Bounding the Makespan of Transaction Schedules

Tim Baccaert
Software Languages Lab, Vrije Universiteit Brussel, Belgium

Brecht Vandevoort
UHasselt, Data Science Institute, ACSL, Diepenbeek, Belgium

Bas Ketsman
Software Languages Lab, Vrije Universiteit Brussel, Belgium

Abstract

The performance of transactional database systems is typically evaluated by measuring the amount of transactions they can commit to the database per second. However, fairly measuring this for the same workload on different systems is not trivial. It is therefore relevant to formalize schedule efficiency, investigate the space of all possible efficient schedules, and identify whether there is any room for improvement. Prior transaction theory largely centers on decision problems relating to safety, such as the serializability, robustness, and allocation problems. Most pertinently, these problems take already scheduled transactions as input, and do not directly consider the efficiency of those schedules.

In this work, we define schedules as assignments of operations on objects to discrete points in time. This allows us to quantify efficiency as the elapsed duration between the schedule's beginning and end, more commonly known as the makespan in the scheduling literature. We establish that, given some set of transactions and a desired makespan, it is NP-complete to decide if there exists a conflict serializable schedule which is bounded by that makespan. We additionally provide an instance optimal algorithm for scheduling transaction sets with a single contention point, that is, exactly one object may appear in conflicting operations. Lastly, we give worst-case optimal bounds on the makespan, meaning that schedules can never exceed this bound, and for the worst transaction sets, the bound is optimal.

2012 ACM Subject Classification Theory of computation → Discrete optimization; Information systems → Database transaction processing

Keywords and phrases Transactions, Scheduling, Discrete Optimization, Complexity

Digital Object Identifier 10.4230/LIPIcs.ICDT.2026.10

Funding This work is partly funded by FWO-grant G019921N.

1 Introduction

Transaction processing is a core aspect of every database system. In most systems it is implemented through *concurrency control*, a set of principled protocols which orchestrate access to resources shared by multiple threads of execution. In the database systems literature, this aspect of the system is evaluated by measuring its *throughput*, or the amount of transactions it can commit to the database per second, on various predetermined workloads such as YCSB [7] or TPC-C [8]. Reliably executing the same workload on different systems is not always evident, as is demonstrated by numerous papers attempting to make this evaluation method less biased [1, 10, 26, 33, 34]. As database theorists, it is therefore pertinent for us to consider a formal theory of schedule efficiency, to investigate the space of all possible efficient schedules, and to identify whether there is room for improvement.

The current theory about transactions largely centers around *safety properties* [18, 23], and we elaborate on this work's relation to this literature in Section 7. In its most elementary form, safety is about deciding if a given schedule is valid according to a definition of validity,

the gold standard being conflict serializability. In other words, the main purpose is to delineate the space of schedules that is valid. In this work, we are concerned with identifying the space of optimal schedules within that space of valid ones.

Concretely, we formalize schedules as assignments of all operations in a set of transactions to positive integers, intuitively representing their point of execution in time. This enables us to explicitly reason about operations that run in parallel, rather than observing the total order of operations after execution has taken place. Moreover, it lets us quantify efficiency as the elapsed duration between a schedule's beginning and end, a quantity better known in the scheduling literature as the *makespan* [4, 25].

Contributions

We focus on two central questions, the first being: "*What is the complexity of finding a conflict serializable schedule of minimal makespan?*". To this extent, we contribute the following results.

1. In Section 3, we show that the schedules with minimal makespan are among those constructed from a total order on the transactions. Hence, we can find a minimal makespan schedule in factorial time by enumerating all total orders on the given transactions.
2. Next, in Section 4, we show that for a set of transactions and a positive integer k, it is NP-complete to decide if there exists a schedule with a makespan less than or equal to k.
3. In Section 5, we give a polynomial time instance optimal scheduling algorithm for transaction sets with at most one object on which operations are allowed to be conflicting.

Second, we handle a complementary question: "*Is there a bound on the optimal makespan?*". To this end, we provide the following contribution.

4. In Section 6, we derive worst-case optimal bounds on the makespan. That is, the makespan can never exceed this bound, and for the worst transaction sets, the bound is optimal. The upper bounds (excluding Proposition 27) imply algorithms that can generate schedules with a makespan matching this bound.

To enable this analysis, we group transaction sets into a class $\mathbb{T}_{\ell,m,n}$ based on the maximal number ℓ of operations in each transaction, the maximal number m of *contention points* (*i.e.*, objects on which operations are allowed to be conflicting), and the maximal number n of transactions in the set. The derived bounds are given in Table 1.

Table 1 Summary of all worst-case optimal bounds for $\mathbb{T}_{\ell,m,n}$ in this work, with $\ell \geq m$. Bounds marked $\downarrow$ (red) or $\uparrow$ (green) are, upper (resp. lower) bounds without matching lower (resp. upper) bounds. For the bounds marked $\not\mathbb{R}$ we have not given a tractable algorithm.

<table>
<tr><th></th><th>$m = 0$</th><th>$m = 1$</th><th>$m = 2$</th><th>$m = 3$</th><th>$4 \leq m$</th></tr>
<tr><td>$n = 1$</td><td rowspan="2">ℓ</td><td>ℓ</td><td>ℓ</td><td>ℓ</td><td>ℓ</td></tr>
<tr><td>$n = 2$</td><td>$\ell + 1$</td><td>2ℓ</td><td>2ℓ</td><td>2ℓ</td></tr>
<tr><td>$3 \leq n < m!$</td><td>$3 \not\leq m!$</td><td>$3 \not\leq m!$</td><td>$3 \not\leq m!$</td><td rowspan="2">$n\ell - n + 1$</td><td>$n\ell - n + 1 \downarrow \not\mathbb{R}$</td></tr>
<tr><td>$n = m!$</td><td rowspan="4">ℓ</td><td>ℓ</td><td>2ℓ</td><td>$n\ell - (n-1)(m-2)$ $\not\mathbb{R}$</td></tr>
<tr><td>$m! < n < m!m$</td><td rowspan="3">$\ell + n - 1$</td><td rowspan="3">$2\ell + n - 2$</td><td>$7\ell + n - 14 \downarrow$
$6\ell + n - 11 \uparrow$</td><td rowspan="3">$(m! + m - 2)\ell + n - (m-1)(m! + m - 2) \downarrow \not\mathbb{R}$</td></tr>
<tr><td>$m!m \leq n < (m+1)!$</td><td>$7\ell + n - 14 \downarrow$
$6\ell + \lfloor \frac{\ell-3}{2} \rfloor + n - 11 \uparrow$</td></tr>
<tr><td>$(m+1)! \leq n$</td><td>$7\ell + n - 14 \downarrow$
$7\ell + n - 15 \uparrow$</td></tr>
</table>

2 Definitions

For integers i and j, we write $[i, j]$ as shorthand notation for $\{i, i+1, \ldots, j\}$, and write $[i]$ as a shorthand for $[1, i]$. By $\mathbb{N}_0$ we denote the positive integers, including 0.

