** $X_{1,1,0}, X_{2,2,1}, X_{3,2,0}$

(c) $X_{1,2,0}, X_{2,1,0}, X_{3,1,1}$ **(d)** $X_{1,2,0}, X_{2,1,1}, X_{3,1,0}$

Figure 11: Different Schedules (transaction 1 in blue, transaction 2 in green and transaction 3 in red) for example configuration E_2

The execution time R should therefore be reasonably estimated in advance to avoid unnecessarily many runs on a quantum annealer. Since a quantum annealer executes many runs of the same problem, by slowly increasing the previously estimated execution time, the minimum execution time allowed by a valid schedule can be determined. The lower bound of the execution time is obviously the sum of all lengths divided by the number of machines:

$$R = \frac{\sum_{i=1}^{n} l_i}{k}$$

This value can be estimated even better when considering the length of transactions. Two transactions with lengths 1 and 9 cannot be executed by two machines in time $R = (1 + 9)/2 = 5$. A more reasonable estimation is:

$$R = \max\{\frac{\sum_{i=1}^{n} l_i}{k}, \max_{i \in \{1,\ldots,n\}} l_i\}$$

For the example configuration E_2 of Section 3.1.1:

$$R = \max\{\frac{4}{2}, 2\} = 2$$

If there is one very long transaction and many short ones, the maximum execution time is obviously defined by the length of the very long transaction. By slowly increasing R the minimum execution time and a suitable schedule are found. A valid schedule is identified by its value, since this value does not differ significantly from $-n$, but is significantly larger for an invalid schedule.

3.4 Proofs

We prove the correctness of the formula P described in Section 3.2 in two steps: We first show that a valid variable assignment minimizes the formula. Then we reason about an optimal solution among all valid solutions minimizes the formula.

3.4.1 Proof of Validity

The proof of validity consists of four steps:

- The formula P is minimized if each transaction is started a maximum of once,
- P is minimized if each transaction is started at least once,
- P is minimized if transactions are not executed at the same time on the same machine, and
- P is minimized if transactions that block each other are not executed at the same time.

Lemma 1. *The formula P is minimized if each transaction is started a maximum of once.*

Proof. Suppose the formula would be minimized if a transaction is started twice, so if $X_{i,j_1,s_1} = X_{i,j_2,s_2} = 1$ for any transaction t_i. If one of the two variables is now set to the value 0, the value of A decreases by 1, the values of B and C remain unchanged and the value of D decreases by at least $\frac{1}{(k+1)^R}$. Thus, if one of the two variables is set to the value 0, the value of the formula decreases, contradicting the initial assumption. □

Lemma 2. *The formula P is minimized if each transaction is started at least once.*

Proof. Assuming the formula would be minimized if a transaction is not started at all, so $\sum_{j=1}^{k} \sum_{s=0}^{r_i} X_{i,j,s} = 0$ for any transaction t_i. If one of the variables is now set to the value 1, the value of A decreases by 1, the values of B and C remain unchanged and the value of D increases by a maximum of $\frac{1}{k+1}$, i.e., by a maximum of $\frac{1}{2}$. Thus, if one of the variables is set to the value 1, the value of the formula decreases, contradicting the initial assumption. □

Lemma 3. *The formula P is minimized if transactions are not executed at the same time on the same machine.*

Proof. All transactions executed at the same time on the same machine increase the value of B and thus the value of the formula by 1. This minimizes the formula if transactions are not executed at the same time on the same machine. □

Lemma 4. *The formula P is minimized if transactions that block each other are not executed at the same time.*

Proof. All transactions that block each other and are executed at the same time increase the value of C and thus the value of the formula by 1. This minimizes the formula when transactions that block each other are not executed at the same time. □

Theorem 1. *A valid variable assignment minimizes the formula P.*

Proof. This results from lemmas 1, 2, 3 and 4. □

3.4.2 Proof of Optimality

Lemma 5. *A minimum execution time minimizes formula D.*

Proof. Assuming R is the minimum execution time and D would be minimized by a non minimum execution time $R+1$. Hence a weight w_{R+1} must appear in D. $w_{R+1} > \sum_{j=1}^{k} \sum_{s=0}^{R} w_s$ contradicts the assumption. Analogous remarks apply for larger execution times $R+a$ with $a > 1$. □

Theorem 2. *An optimal solution among all valid solutions minimizes the formula P.*

Proof. A, B and C take the same value for each valid solution. Hence D alone decides the selection among all valid solutions. Since D requires for each transaction the earliest possible end time and considering also lemma 5, a minimal solution of P represents an optimal transaction schedule with the earliest possible end time of all transactions. □

3.5 Caching Formulas to Reduce the Preprocessing Time

The preprocessing time to generate the formulas for the quantum annealer depends on the length of the generated formula. We will show in Section 5.1.1 that this length and hence the preprocessing time is in $O((n \cdot k \cdot R)^2)$. We discuss in this section how to reduce this preprocessing time by caching and reusing generated formulas.

We propose to optimize the transaction schedule of batches. By using pipelining we can optimize the transaction schedule already of the next batch during the processing of the current batch of transactions (see Figure 12).

In order to increase reusability of the generated formulas and hence reduce cache misses, we assume the following parameters to be fixed:

- the number k of machines is typically fixed, because the system does not change during runtime, and
- the number n of transactions is fixed whenever batches of transactions of the same sizes are processed.

We observe the following facts:

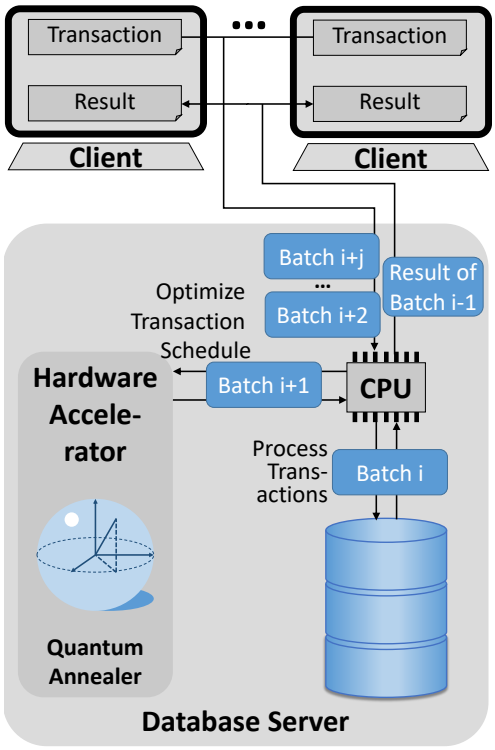


Figure 12: Pipelining the optimization of transaction schedules and the processing of transaction batches: The batches $i+2$ to $i+j$ of transactions are in the queue for optimizing their schedule and processing, the schedule of batch $i+1$ is currently optimized by the quantum annealer, batch i is currently processed and the results of batch $i-1$ are being transmitted to the clients.

- The (maximal) execution time R depends on the lengths of the transactions (see Section 3.3.3),
- hence the upper bounds of start times r_i, ..., r_n depends also on the lengths of transactions,
- the formulas A, B and D depend on the fixed parameters k and n, and on the lengths of the transactions, and
- the formula C is a sum of sub-formulas depending on the fixed parameter k and the lengths of blocking transactions as well as the identifiers of blocking transactions.

Hence we propose to use the lengths of transactions as key to index the formulas A, B and D in a cache. In order to achieve unique keys and avoid symmetric cases, we sort the lengths of transactions and use this unique representation as key. The transactions are renumbered according to the position in the sorted list of their lengths.

For looking up the formulas A, B and D in the cache, the keys are $< 1, 2, 2, 3, 3, 4, 5, 6 >$ for the running

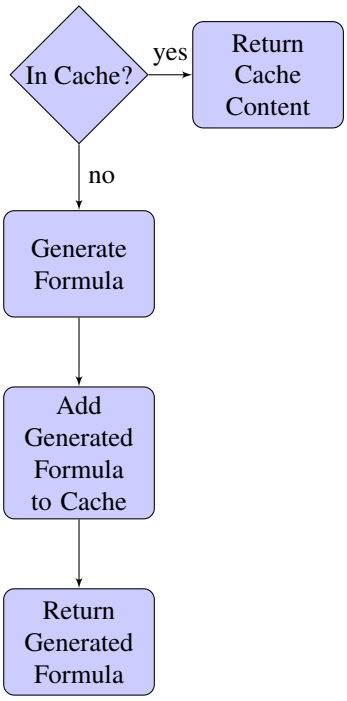


Figure 13: Caching generated formulas

example E_1 with transactions t_1: ■, t_2: ■, t_3: ■, t_4: ■, t_5: ■, t_6: ■, t_7: ■ and t_8: ■ (see Figure 4) and $< 1, 1, 2 >$ for the running example E_2.

For the formula C we propose to index its sub-formulas for each pair of blocking transactions in another cache. Hence the key of this cache is composed of the two lengths of blocking transactions. In order to achieve unique keys and avoid symmetric cases, we sort the lengths of the blocking transaction pairs. For the purpose of a better reusing of the generated sub-formulas and in order to reduce cache misses, we store sub-formulas only for transactions t' representing the shorter transaction and t'' for the longer transaction, but have to rename the binary variables in looked up formulas to the ones of the blocking transactions after cache access.

For looking up the sub-formulas of C, the keys are $< 2, 3 >$, $< 2, 4 >$ and $< 3, 7 >$ for the running example E_1 and $< 1, 1 >$ for the running example E_2.

In the following description, we use the function key to determine the keys in form of a sorted sequence of the lengths of transactions of a transaction configuration E or of a pair of transactions $\{t_{i_1}, t_{i_2}\}$.

Every time we optimize a transaction schedule, we first check our cache of formulas with a given key if the formulas have already been determined before and return the cache content if this is the case (see Figure 13). If they have not been determined before, we compute the formulas, store the result in the cache and return the result. We use the notation $cache_{A+B+D}(key(E))$ for this caching functionality of the sum of the formulas A, B and D for a transaction configuration E. We use the notation $cache_C(key(t', t''))$ for caching the sub-formulas

$$\overbrace{\sum_{j_1=1}^{k}}^{\text{machines}} \underbrace{\sum_{s_1=0}^{r'}}_{\text{start times}} \overbrace{\sum_{j_2 \in J}}^{\text{remaining machines}} \underbrace{\sum_{s_2=q}^{p}}_{\text{invalid start times}} X_{t',j_1,s_1} X_{t'',j_2,s_2}$$

of C for a pair $\{t', t''\}$ of transactions and l' length of t', l'' length of t'', $r' = R - l'$, $r'' = R - l''$, $J = \{1, ..., k\} \setminus \{j_1\}$, $q = \max\{0, s_1 - l'' + 1\}$, $p = \min\{s_1 + l', r''\}$. We use a renaming function $\rho_{t_{i_1} \leftarrow t', t_{i_2} \leftarrow t''}(F)$, which renames all variables in F, which are of the form $X_{t',j,s}$ to $X_{i_1,j,s}$ and $X_{t'',j,s}$ to $X_{i_2,j,s}$.

Hence we alter the computation of P for a transaction configuration E with a set O of blocking transactions in the following way:

$$P = cache_{A+B+D}(key(E)) + \underbrace{\sum_{\{t_{i_1}, t_{i_2}\} \in O}}_{\text{blocking transactions}} \rho_{t_{i_1} \leftarrow t', t_{i_2} \leftarrow t''}(cache_C(key(t_{i_1}, t_{i_2})))$$

For the running example E_1, we compute P according to the following formula:

$$\begin{aligned} P = {} & cache_{A+B+D}(< 1, 2, 2, 3, 3, 4, 5, 6 >) \\ & + \rho_{t_3 \leftarrow t', t_4 \leftarrow t''}(cache_C(< 2, 3 >)) \\ & + \rho_{t_2 \leftarrow t', t_6 \leftarrow t''}(cache_C(< 2, 4 >)) \\ & + \rho_{t_4 \leftarrow t', t_8 \leftarrow t''}(cache_C(< 3, 7 >)) \end{aligned}$$

If the main memory is full up, entries in the cache can be removed to free up space according to least-recently-used or other cache replacement policies. Alternatively a disk-based index (like a B$^+$-tree or LSM tree) can be used as main data structure for the cache in order to cache a larger set of formulas exceeding main memory space.

4 Implementation of the QUBO-Problem

We implement the QUBO-problem described in Section 3.2 by using the Ocean Software of the company D-Wave [10]. The Ocean Software consists of a set of tools that help to formulate a problem for quantum annealers and solves the problem using either a classical computer or a quantum annealer. By only adapting the code for communicating with the quantum annealer, the same code is runnable on a classical computer as well as on a quantum annealer with minimal code modification.