We fix an infinite set of objects $\mathcal{O} = \{a, b, c, \ldots\}$. A *transaction* T over $\mathcal{O}$ is a sequence of operations $T = o_1\ o_2\ \ldots\ o_\ell$. We write $|T|$ to mean the number of operations ℓ in T. To every *operation* $o \in T$ we associate a $\mathsf{type}(o) \in \{\mathtt{R}, \mathtt{W}\}$ and an $\mathsf{obj}(o) \in \mathcal{O}$. We say that o is a *read* if $\mathsf{type}(o) = \mathtt{R}$ and that it is a *write* if $\mathsf{type}(o) = \mathtt{W}$. Furthermore, we say that o is an operation on object $\mathsf{obj}(o)$.

For an integer $i \in [\ell]$, we write $T[i]$ to denote the ith operation of T. That is, $T[i] = o_i$. Slightly abusing notation, we will often treat transactions as sets of objects, allowing to write $o \in T$ to mean that o is one of its operations. For two operations $T[i], T[j]$ of T, we write $T[i] \leq_T T[j]$ to mean $i \leq j$, and $T[i] <_T T[j]$ to mean $i < j$. We will leave $\mathcal{O}$ implicit when it can be derived from the context and say transaction T instead of transaction T over $\mathcal{O}$.

When considering a set of transactions $\mathcal{T}$, we will assume that every transaction has a unique id i, and we write T_i to make this id explicit. For notational convenience we assume that each transaction can read to each object at most once, and write to each object at most once. *This is an assumption commonly made in other literature [18, 19, 30, 31], and the problems we consider become computationally more difficult if this is not the case.* It also allows us to refer to a specific operation o in $T_i \in \mathcal{T}$, using the notation $\mathtt{R}_i[x]$ (resp., $\mathtt{W}_i[x]$), with $x = \mathsf{obj}(o)$ and $\mathsf{type}(o) = \mathtt{R}$ (resp., $\mathsf{type}(o) = \mathtt{W}$).

As usual, we say that two operations o and p occurring in transactions in $\mathcal{T}$ are *conflicting* if they are from different transactions in $\mathcal{T}$, and with $\mathsf{obj}(o) = \mathsf{obj}(p)$ and $\mathtt{W} \in \{\mathsf{type}(o), \mathsf{type}(p)\}$.

We define schedules as functions assigning integers to the operations of transactions, intuitively representing the timing of these operations.

▶ **Definition 1.** *A* schedule s *for a set* $\mathcal{T}$ *of transactions is a function* $s : O_s \to \mathbb{N}_0$ *that assigns a* $s(o) \in \mathbb{N}_0$ *to all the operations* $o \in O_s$ *of transactions in* $\mathcal{T}$*.*

For notational convenience, we will sometimes represent a schedule for $\mathcal{T}$ as a sequence of sets of operations $A_1\ A_2\ \ldots\ A_k$ (with every operation of a transaction in $\mathcal{T}$ occurring in precisely one of the sets A_i), which then should be interpreted as the schedule for $\mathcal{T}$ with $s(o) = i - 1$ if $o \in A_i$. In other words, the set A_i represents all operations that are executed at precisely time i.

Henceforth, we will only consider schedules having the next two additional properties:

1. for operations $\{o, p\} \subseteq T$ with $o <_T p$, we have $s(o) < s(p)$;
2. for conflicting operations $\{o, p\} \subseteq O_s$ we have $s(o) < s(p)$ or $s(p) < s(o)$.

Property (1) expresses that s must preserve the order in which operations occur in the transaction they belong to. Property (2) guarantees that conflicting operations are not executed at precisely the same time.

We call a schedule s *dense* if for every operation $o \in O_s$, either $s(o) = 0$ *or* there is an operation $p \in O_s$ with $s(o) = s(p) + 1$ and *either* p conflicts with o *or* it is from the same transaction as o. In other words, a schedule is dense if none of its operations can be moved one time unit to the left without invalidating Property (1) or Property (2).

▶ **Example 2.** We define a set of transactions $\mathcal{T} = \{T_1, T_2, T_3, T_4\}$, where

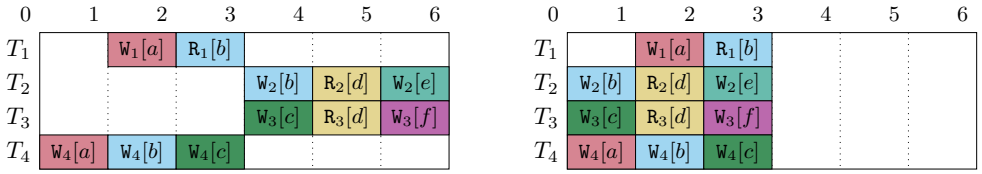
$$T_1 = \mathtt{W}_1[a]\ \mathtt{R}_1[b];\quad T_2 = \mathtt{W}_2[b]\ \mathtt{R}_2[d]\ \mathtt{W}_2[e];\quad T_3 = \mathtt{W}_3[c]\ \mathtt{R}_3[d]\ \mathtt{W}_3[f];\ \text{and}\ T_4 = \mathtt{W}_4[a]\ \mathtt{W}_4[b]\ \mathtt{W}_4[c].$$

We remark that usage of blind writes, or a write to an object without first reading to it within the same transaction, is not common in real-world transaction workloads. However, we use them to keep examples concise. One can always replace each write operation with a read followed by a write.

A possible schedule s_1 for $\mathcal{T}$ is:

$$s_1(\mathtt{W}_4[a]) = 0 \quad s_1(\mathtt{W}_4[b]) = 1 \quad s_1(\mathtt{W}_4[c]) = 2 \quad s_1(\mathtt{W}_1[a]) = 1 \quad s_1(\mathtt{R}_1[b]) = 2 \quad s_1(\mathtt{W}_2[b]) = 3$$
$$s_1(\mathtt{R}_2[d]) = 4 \quad s_1(\mathtt{W}_2[e]) = 5 \quad s_1(\mathtt{W}_3[c]) = 3 \quad s_1(\mathtt{R}_3[d]) = 4 \quad s_1(\mathtt{W}_3[f]) = 5$$

We can represent schedules using a Gantt-chart, as seen in Figures 1a and 1b.



(a) A schedule s_1 for $\mathcal{T}$. **(b)** A schedule s_2 for $\mathcal{T}$.

Figure 1 Two different schedules for the same transaction set $\mathcal{T}$. Time advances from left to right and is denoted at the top, and each row is a transaction. The operations o_j within each transaction $T_i \in \mathcal{T}$ are denoted by rectangles colored according to their object.

We remark that this definition of schedules differs from most literature[1]*, e.g. [11, 18, 19, 30, 31], which typically defines schedules as total orders on operations. We deviate from this because we need to reason about the specific length of a schedule s.* Two operations o and p in O_s that execute *in parallel*, that is $s(o) = s(p)$ in our model, are usually reduced to a freely chosen order on $\{o, p\}$ in the total order definition. Since the length of a schedule matters in our setting, we make this parallelism more explicit.

We are interested in precisely those schedules that maximally execute operations in parallel, while simultaneously exhibiting the same outcome as a schedule that executes each transaction one by one. We first define the latter condition, and then discuss what it means to maximally parallelize operations.

If a schedule s for $\mathcal{T}$ executes transactions one by one we say it is *serial*. Formally, a schedule s is serial if it has no concurrent operations (*i.e.*, for every pair of different operations o, p in O_s, we either have $s(o) < s(p)$ or $s(p) < s(o)$) and it is non-interleaving (*i.e.*, for every pair of different transactions T_i, T_j in $\mathcal{T}$, and every pair of different operations o, p in T_i, with $s(o) < s(p)$ there is no $q \in T_j$ with $s(o) < s(q) < s(p)$).

If o conflicts with p and $s(o) < s(p)$, then p *depends on* o *in* s, which is written as $o \rightarrow_s p$. We now define *conflict serializable* schedules, which formalizes one notion of what it means for a schedule to have the same outcome as a serial schedule.

▶ **Definition 3** (Papadimitriou [23]). *Let $\mathcal{T}$ be a set of transactions and let s_1 and s_2 be schedules for $\mathcal{T}$. Then, s_1 is* conflict equivalent *to s_2 if $o \rightarrow_{s_1} p$ implies $o \rightarrow_{s_2} p$ and vice versa, for all conflicting operations o and p in $\mathcal{T}$. Furthermore, s_1 is* conflict serializable *if there exists a serial schedule s_3 for $\mathcal{T}$ such that s_1 is conflict equivalent to s_3.*

[1] [32] has a definition based on partial orders, but does not use them to reason about schedule efficiency.

We can test whether a schedule s is serializable by creating a *conflict graph* for s, which is a directed graph $\mathrm{CG}(s) = (\mathcal{T}, E)$, where $(T_i, T_j) \in E$ if there exists $o \to_s p$ with $o \in T_i$ and $p \in T_j$ for distinct $\{T_i, T_j\} \subseteq \mathcal{T}$. We can then apply the next textbook result to obtain a decision procedure.

▶ **Theorem 4** (Papadimitriou [23]). *A schedule s is conflict serializable if, and only if,* $\mathrm{CG}(s)$ *is a directed acyclic graph.*

To be able to compare the efficiency of schedules, we associate every schedule with a cost that equals its makespan. The *makespan* of a schedule s is defined as

$$\mathsf{span}(s) = \max\{s(o) + 1 \mid o \in T, T \in \mathcal{T}\}.$$

Intuitively, $\mathsf{span}(s)$ can be thought of as the completion time of the final operation in the given schedule.

▶ **Example 5.** Both the schedule s_1 for $\mathcal{T}$ from Example 2 and the schedule s_2 for $\mathcal{T}$ defined in Figure 1b are serializable. Namely, s_1 is equivalent to a serial schedule which first executes T_4, then T_1 and T_2, and finally T_3. For s_2 a possible equivalent serial schedule, is T_2 followed by T_3, then T_4 and finally T_1. The makespan of s_1 is 6, while the makespan of s_2 is 3. Hence, the more efficient schedule is s_2.

3 Precedence Graphs and Canonical Schedules

We define the *precedence graph* P_s of a schedule s for some set $\mathcal{T}$ of transactions as the directed graph $\mathrm{P}_s = (V, E)$ with $V = O_s$ and

$$E = \{(T[j], T[j+1]) \mid T \in \mathcal{T}, j \in [|T| - 1]\} \cup \{(o, p) \in O_s \times O_s \mid o \to_s p\}.$$

Further on in this paper, we will refer to the precedence graph P_Σ. This is notation for the precedence graph of the unique dense serializable schedule for $\mathcal{T}$ originating from a total order $\Sigma = (\mathcal{T}, \leq_\mathcal{T})$ on $\mathcal{T}$. More formally one replaces $\{(o, p) \in O_s \times O_s \mid o \to_s p\}$ in the above definition by all (o, p) such that $o \in T$, $p \in T'$ with $\{T, T'\} \subseteq \mathcal{T}$, $T \neq T'$ and $T \leq_\mathcal{T} T'$.

The following property follows directly from Theorem 4:

▶ **Corollary 6.** *For a serializable schedule s for some set $\mathcal{T}$ of transactions, the precedence graph P_s is acyclic and there is an order $\Sigma = (\mathcal{T}, \leq_\mathcal{T})$ such that $P_s = P_\Sigma$.*

Note that for schedules that are not serializable, the precedence graph must be cyclic.

Let us now denote for an acyclic directed graph G by $\mathsf{max\text{-}path}(G)$ the maximum number of nodes occurring in any directed path of G. Then, $\mathsf{max\text{-}path}(P_\Sigma)$ for a totally ordered set of transactions $\Sigma = (\mathcal{T}, \leq_\mathcal{T})$ corresponds to the makespan of a particular schedule for $\mathcal{T}$.

▶ **Definition 7.** *The canonical schedule for $\Sigma = (\mathcal{T}, \leq_\mathcal{T})$ is defined as the schedule $\mathsf{canonical}(\Sigma) = A_1\, A_2 \ldots A_k$ over $\mathcal{T}$ with $k = \mathsf{max\text{-}path}(P_\Sigma)$, and*

$$A_i = \{p \mid (o, p) \in E,\ o \in A_{j<i},\ p \notin A_{j<i}\},$$

where $A_{j<i} = \bigcup_{j<i} A_j$, and E is the set of directed edges in P_Σ.

It is straightforward to verify that canonical schedules are dense by construction.

Next we will show that a minimal cost schedule can be found among the canonical schedules. For this, let $\mathsf{cost}(\mathcal{T}) = \min\{\mathsf{span}(s) \mid s \text{ is a serializable schedule for } \mathcal{T}\}$. The following corollary is immediate.