The software provides tools for communication with the quantum annealer, for problem formulation through various constraints, for problem solving through various approaches, for embedding the problem on the quantum annealer and many more. Additionally, the software "PyQUBO" [30] is used, which allows to design QUBO formulations from mathematical expressions. Since both software is developed for Python, the implementation is also written in Python. The packages "dimod" [1] (for supporting QUBO models) and "dwave-neal" [2] (for comparing the runtimes of quantum with those of the simulated annealing) are additionally used. The implementation calculates the QUBO-problem from a list of transactions and their lengths, a number of machines and a list of blocking transactions. The execution time R for these transactions is estimated in advance according to Section 3.3.3. The resulting QUBO-problem is precisely solvable by testing all possible variable assignments and thus those with the lowest total value represent the optimal and valid solution. For determining a solution on a classical computer the Simulated-Annealing-Solver [2], one of the Ocean Software tools, is used. For quantum annealing the D-Wave Quantum-Annealer [11] offered as cloud service is used.

5 EVALUATION

In this section, we analyze the complexity (see Section 5.1) in terms of preprocessing time (see Section 5.1.1), cache sizes (see Section 5.1.2) and required qubits (see Section 5.1.3), and experimental results comparing simulated with quantum annealing (see Section 5.2).

5.1 Formal Analysis

A time analysis for quantum computers as a function of problem variables such as number of transactions or number of machines is generally not possible, since for quantum computers the number of annealing runs as well as the times per annealing run and the times for readout can be determined in advance.

We first determine the number of required qubits and the time of preprocessing in terms of a complexity analysis in Section 5.1.1. The problem sizes remain n for the number of transactions, k for the number of machines and R for the maximum execution time.

5.1.1 Preprocessing Time Without Caching

$O(n \cdot k \cdot R)$ binary variables are used within the formula for distributing transactions to k machines to be minimized. A and D contain $O(n \cdot k \cdot R)$ terms, B contains $O(n^2 \cdot k \cdot R^2)$ terms and C in case all transactions block each other $O((n \cdot k \cdot R)^2)$ terms. Calculating the weights for D is performed in constant time. For the worst case, we overall achieve a quadratic time for preprocessing depending on the problem sizes:

$$O((n \cdot k \cdot R)^2)$$

For cases, where the number of conflicting transactions is at most linear to n (which might be the typical case for transaction workloads in operational systems), the number of terms is dominated by the number of terms in B (assuming $n > k$) and is hence $O(n^2 \cdot k \cdot R^2)$.

5.1.2 Caching

Whenever the formula $A + B + D$ is already given in the cache $cache_{A+B+D}$ and all sub-formulas of C in $cache_C$, then we can determine the formula P in time $O(c + c \cdot |O| \cdot k^2 \cdot R^2)$, where c represents the time of an index access, which is constant for hash tables and quasi-constant (in practice with a logarithmic time in theory) for indices typically used in databases like B^+-tree, LSM tree and similar[5]. The factors $k^2 \cdot R^2$ are due to renaming the binary variables, but may be reduced by implementation tricks. Dependent on the application, $|O|$ is also constant having few entries, is in $O(n)$ whenever each transaction has a constant number of conflicts, or is in $O(n^2)$ in the worst case whenever each transaction is in conflict with (nearly) all the other transactions.

We achieve optimal preprocessing times whenever there are no cache misses, i.e., the caches contain all queried sub-formulas. This is definitely the case for caches filled with formulas for all possible keys.

Let $R_T \leq R$ be an upper bound for the transaction length, i.e., $\forall i \in \{1, ..., n\} : l_i \leq R_T$. Whenever k and n are also fixed, we can determine the number of possible keys in the following way: Due to sorting the lengths of transactions for unique keys, the number of possible keys is a n-combination with repetitions and hence is in

- $O(\binom{R_T+n-1}{n}) = O(\frac{(R_T+n-1)!}{n! \cdot (R_T-1)!})$ for $cache_{A+B+D}$, and in
- $O(\binom{R_T+2-1}{2}) = O(\frac{(R_T+2-1)!}{2! \cdot (R_T-1)!})$ for $cache_C$.

While the number of possible keys is moderate for small R_T and n and hence all formulas can be computed in advance, these numbers are very large for bigger R_T and n. The only feasible approach is to use a cache for bigger R_T and n, such that the formulas for the different keys are computed one after the other and reused for following computations. For example, $R_T \leq 10$ and

[5] However, for cache implementations there is still tool support missing for precompiling only parts of the formulas, which can be combined afterwards quickly for deployment on the quantum annealer.

$n \leq 20$ causes at most $10,015,005$ possible keys, which might be suitable for precomputing all formulas in advance (using a big cluster), while there are above 1 billion possible keys for $R_T = 20$ and $n \geq 15$.

Real-world applications often use some form of transaction templates [34], such that transaction configurations and hence the keys for the caches may often repeat. According to this observation, the cache miss rate will quickly decrease when using a cache for precomputed formulas.

5.1.3 Required Qubits

In quantum annealers from D-Wave, qubits are arranged in the Chimera structure, which simply states that each qubit is entangled with a maximum of six other qubits. If the binary variables occur in quadratic terms, the entanglements are weighted accordingly. Since only entanglements can be weighted, but no single qubits and binary variables usually occur weighted with more than six other variables together, several qubits are defined as one variable. The individual weights are now distributed over the entanglements of the qubits of a variable. For the entanglements one qubit per variable is selected and the entanglement is weighted. Consequently, the number of qubits required depends on how many connections the binary variables have with each other. A and D do not contain quadratic terms and are therefore not important. In B each variable is connected to a maximum of $O(n \cdot R)$ others, in C to a maximum of $O(n \cdot k \cdot R)$. The number of required qubits thus increases quadratically dependent on the problem sizes:

$$O((n \cdot k \cdot R)^2)$$

The maximum number of variables is thus limited by the root of the number of qubits. Figure 14 shows the number of required qubits as a function of the number of variables, where the horizontal lines represent the numbers of qubits of available quantum annealers.

5.2 Experimental Analysis

For the experimental analysis, the time of solving is measured using simulated annealing and quantum annealing. We present the details about the test configurations (i.e., number of cores, maximum execution times, transactions and conflicts between them, their start times and lengths) in the Appendix.

For quantum annealing we use the cloud-service available at [11], which offers free access to a D-Wave 2000Q quantum computer for some time. The offered free time was enough to run the experimental evaluation described in Section 5.2.1.

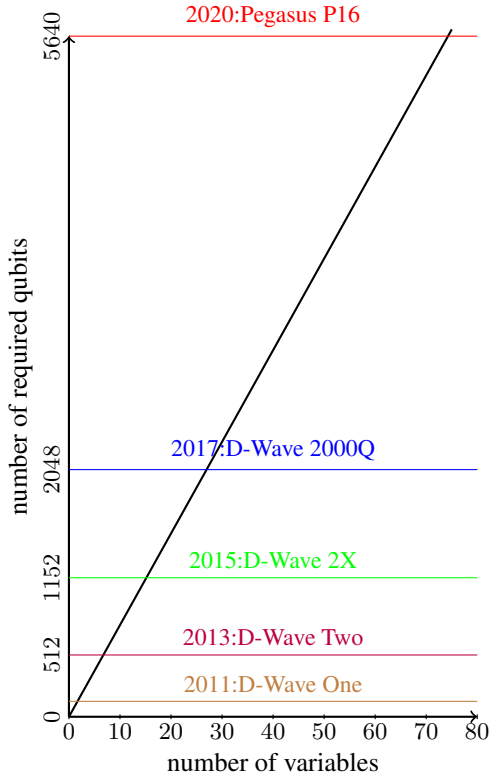


Figure 14: Required qubits depending on the number of variables and available qubits of D-Wave Quantum-Annealers

The experiments for preprocessing and simulated annealing are run on an i7-4510U dual core CPU with 2.0 GHz and 8 GB main memory running Windows 10. Every simulated annealing run and quantum annealing run was measured a hundred times and the lowest value is determined as an (almost) optimal solution. We present the overall total time of these hundred runs in the following sections.

5.2.1 Runtimes Quantum Annealing

Due to the limited number of qubits only small problem instances could be measured by quantum annealing (see Figure 15). The preprocessing time for generating the formula is the same for simulated as well as for quantum annealing, as both take the formula as input and find a minimal solution for it.

For every quantum annealing run the determined runtime is 20 microseconds as well as 274 microseconds for the readout, such that we verify the constant runtimes of quantum annealing in the experiments. Measuring a hundred times takes hence 29.4 milliseconds. While for the runtime of two transactions simulated annealing is still faster than quantum annealing, already problem

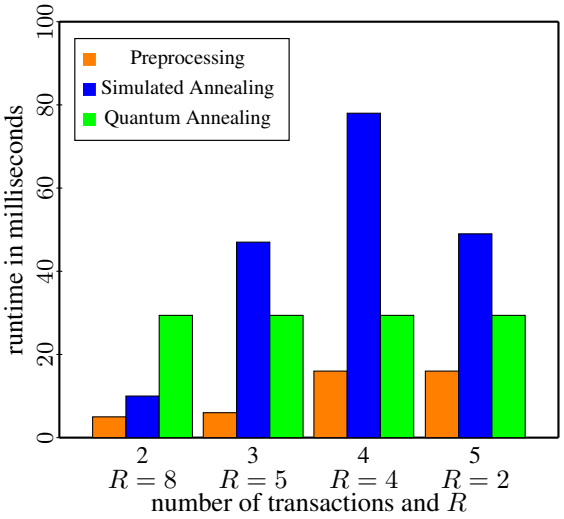


Figure 15: Runtimes of simulated annealing versus quantum annealing for $k = 2$

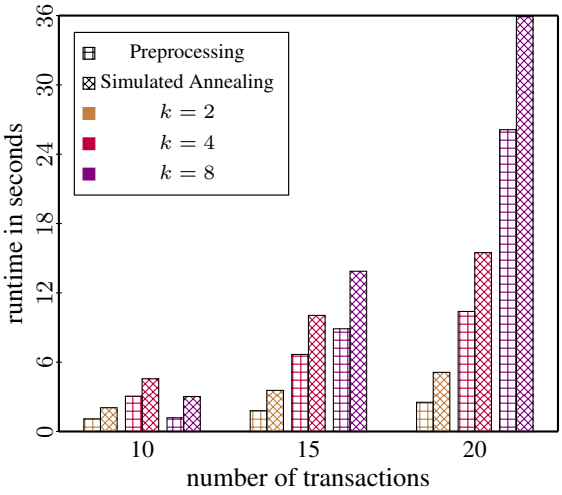


Figure 16: Runtimes for $R = 20$

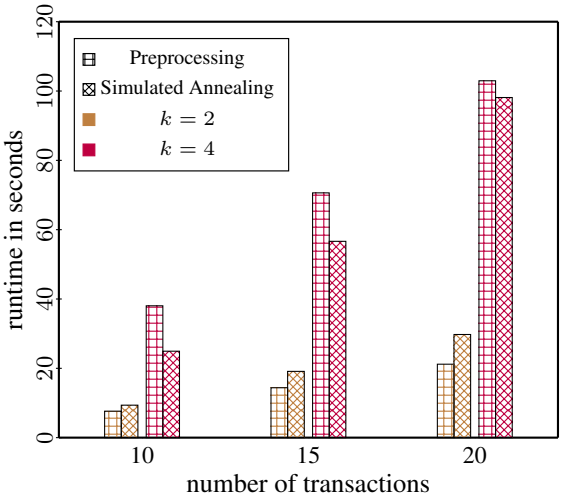


Figure 17: Runtimes for $R = 50$

sizes with 3 transactions and above are faster with quantum annealing with an observed speedup up to approx. 2.65. Once quantum annealers supporting more qubits are available, larger problem sizes can be solved with quantum annealing. As the times for quantum annealing and readout are constant, but simulated annealing runs become slower for larger problem sizes due to the more complex evaluation of larger formulas, larger speedups will be probably achieved when running on future quantum annealers.

5.2.2 Runtimes Simulated Annealing

For simulated annealing, we don't have limitations concerning the number of variables used in the formula to minimize. Hence we measure the runtimes of simulated annealing for larger problem sizes (see Figure 16 and 17). With a total of around 30 milliseconds the (constant) runtime of pure optimization by the quantum annealer is significantly better than that of simulated annealing even for relatively small problem instances. Assuming that also future quantum annealers will have approximately the same (constant) runtimes, speedups of quantum in comparison to simulated annealing up to 3300 for the problem sizes in Figure 16 and 17 are achievable.