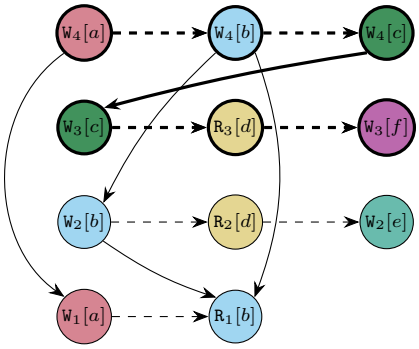


Figure 2 A visual depiction of a precedence graph defined in Example 9. Edges indicating the relative order of operations of the same transaction are given by dashed arrows and those indicating the order of conflicting operations are given by solid lines. A longest path is highlighted by a bold outline.

▶ **Proposition 8.** *Let $\mathcal{T}$ be a set of transactions.*

- *For every serializable schedule s for $\mathcal{T}$, we have that* max-path$(P_s) \leq$ span(s). *If s is dense, then* span$(s) =$ max-path(P_s).
- *There exists a total order $\Sigma = (\mathcal{T}, \leq_{\mathcal{T}})$ such that* max-path$(P_\Sigma) =$ cost$(\mathcal{T})$.
- *There exists a total order $\Sigma = (\mathcal{T}, \leq_{\mathcal{T}})$ such that* span(canonical$(\Sigma)) =$ cost$(\mathcal{T})$.

Notice that this implies $\mathsf{cost}(\mathcal{T}) = \min\{\mathsf{span}(\mathsf{canonical}(\Sigma)) \mid \Sigma \textit{ is a total order for } \mathcal{T}\}$.

▶ **Example 9.** Let $\mathcal{T}$ be the set of transactions from Example 2. Then, P_Σ with $\Sigma = (\mathcal{T}, \{(T_i, T_j) \mid i \geq j\})$ is the precedence graph depicted in Figure 2. The longest path consists of edges $(\mathtt{W}_4[a], \mathtt{W}_4[b])$ $(\mathtt{W}_4[b], \mathtt{W}_4[c])$ $(\mathtt{W}_4[c], \mathtt{W}_3[c])$ $(\mathtt{W}_3[c], \mathtt{R}_3[d])$ $(\mathtt{R}_3[d], \mathtt{W}_3[f])$ and hence max-path$(P_\Sigma) = 6$. The canonical schedule for Σ equals schedule s_1 from Example 2.

4 Time Complexity

Unfortunately, it is unlikely that there are tractable algorithms that find an optimal serialization order for arbitrary sets of transactions. In this section, we consider a decision variant[2] of our scheduling problem asking for a set $\mathcal{T}$ of transactions and number k if a serializable schedule s for $\mathcal{T}$ exists with span$(s) \leq k$ and show that it is NP-complete.

Schedulability
input: a set of transactions $\mathcal{T}$ and positive integer $k \in \mathbb{N}_0$.
output: *true* if a serializable schedule s for $\mathcal{T}$ with span$(s) \leq k$ exists; *false* otherwise.

By Proposition 8, we can also give a total order Σ for $\mathcal{T}$ with span(canonical$(\Sigma)) \leq k$.

▶ **Theorem 10.** *Schedulability is NP-complete.*

First, we argue that *Schedulability* is straightforwardly in NP, because, if for $\mathcal{T}$ and k a schedule s exists with span$(s) \leq k$, then s trivially serves as polynomial-sized witness.

[2] There exists an analog definition of NP-completeness for optimization problems called NPO-complete. One possible way to show membership in this class is precisely by showing that the decision problem we consider, often called the threshold problem, is an NP-complete decision problem [9, 15, 21].

To show that *Schedulability* is NP-hard, we make a reduction from a simpler problem called 2-*Orientability*. For its definition, let $G = (V, E)$ be an undirected graph and let $\leq_V$ be a total order on its nodes. Then by $\mathsf{OG}(G, \leq_V)$ we denote the *orientation* of G based on $\leq_V$, that is, the directed graph with nodes V and with the directed edges $\{(u, v) \mid \{u, v\} \in E, u \leq_V v\}$. Now, 2-*Orientability* asks for an undirected graph G if there exists a total order $\leq_V$ on the nodes in G such that the length (number of edges) of every directed path in $\mathsf{OG}(G, \leq_V)$ is at most 2.

2-*Orientability*
input: an undirected graph $G = (V, E)$.
output: *true* if a total order $\leq_V$ on V exists such that the length of the longest path in $\mathsf{OG}(G, \leq_V)$ is at most 2; *false* otherwise.

The next result follows from the Gallai-Roy-Vitaver Theorem [5].

▶ **Proposition 11.** *2-Orientability is* NP-hard.

We now conclude the proof for Theorem 10 by showing there exists a polynomial-time reduction from this problem to *Schedulability*.

▶ **Proposition 12.** *2-Orientability* $\leq_P$ *Schedulability.*

Given some graph $G = (V, E)$ as input for 2-*Orientability*, we will explain the construction of a set $\mathcal{T}$ of transactions and will then argue that:

$$G \in \text{2-}\textit{Orientability} \iff (\mathcal{T}, 3|E| + 3) \in \textit{Schedulability}.$$

For the construction, we first fix an arbitrarily chosen order on the edges of E, which allows referring to them as e_1, e_2, …, $e_{|E|}$. We also associate to every edge e_i three different and unique objects x_i, y_i, and z_i. Then, the set $\mathcal{T}$ that we construct, will consist of precisely $|V|$ transactions, such that there is one transaction for each node in V. We write T_v to denote the transaction associated to node $v \in V$. Every transaction T_v consists of precisely $3|E|$ operations, namely

$$\mathtt{W}[\chi_1] \cdots \mathtt{W}[\chi_{|E|}]\, \mathtt{W}[\gamma_1] \cdots \mathtt{W}[\gamma_{|E|}]\, \mathtt{W}[\zeta_1] \cdots \mathtt{W}[\zeta_{|E|}]$$

The objects these operations interact with are chosen as follows: if $v \in e_i$, then χ_i, γ_i and ζ_i are the earlier defined objects x_i, y_i, and z_i, respectively. Otherwise, if $v \notin e_i$, then χ_i, γ_i and ζ_i are (all different) unique objects not occurring anywhere else in our construction.

We remark that this construction is indeed polynomial. To conclude the proof of Proposition 12, we make use of Proposition 8 and the next result.

▶ **Lemma 13.** *For any total order $\leq_V$ on V, the longest path in $\mathsf{OG}(G, \leq_V)$ counts at most two edges if and only if $\mathsf{max\text{-}path}(P_\Sigma) \leq 3|E| + 3$, with $\Sigma = (\mathcal{T}, \{(T_u, T_v) \mid u \leq_V v\})$.*

Some visual intuition for this lemma, in particular the construction of these longest paths through P_Σ, can be seen in Figure 3.

5 Instance Optimal Scheduling

We now turn our attention to instance optimal scheduling algorithms. That is, we will identify a restriction on transaction sets that will enable us to compute schedules with a minimal makespan, for any set adhering to this restriction, in polynomial time.

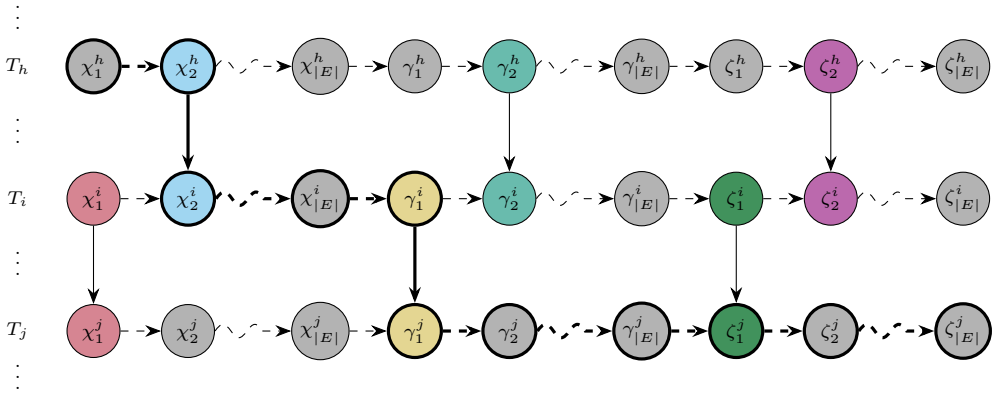


Figure 3 A visualization of a longest path (marked bold) in an arbitrary P_Σ, where the order Σ is defined as per Lemma 13. Each row represents a single transaction T_h, T_i or T_j, sorted top-down consistent with $\leq_V$. Some cross-transaction edges are omitted for clarity of presentation. Waved lines denote one or more intermediate operations. We write χ_j^g, γ_j^g, and ζ_j^g to denote $\mathtt{W}_g[\chi_j]$, $\mathtt{W}_g[\gamma_j]$, and $\mathtt{W}_g[\zeta_j]$, where g is the transaction identifier.

To this extent, we define the *contention points* of $\mathcal{T}$ as a set of objects $\mathcal{C}(\mathcal{T}) \subseteq \mathcal{O}$ such that if o and p are conflicting operations in $\mathcal{T}$, then $\mathsf{obj}(o) \in \mathcal{C}(\mathcal{T})$. The converse does not necessarily have to hold, not every contention point must appear in a conflicting operation. We will refer to an element of $\mathcal{C}(\mathcal{T})$ as a *contention point*, and say that a $T \in \mathcal{T}$ *contends* if T contains an operation on a contention point.

We now define $\mathbb{T}_{\ell,m,n}$ as a set of transaction sets with $\mathcal{T} \in \mathbb{T}_{\ell,m,n}$ if $|T| \leq \ell$ for all $T \in \mathcal{T}$, $|\mathcal{C}(\mathcal{T})| \leq m$, and $|\mathcal{T}| \leq n$. Exclusively in this section, we write $\mathbb{T}_m$ instead of $\mathbb{T}_{\ell,m,n}$, as all results can be phrased such that ℓ is the length of the longest transaction in our set, and we can let $n = |\mathcal{T}|$. We also define $\mathbb{T}_m^+$ with $\mathcal{T} \in \mathbb{T}_m^+$ if $\mathcal{T} \in \mathbb{T}_m$ and T contends for all $T \in \mathcal{T}$.

Given $\mathcal{T} \in \mathbb{T}_1^+$, we can see each $T \in \mathcal{T}$ as a sequence:

$$\underbrace{o_1\ o_2\ \ldots\ o_{r-1}}_{\text{prefix}}\ \mathtt{W}[a]\ \underbrace{o_{r+1}\ \ldots\ o_{\ell-1}\ o_\ell}_{\text{suffix}}$$

where r is the position in T such that $T[r] = \mathtt{W}[a]$, and T consists of a (potentially empty) subsequence of operations, followed by a conflicting operation to the contention point a, and another (potentially empty) subsequence of operations. In the above sequence we can also replace $\mathtt{W}[a]$ by $\mathtt{R}[a]$ as long as it is a conflicting operation, we typically use $\mathtt{W}[a]$ throughout this section, this does not impact the generality of our results. We call the former subsequence the *prefix* of T; the latter is the *suffix* of T. We define $\mathsf{pre}(T)$ and $\mathsf{suf}(T)$ to be the number of operations before the first (respectively, after the last) contention point.

▶ **Definition 14.** *Let $k \in \mathbb{N}_0$. An* allocation of $\mathcal{T} \in \mathbb{T}_1^+$ *to* $[0, k]$ *is an injective function α_k, which maps transactions $T \in \mathcal{T}$ to a $\alpha_k(T) \in [0, k]$ such that $\mathsf{pre}(T) \leq \alpha_k(T)$ and $\mathsf{suf}(T) \leq k - \alpha_k(T)$. We call $\alpha_k(T)$ the* bin *of T.*

We will also denote the inverse of α_k by α_k^{-1}. Notice that, $[0, k]$ means there are $k + 1$ bins in total, as we start counting from 0.

For any allocation of a $\mathcal{T} \in \mathbb{T}_1^+$ to $[0, k]$, we can use the allocation as an ordering on the transactions. Indeed, if $\alpha_k(T_i) \leq \alpha_k(T_j)$ for $\{T_i, T_j\} \subseteq \mathcal{T}$, then we define an order $\mathsf{ord}(\alpha_k) = (\mathcal{T}, \leq_{\alpha_k})$ such that $T_i \leq_{\alpha_k} T_j$. Conversely, we can define an allocation $\mathsf{alloc}(\Sigma, k)$

to $[0, k]$ based on a total order Σ with $\mathsf{span}(\mathsf{canonical}(\Sigma)) \leq k + 1$. Let this total order be $T_1, T_2, \ldots, T_n$. Then for each T_i, with $i \in [n]$, we define $\mathsf{alloc}(\Sigma, k)(T_i) = j$ such that $j \in [0, k]$ is the minimal bin with $\mathsf{pre}(T_i) \leq j$, $\mathsf{suf}(T_i) \leq k - j$, and $\mathsf{alloc}(\Sigma, k)(T_h) < j$ for all $h < i$. We then have the following result.