5.2.3 Runtimes versus Number of Required Variables

The runtimes of the preprocessing phase as well as simulated annealing are dependent on the number of required variables in the generated formulas (see Figure 18). The number of required variables is hence a better and more simple measure for the problem size than considering all information of the configuration of transactions.

The runtimes of quantum annealing is obviously constant and hence does not depend on the number of required variables, but only small problem sizes can be solved due to the limited number of qubits supported by the quantum annealer.

6 Further Related Work

We introduce related work about the job shop scheduling problems in Section 6.1, optimizing transaction schedules in Section 6.2, simulated annealing in Section 6.3, quantum computing in Section 6.4,

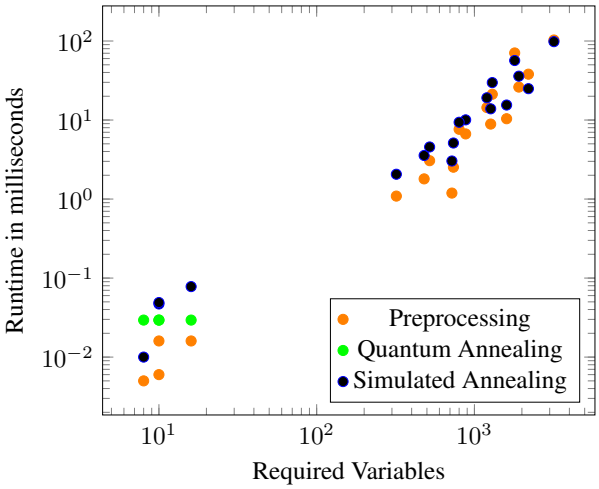


Figure 18: Execution times in relation to number of required variables

quantum databases in Section 6.5 and hardware-accelerated databases in Section 6.6.

6.1 Job Shop Scheduling Problems

The job shop scheduling problems (JSSP) are among the hardest combinatorial optimization problems [16]. Many heuristic approaches have been used to solve JSSP like simulated annealing [29], neural network [15], genetic algorithm [12], ant colony optimization [26], evolutionary algorithms [40], tabu search [35], and other approaches [7, 44, 28]. The contribution in [43] discusses solving job shop scheduling problems with quantum computers. The transaction schedule problem is similar to the job shop scheduling problem, but additionally considers that transactions can block each other when accessing the same data objects.

6.2 Optimizing Transaction Schedules

The authors in [13] have investigated reordering for optimistic concurrency control (OCC) approaches and introduce

- storage batching for reordering transaction reads and writes at the storage layer (in order to reduce conflicts on the same object), and
- validator batching for reordering transactions before validation (in order to reduce conflicts between transactions).

Transaction reordering for OCC is a non-trivial problem because of the dependencies between transactions. The authors in [13] hence introduce several approaches considering various transaction precedence policies like reducing tail latency. The presented approaches are parallel approaches in order to speed up reordering. The contributions in [13] investigate only backward validation for OCC, but we consider optimizing transaction schedules for 2 phase locking.

6.3 Database Tasks and Simulated Annealing

Simulated Annealing [36] has early been used for query optimizing [27], where a state space representing the equivalent algebraic forms of the query is defined together with a cost function for the states. Using simulated annealing the state with minimal costs are searched for.

[31] proposes a simulated annealing approach to select an optimal set of views to be materialized.

The authors of [25] formulate the data clustering problem as a graph partition problem by introducing a decomposition-based approach for reducing excessive disk accesses during simulated annealing. In [5] the horizontal fragmentation selection problem defining optimal disjoint sets of rows in relational warehouse tables to be separately stored and processed in parallel is solved by proposing a hybrid method combining genetic and simulated annealing algorithms.

Liu proposes in [32] a data mining algorithm for association rules mining by combining quantum-inspired genetic algorithm and simulated annealing.

[46] introduces spiders for internet search engines based on simulated annealing adjusting themselves according to the search progress or personalizations considering users' preferences or behavior.

6.4 Quantum Computing Including Quantum Annealing and Adiabatic Quantum Computation

In the last decade quantum computing including special forms like quantum annealing solving quadratic unconstrained binary optimization (QUBO) problems [10, 17] has become more and more popular and the range of applications wider and wider [22, 24, 37, 38], leading to a lot of time and money being spent on research in this area. Many scientific papers deal with the comparison of standard algorithms and quantum algorithms [4] and quite a few of them belong to the field of optimization problems [39, 42, 43]. For example, quantum annealing has been used to calculate urban motion flows for traffic control purposes [18] in a large city like Barcelona for the prediction of who wants to use which means of transport when and where[6].

[6] investigated by Volkswagen, see https://www.volkswagenag.com/en/news/stories/2018/11/intelligent-traffic-control-with-quantum-computers.html

6.5 Quantum Databases

There are only few contributions dealing with optimizing database problems with quantum computers like query optimization [42].

The contribution in [42] deals with the problem of multiple query optimization (MQO) by transforming a MQO problem instance into a mathematical formula that can be solved by the quantum annealer. In experiments the quantum annealer is already up to three orders of magnitude faster than other MQO algorithms executed on a traditional computer, although a limited number of qubits of a quantum annealer results in only small problem sizes of MQO that can be solved by a quantum annealer.

The authors in [37] introduce an execution model for resource transactions enabling deferred assignment of values to variables in committed transactions. Their approach maintains the database in a partially uncertain state, which they call *quantum state*, and updating it according to succeeding transactional operations. While the authors call the resulting database a quantum database, we prefer to call all databases that make use of a quantum computer as hardware accelerator *quantum databases*, which is more general than restricting the term quantum database to the authors' transaction model. Although the authors' transaction model seems to be quite suitable to be executed on a quantum computer, the authors run only experiments on traditional computers and did not prove the suitability of their model for quantum computers.

We introduce in [6] our approach to optimize transaction schedules in order to avoid blocking by a quantum annealer. In this paper we extend the contributions of [6] by detailing more about the basics, provide the proofs of validity and optimality, discuss approaches to reduce the preprocessing times by caching parts of the formulas to be generated and discuss further related work.

6.6 Hardware-Accelerated Databases

Hardware-accelerated servers [20] speed up database tasks by utilizing the massive parallelism of special hardware behind today's multi-core CPUs. Most contributions of hardware-accelerated servers in research use graphical processing units (GPUs) consisting of several thousand computing cores or field-programmable gate arrays (FPGAs) supporting to reconfigure interconnects for connecting programmable logic blocks with each other. The massive parallel processing of execution plans are ideal for many-core CPUs and GPUs as well as whenever the best possibilities among enumerated ones must be found (like in query optimization and multi-version concurrency control (MVCC)). Complex operations like joins processing large data inputs are very suitable for GPU-acceleration, too (see e.g. [47] for especially designed joins for SPARQL processing on GPUs). Other contributions include approximate searching [21].

FPGAs are ideal suitable for data-flow-driven algorithms (like processing an execution plan for evaluating queries in a streaming way without block-wise materialization of intermediate steps like it is the case for many-core CPUs and GPUs), but also any arbitrary type of parallelism can be offered by FPGAs. FPGA-acceleration of SPARQL query processing as discussed in e.g. [45] achieves scalable speedups even increasing with larger data sets. Dynamic partial reconfiguration enables FPGAs to dynamically exchange their configurations to process different queries at runtime [45].

While GPUs and FPGAs are suitable to find all solutions in a huge problem space, quantum computers are able to find only one of the best solutions, which limits the kinds of problems to be solved.

7 Summary and Conclusions

In this paper, we show how to transform the optimization problem of distributing transactions to cores of multi-core CPU machines, such that they are not blocking each other and are finished at the earliest possible time, into a QUBO-problem that can be solved by quantum annealers, which are quantum computers specialized to solve QUBO-problems. We prove our transformation scheme to be valid and optimal. We propose to use a cache of transformed problems in order to further decrease processing times. In the evaluation, the number of qubits required for the QUBO-problem of distributing transactions and the time needed for the transformation (with and without caching) are analyzed. Furthermore, we determine the size of a fully materialized cache for all possible problems (when fixing the number of cores, transactions and maximum processing time). We achieve runtime speedups of up to 2.6 of a quantum annealer compared to the one of simulated annealing on a classical computer in comprehensive experiments, although we tested so far only relatively small problem sizes due to limited number of qubits of current quantum annealers. However, these experiments promise much higher speedups for future generations of quantum annealers supporting many more qubits. Thus an increased interest in solving these kinds of optimization problems by quantum computers is very likely in the future.

In our future work we will investigate how to improve

e.g. the preprocessing time by using heuristics and allowing suboptimal solutions, which are close to the optimal one. Furthermore, we will investigate how to solve the problem of optimizing transaction schedules by applying quantum computers following the gate model and will compare it with our proposed approach utilizing quantum annealing. We may also consider to accelerate other transaction models and synchronization problems as well as other areas of database research by quantum computers.

REFERENCES

[1] Arcondello, "dimod," 2017. [Online]. Available: https://github.com/dwavesystems/dimod

[2] Arcondello, "dwave-neal," 2017. [Online]. Available: https://github.com/dwavesystems/dwave-neal

[3] A. Barenco, D. Deutsch, A. Ekert, and R. Jozsa, "Conditional quantum dynamics and logic gates," *Physical Review Letters*, vol. 74, no. 20, p. 4083, 1995.

[4] C. Barrón-Romero, "Classical and quantum algorithms for the boolean satisfiability problem," *arXiv preprint arXiv:1510.02682*, 2015.

[5] L. Bellatreche, K. Boukhalfa, and H. I. Abdalla, "Saga: A combination of genetic and simulated annealing algorithms for physical data warehouse design," in *British National Conference on Databases*, 2006.

[6] T. Bittner and S. Groppe, "Avoiding blocking by scheduling transactions using quantum annealing," in *24th International Database Engineering & Applications Symposium, Seoul, Republic of Korea*, 2020.

[7] H. Chen and P. B. Luh, "An alternative framework to lagrangian relaxation approach for job shop scheduling," *European Journal of Operational Research*, vol. 149, no. 3, pp. 499 – 512, 2003.

[8] T. M. Connolly and C. E. Begg, *Database systems: a practical approach to design, implementation, and management*. Pearson Education, 2005.

[9] D-Wave, "The d-wave 2000q^{TM} system," 2017. [Online]. Available: https://www.dwavesys.com/d-wave-two-system

[10] D-Wave, "D-wave's ocean software," 2017. [Online]. Available: https://ocean.dwavesys.com

[11] D-Wave, "Take the leap," 2020. [Online]. Available: https://www.dwavesys.com/take-leap

[12] R. Q. dao-er ji and Y. Wang, "A new hybrid genetic algorithm for job shop scheduling problem," *Computers & Operations Research*, vol. 39, no. 10, pp. 2291 – 2299, 2012.

[13] B. Ding, L. Kot, and J. Gehrke, "Improving optimistic concurrency control through transaction batching and operation reordering," *Proc. VLDB Endow.*, vol. 12, no. 2, p. 169–182, 2018.

[14] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser, "Quantum computation by adiabatic evolution," *arXiv preprint quant-ph/0001106*, 2000.

[15] D. J. Fonseca and D. Navaresse, "Artificial neural networks for job shop simulation," *Advanced Engineering Informatics*, vol. 16, no. 4, pp. 241 – 246, 2002.

[16] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Mathematics of Operations Research*, vol. 1, no. 2, pp. 117–129, 1976.

[17] F. Glover, G. Kochenberger, and Y. Du, "Quantum bridge analytics i: a tutorial on formulating and using qubo models," *4OR*, vol. 17, no. 4, pp. 335–371, 2019.

[18] P. Goddard, S. Mniszewski, F. Neukart, S. Pakin, and S. Reinhardt, "How will early quantum computing benefit computational methods?" in *Proc. SIAM Annu. Meeting*, 2017.

[19] S. Groppe, "Emergent models, frameworks, and hardware technologies for big data analytics," *The Journal of Supercomputing*, vol. 76, no. 3, pp. 1800–1827, 2020.

[20] S. Groppe and J. Groppe, "Hybrid multi-model multi-platform (hm3p) databases," in *Proceedings of the 9th International Conference on Data Science, Technology and Applications (DATA)*, 2020.