▶ **Proposition 15.** *There exists an allocation α_k of $\mathcal{T} \in \mathbb{T}_1^+$ to $[0, k]$, if, and only if, there exists a total order Σ of $\mathcal{T}$ such that $\mathsf{span}(\mathsf{canonical}(\Sigma)) \leq k + 1$. Moreover, if α_k exists then $\Sigma = \mathsf{ord}(\alpha_k)$. And, if Σ exists then $\alpha_k = \mathsf{alloc}(\Sigma, k)$.*

The former characterization gives us a tool for constructing an optimal order, provided we have an algorithm for allocations. To that end, we define an algorithm $\mathsf{Allocate}(\mathcal{T}, k)$.

Algorithm Allocate(*transaction set* $\mathcal{T} \in \mathbb{T}_1^+$, *threshold* $k \in \mathbb{N}_0$) : α_k or $\perp$.

```
1 fn SuffixLesserThan(T_i, T_j)
2     if suf(T_i) ≠ suf(T_j) then return suf(T_i) < suf(T_j)
3     return pre(T_i) > pre(T_j)
4 if T = ∅ then return ⊥
5 foreach T ∈ SortedBy(T, SuffixLesserThan) do
6     foreach i ∈ [0, k + 1] do
7         if (pre(T) ≤ i) ∧ (suf(T) ≤ k − i) ∧ ∄T_h ∈ T : α_k(T_h) = i then α_k(T) = i
8     if α_k(T) is undefined then return ⊥
9 return α_k
```

It is clear that $\mathsf{Allocate}(\mathcal{T}, k)$ is sound by construction. That is, if it does not terminate with $\perp$, it terminates with a valid allocation of $\mathcal{T}$ to $[0, k]$. Showing completeness, that is, termination with $\perp$ implies there exists no valid allocation from $\mathcal{T}$ to $[0, k]$, requires slightly more argumentation. The following lemma is a useful tool.

▶ **Lemma 16.** *Let $\mathcal{T} \in \mathbb{T}_1^+$. For every $\mathcal{T}' \subseteq \mathcal{T}$, and every allocation of $\mathcal{T}$ on $[0, k]$, we have that $|\mathcal{T}'| + i + j - 1 \leq k$.*

Completeness follows from assuming a smaller allocation exists despite the algorithm yielding $\perp$, and then showing it contradicts Lemma 16. Proposition 15 ensures no order exists either.

▶ **Theorem 17.** *For any $\mathcal{T} \in \mathbb{T}_1^+$ and $k \in \mathbb{N}_0$, the algorithm $\mathsf{Allocate}(\mathcal{T}, k)$ runs in $O(kn \log(n))$ time and returns an allocation of $\mathcal{T}$ to $[0, k]$ if it exists and $\perp$ otherwise.*

We can repeat the algorithm with a k that starts at the length of the largest transaction, incrementing whenever it yields $\perp$. We terminate when the first allocation is returned. We remark that k is at most $\sum_{T \in \mathcal{T}} |T|$, hence k depends entirely on the transaction set, and therefore its time complexity is polynomial and entirely dependent on $\mathcal{T}$.

Until now, we have considered transactions from $\mathbb{T}_1^+$, but this restriction can be easily relaxed to any $\mathcal{T} \in \mathbb{T}_1$ by only considering the transactions in $\mathcal{T}$ which contend. To obtain an ordering for the full $\mathcal{T}$ we can append the remaining transactions arbitrarily onto the allocation order. The corollary then readily follows from Proposition 15 and Theorem 17.

▶ **Corollary 18.** *If $\mathcal{T} \in \mathbb{T}_1$ and $k \in \mathbb{N}_0$, then $(\mathcal{T}, k) \in$ SCHEDULABILITY is decidable in $O(kn \log(n))$ time.*

The algorithm in this section resembles an instance optimal algorithm for the Job Shop Scheduling problem called *Johnson's Rule* [16]. We illustrate it is subtly different.

▶ **Example 19.** Job shop problem inputs consist of jobs (analogous to our transactions), containing a sequence of operations lasting a duration $x \in \{1, 2, \ldots\}$ on machines (analogous to our objects). Johnson's rule is applicable when all jobs have two operations on two machines in a single order, hence we represent jobs as pairs (x, y) where x and y are the durations spent on the first and second operation.

Consider the job shop instance $\mathcal{J} = \{A = (1, 3), B = (1, 2), C = (2, 1)\}$. Johnson's rule repeatedly picks the operation with the shortest duration and places it in the first free position starting from the front of the schedule if that duration is for the first machine, and places it in the first free position starting from the end of the schedule if it is for the last machine, while breaking ties arbitrarily. A valid schedule would be B, A, C.

If we now create transactions $T_J \in \mathcal{T}$ for each job $(x, y) \in \mathcal{J}$ such that $\mathsf{pre}(T_J) = x$ and $\mathsf{suf}(T_j) = y$, then the allocation for $\mathcal{T}$ matching the order B, A, C is $\alpha_5(T_B) = 1$, $\alpha_5(T_A) = 2$, and $\alpha_5(T_C) = 3$. However, an allocation α_4 for $\mathcal{T}$ is given by our algorithm, namely, $\alpha_4(T_A) = 1$, $\alpha_4(T_B) = 2$, $\alpha_4(T_C) = 3$. Hence, Johnson's rule does not give an optimal schedule for the transaction scheduling problem.

6 Worst-Case Optimal Scheduling

We now look at the transaction scheduling problem from the perspective of worst-case analysis. More specifically, we derive a worst-case optimal bound on the makespan for any transaction set, and in doing so, construct algorithms for any transaction set which yield schedules with a makespan that never exceeds this bound. Additionally, on the worst-behaving transaction sets, the resulting schedules must be optimal.

We define the *worst-case optimal cost* for $\mathbb{T}_{\ell,m,n}$ as

$$\mathsf{wco\text{-}cost}(\mathbb{T}_{\ell,m,n}) = \max\{\mathsf{cost}(\mathcal{T}) \mid \mathcal{T} \in \mathbb{T}_{\ell,m,n}\}.$$

Since we are concerned with a worst-case scenario, we will consider transactions to be long enough for them to contain operations on each contention point, that is, $\ell \geq m$. Alongside this, we always have $n \geq 1$ and $m \geq 0$.