[21] T. Groth, S. Groppe, M. Koppehel, and T. Pionteck, "Parallelizing approximate search on adaptive radix trees," in *Proceedings of the 28th Italian Symposium on Advanced Database Systems, Villasimius, Sud Sardegna, Italy (virtual due to Covid-19 pandemic)*, 2020. [Online]. Available: http://ceur-ws.org/Vol-2646/16-paper.pdf

[22] L. K. Grover, "A fast quantum mechanical algorithm for database search," *arXiv preprint quant-ph/9605043*, 1996.

[23] L. K. Grover, "Quantum computers can search arbitrarily large databases by a single query," *Physical review letters*, vol. 79, no. 23, p. 4709, 1997.

[24] T. Hogg, "Adiabatic quantum computing for random satisfiability problems," *Physical Review A*, vol. 67, no. 2, p. 022314, 2003.

[25] K. A. Hua, S. D. Lang, and W. K. Lee, "A decomposition-based simulated annealing technique for data clustering," in *Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, ser. PODS '94. New York, NY, USA: Association for Computing Machinery, 1994.

[26] K.-L. Huang and C.-J. Liao, "Ant colony optimization combined with taboo search for the job shop scheduling problem," *Computers & Operations Research*, vol. 35, no. 4, pp. 1030 – 1046, 2008.

[27] Y. E. Ioannidis and E. Wong, "Query optimization by simulated annealing," in *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '87, New York, NY, USA, 1987.

[28] K. Jansen, M. Mastrolilli, and R. Solis-Oba, "Approximation schemes for job shop scheduling problems with controllable processing times," *European Journal of Operational Research*, vol. 167, no. 2, pp. 297 – 319, 2005.

[29] M. Kolonko, "Some new results on simulated annealing applied to the job shop scheduling problem," *European Journal of Operational Research*, vol. 113, no. 1, pp. 123 – 136, 1999.

[30] Kotarotanahashi, "Pyqubo," 2017. [Online]. Available: https://github.com/recruit-communications/pyqubo

[31] T. V. Kumar and S. Kumar, "Materialized view selection using simulated annealing," in *International Conference on Big Data Analytics*. Springer, 2012, pp. 168–179.

[32] D. Liu, "Improved genetic algorithm based on simulated annealing and quantum computing strategy for mining association rules," *Journal of Software*, vol. 5, no. 11, pp. 1243–1249, 2010.

[33] A. Marchenkova. (2016) What's the difference between quantum annealing and universal gate quantum computers? [Online]. Available: https://medium.com/quantum-bits

[34] M. Morsey, J. Lehmann, S. Auer, and A. N. Ngomo, "Dbpedia SPARQL benchmark - performance assessment with real queries on real data," in *The Semantic Web - 10th International Semantic Web Conference (ISWC 2011), Bonn, Germany*, 2011, pp. 454–469.

[35] F. Pezzella and E. Merelli, "A tabu search method guided by shifting bottleneck for the job shop scheduling problem," *European Journal of Operational Research*, vol. 120, no. 2, pp. 297 – 310, 2000.

[36] M. Pincus, "Letter to the editor-a monte carlo method for the approximate solution of certain types of constrained optimization problems," *Operations Research*, vol. 18, no. 6, pp. 1225–1228, 1970.

[37] S. Roy, L. Kot, and C. Koch, "Quantum databases," in *Proc. CIDR*, no. CONF, 2013.

[38] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 1994, pp. 124–134.

[39] T. Stollenwerk and A. Basermann, "Experiences with scheduling problems on adiabatic quantum computers," in *Proceedings of the 1st International Workshop on Post-Moore Era Supercomputing (PMES), Future Technologies Group Technical report FTGTR-2016-11*, 2016, pp. 45–46.

[40] I. T. Tanev, T. Uozumi, and Y. Morotome, "Hybrid evolutionary algorithm-based real-world flexible job shop scheduling problem: application service provider approach," *Applied Soft Computing*, vol. 5, no. 1, pp. 87 – 100, 2004.

[41] A. Thomson, T. Diamond, S.-C. Weng, K. Ren, P. Shao, and D. J. Abadi, "Calvin: Fast distributed transactions for partitioned database systems," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, ser. SIGMOD '12, 2012, p. 1–12.

[42] I. Trummer and C. Koch, "Multiple query optimization on the d-wave 2x adiabatic quantum computer," *Proc. VLDB Endow.*, vol. 9, no. 9, pp. 648–659, May 2016.

[43] D. Venturelli, D. J. J. Marchand, and G. Rojo, "Quantum annealing implementation of job-shop scheduling," *arXiv*, 2015. [Online]. Available: https://arxiv.org/abs/1506.08479

[44] H. Wenqi and Y. Aihua, "An improved shifting bottleneck procedure for the job shop scheduling problem," *Computers & Operations Research*, vol. 31, no. 12, pp. 2093 – 2110, 2004.

[45] S. Werner, D. Heinrich, S. Groppe, C. Blochwitz, and T. Pionteck, "Runtime adaptive hybrid query engine based on fpgas," *Open Journal of Databases (OJDB)*, vol. 3, no. 1, pp. 21–41, 2016. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:101:1-201705194645

[46] C. C. Yang, J. Yen, and H. Chen, "Intelligent internet searching agent based on hybrid simulated annealing," *Decision Support Systems*, vol. 28, no. 3, pp. 269–277, 2000.

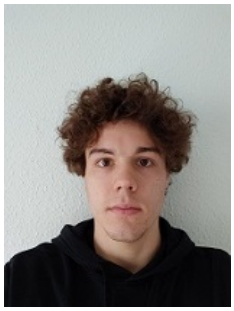
[47] X. Zhang, M. Zhang, P. Peng, J. Song, Z. Feng, and L. Zou, "A scalable sparse matrix-based join for sparql query processing," in *International Conference on Database Systems for Advanced Applications*. Springer, 2019, pp. 510–514.

APPENDIX

We present the test configurations for Figure 15,16 and 17 in the following tables. For each test configuration, we assume $T = \{t_1, ..., t_n\}$ and $M = \{m_1, ..., m_k\}$. The last column of the following table contains the number of required variables.