For $\mathbb{T}_{\ell,m,1}$, $\mathbb{T}_{\ell,0,n}$, and $\mathbb{T}_{\ell,1,n}$, this cost is trivial to determine. Indeed, if $n = 1$ the makespan of all schedules is simply ℓ. Similarly, if $m = 0$ we can always execute all transactions entirely in parallel, hence $\mathsf{wco\text{-}cost}(\mathbb{T}_{\ell,0,n}) = \mathsf{wco\text{-}cost}(\mathbb{T}_{\ell,m,1}) = \ell$. Lastly, if $m = 1$, the lone operation on the contention point occurs in the same position in every transaction T in all instances $\mathcal{T}$ for which $\mathsf{cost}(\mathcal{T})$ is maximal. An optimal schedule will shift each operation in each consecutive transaction forwards by one time unit, hence $\mathsf{wco\text{-}cost}(\mathbb{T}_{\ell,1,n}) = \ell + n - 1$ given $n \geq 2$. Determining the worst-case optimal makespan for $m \geq 2$ and $n \geq 2$ is non-trivial, and concerns the rest of this section.

6.1 Extra Notation

We first introduce some notation. Let $\mathbf{x} = x_1\, x_2\, \ldots\, x_k$ be a finite sequence of distinct objects. We will write $\mathbf{x}^{(i)}$ to denote the ith object in $\mathbf{x}$. For a transaction T in a transaction set $\mathcal{T}$, we will let $\vec{T}$ represent the sequence of contention points ordered in the way T accesses them. We write $\mathsf{cpos}(T, i) \in [|T|]$ to refer to the position of the operation on the contention point $\vec{T}^{(i)}$ in some transaction T. Lastly, we define $\mathcal{T}_{\langle \mathbf{x} \rangle}$ as $\{T \mid T \in \mathcal{T}, \vec{T} = \mathbf{x}\}$.

Next, let $\mathbf{x}$ and $\mathbf{y}$ be two length m sequences of distinct objects. We define the *maximal overlap between* $\mathbf{x}$ and $\mathbf{y}$, denoted $\mathsf{max\text{-}ovl}(\mathbf{x}, \mathbf{y})$, as the maximal number $k \in [1, m-1]$ such that for all $j \in [k]$ and all $i \in [j + m - k, m]$ we have $\mathbf{x}^{(i)} \neq \mathbf{y}^{(j)}$; if no such k exists, then

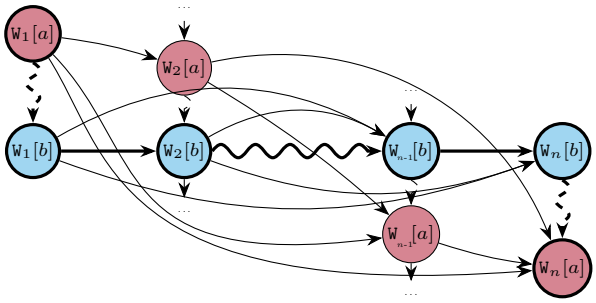


Figure 4 A longest path (bold line) in P_{Σ_2}. Waved lines represent one or multiple intermediate operations.

$\mathsf{max\text{-}ovl}(\mathbf{x}, \mathbf{y}) = 0$. Given a totally ordered finite set of same-length distinct object sequences $(K, \leq_K)$, its *total overlap* $\mathsf{tot\text{-}ovl}(K, \leq_K)$ is defined as the sum $\sum_{i \in [|K|-1]} \mathsf{max\text{-}ovl}(\mathbf{x}_i, \mathbf{x}_{i+1})$, where all $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{|K|}$ are ordered as per Σ. The *maximal total overlap* for K is $\mathsf{max\text{-}tot\text{-}ovl}(K) = \max\{\mathsf{tot\text{-}ovl}(K, \leq_K) \mid \text{with } (K, \leq_K) \text{ a total order on } K\}$.

Finally, for any two disjoint ordered sets $(\mathcal{T}_1, \leq_1)$ and $(\mathcal{T}_2, \leq_2)$, we define the *concatenation* of $\mathcal{T}_1$ and $\mathcal{T}_2$ as $\mathcal{T}_1 \parallel \mathcal{T}_2 = (\mathcal{T}_1 \cup \mathcal{T}_2, \leq_{\mathsf{cat}})$ where $T_i \leq_{\mathsf{cat}} T_j$ is either $T_i \leq_1 T_j$ with $\{T_i, T_j\} \subseteq \mathcal{T}_1$, or $T_i \leq_2 T_j$ with $\{T_i, T_j\} \subseteq \mathcal{T}_2$, or $T_i \in \mathcal{T}_1$ and $T_j \in \mathcal{T}_2$. Notice that this operation is associative, so we will write $\mathcal{T}_1 \parallel \mathcal{T}_2 \parallel \mathcal{T}_3$ for $(\mathcal{T}_1 \parallel \mathcal{T}_2) \parallel \mathcal{T}_3$.

6.2 Two Contention Points or Transactions

Transaction sets containing two transactions or two points of contention (or both) have the following worst-case optimal bound on their makespan.

▶ **Theorem 20.** *We have that* $\mathsf{wco\text{-}cost}(\mathbb{T}_{\ell,m,2}) = 2\ell$ *where* $m \geq 2$ *and* $\ell \geq m$*, and* $\mathsf{wco\text{-}cost}(\mathbb{T}_{\ell,2,n}) = 2\ell + n - 2$ *where* $n \geq 3$ *and* $\ell \geq 2$*.*

If $n = 2$ and $m \geq 2$, then a trivial upper bound is 2ℓ as there are maximally that many operations in $\mathcal{T}$. A matching lower bound can be constructed by taking the set $\{T, T'\}$ where $T = \{\mathtt{W}[a], \ldots, \mathtt{W}[z]\}$ and $T' = \{\mathtt{W}[z], \ldots, \mathtt{W}[a]\}$, where the dots "…" represent read operations on unique objects, and with both transactions of length $\ell \geq m$. Intuitively, T' flips the first and last operations in T. Indeed, both total orders on $\{T, T'\}$ lead to a schedule with an optimal makespan of 2ℓ as it is impossible for the transactions to be concurrently scheduled, that is, $\mathsf{max\text{-}ovl}(\vec{T}, \vec{T'}) = 0$.

For the upper bound when $n \geq 3$ and $m = 2$, we give the construction of an order for $\mathcal{T} \in \mathbb{T}_{\ell,2,n}$. Let $T \in \mathcal{T}$. We have either $\vec{T} = a\,b$ or $\vec{T} = b\,a$. In the former case, we will define $\leq_{\langle a\,b\rangle}$ on $\mathcal{T}_{\langle a\,b\rangle}$ such that transactions T with the largest distance between the first and second contention point, that is $(\mathsf{cpos}(T, 2) - \mathsf{cpos}(T, 1))$, appear first. And, in the latter case, let $\leq_{\langle b\,a\rangle}$ on $\mathcal{T}_{\langle b\,a\rangle}$ such that transactions T with the smallest $(\mathsf{cpos}(T, 2) - \mathsf{cpos}(T, 1))$ appear first. Lastly, we get an order $\Sigma_2 = \mathcal{T}_{\langle a\,b\rangle} \parallel \mathcal{T}_{\langle b\,a\rangle}$. We give a visualization of P_{Σ_2} in Figure 4, which gives intuition for the next result.