Fig.	k	n	R	O	$l_1, ..., l_n$	$r_1, ..., r_n$	**req. var.**
15	2	2	8	$\{\}$	8, 4	0, 4	8
		3	5	$\{(t_1, t_3)\}$	4, 5, 1	1, 0, 4	10
		4	4	$\{(t_2, t_4)\}$	3, 2, 1, 2	1, 2, 3, 2	16
		5	2	$\{(t_1, t_2), (t_4, t_5)\}$	1, 1, 1, 1, 1	1, 1, 1, 1, 1	10
16	2	10	20	$\{(t_1, t_3), (t_3, t_4), (t_7, t_8)\}$	5, 7, 1, 1, 3, 4, 4, 2, 8, 5	15, 13, 19, 19, 15, 16, 16, 18, 12, 15	320
	4			$\{(t_1, t_2), (t_9, t_{10})\}$	10, 14, 8, 5, 4, 7, 6, 10, 3, 13	10, 6, 12, 15, 16, 13, 14, 10, 17, 7	480
	8			$\{(t_1, t_2), (t_3, t_4), (t_5, t_6)\}$	16, 20, 5, 15, 4, 6, 14, 10, 6, 12	4, 0, 15, 5, 16, 14, 6, 10, 14, 8	736
	2	15		$\{(t_2, t_3), (t_4, t_6), (t_5, t_{10}), (t_1, t_{15})\}$	4, 2, 3, 1, 4, 6, 2, 1, 5, 1, 1, 3, 2, 4, 1	16, 18, 17, 19, 16, 14, 18, 19, 15, 19, 19, 17, 18, 16, 19	520
	4			$\{(t_1, t_2), (t_3, t_4), (t_7, t_8)\}$	6, 4, 3, 1, 2, 8, 7, 10, 6, 11, 1, 3, 6, 4, 8	14, 16, 17, 19, 18, 12, 13, 10, 14, 9, 19, 17, 14, 16, 12	880
	8			$\{(t_2, t_4), (t_3, t_6), (t_3, t_{14})\}$	9, 12, 5, 3, 8, 14, 3, 7, 8, 1, 2, 4, 20, 1, 3	11, 8, 15, 17, 12, 6, 17, 13, 12, 19, 18, 16, 0, 19, 17	1600
	2	20		$\{(t_5, t_{12}), (t_6, t_{18}), (t_7, t_{20}), (t_8, t_{20})\}$	1, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 2, 2, 1, 1, 1, 1, 6, 5, 4	19, 19, 19, 18, 16, 19, 19, 18, 19, 19, 18, 18, 18, 19, 19, 19, 19, 14, 15, 16	720
	4			$\{(t_1, t_2), (t_2, t_{17}), (t_5, t_9)\}$	3, 4, 1, 2, 4, 1, 1, 7, 1, 10, 1, 9, 5, 3, 8, 5, 2, 7, 1, 8	17, 16, 19, 18, 16, 19, 19, 13, 19, 10, 19, 11, 15, 17, 12, 15, 18, 13, 19, 12	1268
	8			$\{(t_2, t_4), (t_8, t_{19}), (t_{11}, t_{13})\}$	7, 16, 8, 4, 11, 4, 6, 5, 7, 12, 19, 13, 1, 9, 1, 2, 3, 4, 15, 14	13, 4, 12, 16, 9, 16, 14, 15, 13, 8, 1, 7, 19, 11, 19, 18, 17, 16, 5, 6	1912
17	2	10	50	$\{(t_2, t_{10}), (t_3, t_4)\}$	15, 17, 10, 10, 4, 9, 8, 12, 2, 13	35, 33, 40, 40, 46, 41, 42, 38, 48, 37	800
	4			$\{(t_1, t_2), (t_9, t_{10})\}$	20, 21, 12, 25, 24, 16, 27, 15, 18, 22	30, 29, 38, 25, 26, 34, 23, 35, 32, 28	1200
	2	15		$\{(t_1, t_2), (t_3, t_{11}), (t_5, t_{10})\}$	6, 7, 3, 1, 12, 8, 7, 10, 4, 11, 1, 13, 6, 4, 8	44, 43, 47, 49, 38, 42, 43, 40, 46, 39, 49, 37, 44, 46, 42	1298
	4			$\{(t_1, t_3), (t_3, t_4), (t_7, t_9), (t_{11}, t_{12})\}$	15, 12, 9, 1, 12, 22, 23, 14, 9, 8, 14, 17, 24, 13, 7	35, 38, 41, 49, 38, 28, 27, 36, 41, 42, 36, 33, 26, 37, 43	2200
	2	20		$\{(t_4, t_5), (t_5, t_6), (t_{12}, t_{13}), (t_{17}, t_{18})\}$	3, 4, 1, 2, 4, 1, 10, 7, 1, 11, 1, 9, 8, 3, 8, 5, 2, 7, 1, 12	47, 46, 49, 48, 46, 49, 40, 43, 49, 39, 49, 41, 42, 48, 42, 45, 48, 43, 49, 38	1800
	4			$\{(t_3, t_{17}), (t_5, t_7), (t_6, t_8)\}$	12, 17, 19, 4, 11, 22, 3, 14, 23, 2, 9, 12, 2, 12, 12, 4, 5, 2, 3, 14	38, 33, 31, 46, 39, 28, 47, 36, 27, 48, 41, 38, 48, 38, 38, 46, 45, 48, 47, 36	3192

AUTHOR BIOGRAPHIES



Tim Bittner studied computer science at the University of Lübeck. He earned his bachelor's degree in 2020. His bachelor thesis deals with the optimization of transaction synchronization by quantum computing.

Sven Groppe earned his diploma degree in Computer Science in 2002 and his Doctor degree in 2005 from the University of Paderborn. He earned his habilitation degree in 2011 and his Professor title in 2019 from the University of Lübeck. He worked in the European projects B2B-ECOM, MEMPHIS, ASG and TripCom. He was a member of the DAWG W3C Working Group, which developed SPARQL. He was the project leader of the DFG project LUPOSDATE, an open-source Semantic Web database, and one of the project leaders of two research projects, which research on FPGA acceleration of relational and Semantic Web databases. He is leading a DFG project on GPU and APU acceleration of main-memory database indexes, and a DFG project about Semantic Internet of Things. He is also the chair of the Semantic Big Data workshop series, which is affiliated with the ACM SIGMOD conference (so far 2016 to 2020), and of the Very Large Internet of Things workshop in conjunction with the VLDB conference (so far 2017 to 2020). His research interests include databases, Semantic Web, query and rule processing and optimization, Cloud Computing, acceleration via GPUs and FPGAs, peer-to-peer (P2P) networks, Internet of Things, data visualization and visual query languages. Updated information is available at `https://www.ifis.uni-luebeck.de/~groppe/`.