▶ **Proposition 21.** *For* $\mathcal{T} \in \mathbb{T}_{\ell,2,n}$ *with* $n \geq 3$ *and* $\ell \geq 2$*:* $\mathsf{span}(\mathsf{canonical}(\Sigma_2)) \leq 2\ell + n - 2$*.*

Lastly, for the lower bound, we construct a transaction set $\mathcal{T}_{\uparrow 2} \in \mathbb{T}_{\ell,2,n}$ with $n > 2$ and $\ell \geq 2$. We will populate the set $\mathcal{T}_{\uparrow 2}$ to reach n transactions by adding k_1 copies of the transaction $T_i = \mathtt{W}_i[a]\ o_1\ o_2\ \ldots\ o_{\ell-2}\ \mathtt{W}_i[b]$, and k_2 copies of $T_j = \mathtt{W}_j[b]\ p_1\ p_2\ \ldots\ p_{\ell-2}\ \mathtt{W}_j[a]$, such that $n = k_1 + k_2$. A precedence graph for any order on $\mathcal{T}_{\uparrow 2}$, along with a longest path in this graph, is given in Figure 5.

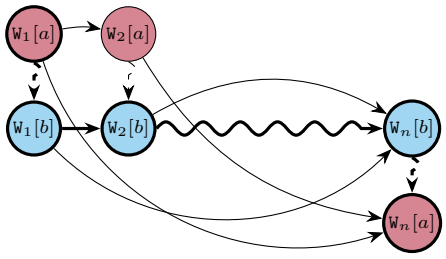


Figure 5 A longest path (bold line) in a precedence graph P_Σ for any total order $\Sigma = (\mathcal{T}_{\uparrow 2}, \leq_{\mathcal{T}_{\uparrow 2}})$. Waved lines denote one or more intermediate operations.

▶ **Proposition 22.** *cost*$(\mathcal{T}_{\uparrow 2}) = 2\ell + n - 2$.

Proof. Let $\Sigma = (\mathcal{T}_{\uparrow 2}, \leq_{\mathcal{T}_{\uparrow 2}})$ be a total order, and refer to $\mathcal{T}_{\uparrow 2}$'s elements by $T_1, T_2, \ldots, T_n$, reflecting this order. Because P_Σ must respect each transaction's order of operations, there exists a path of ℓ nodes between each $\mathtt{W}_i[a]$ and $\mathtt{W}_i[b]$ for $i \in [n]$. And, because the directed edges in P_Σ must respect Σ for conflicting operations, there exists a path $\mathtt{W}_1[a]\,\mathtt{W}_2[a]\,\ldots\,\mathtt{W}_n[a]$ and a path $\mathtt{W}_1[b]\,\mathtt{W}_2[b]\,\ldots\,\mathtt{W}_n[b]$ in P_Σ that both have n nodes. Since there are k_1 transactions T with $\vec{T} = a\,b$ and k_2 transactions T with $\vec{T} = b\,a$, there exists a path $\mathtt{W}_i[a]\,\ldots\,\mathtt{W}_i[b]\,\mathtt{W}_{i+1}[b]\,\ldots\,\mathtt{W}_{i+1}[a]$ in P_Σ of length 2ℓ for some $i \in [n]$. If we outline these paths such as in Figure 5, we see a grid-like pattern with 2ℓ columns and n rows.

Clearly, the longest path starts in $\mathtt{W}_1[a]$, passes through $\mathtt{W}_i[b]$ for all $1 \leq i \leq n$, and ends in $\mathtt{W}_n[a]$. Hence, max-path$(P_\Sigma) = 2\ell + n - 2$. The desired result follows by Proposition 8. ◀

Theorem 20 follows as a corollary of Proposition 21 and Proposition 22. Additionally, we remark that canonical(Σ_2) can be trivially computed in polynomial time.

6.3 Three Contention Points

First, we concern ourselves with the case where $2 < n \leq 6$. The following insight is key.

▶ **Proposition 23.** *Every* $K \subseteq \{a\,b\,c, a\,c\,b, b\,a\,c, b\,c\,a, c\,a\,b, c\,b\,a\}$ *with* $|K| \geq 3$ *satisfies* max-tot-ovl$(K) = |K| - 1$.

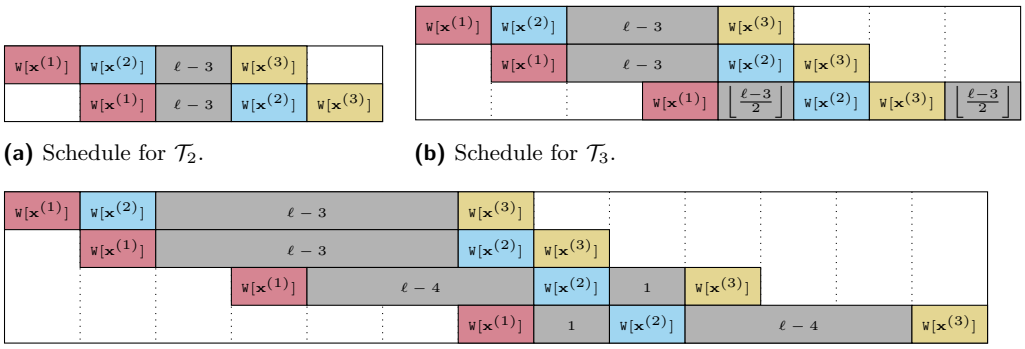
We can use the above result for all $\vec{T}$ in each transaction $T \in \mathcal{T}$ to obtain a transaction order in which the total overlap between transactions is $n - 1$. This results in an upper bound of $n\ell - (n-1) = n\ell - n + 1$. For the lower bound, $\mathcal{T}$ consists of n transactions T of the shape $\{\mathtt{W}[\vec{T}^{(1)}], \ldots, \mathtt{W}[\vec{T}^{(2)}], \mathtt{W}[\vec{T}^{(3)}]\}$, where each $\vec{T}$ is unique in $\mathcal{T}$ ordered as per the above result, and with "…" being reads on unique objects.

▶ **Theorem 24.** *Let* $\ell \geq 3$ *and* $2 < n \leq 6$*, then* wco-cost$(\mathbb{T}_{\ell,3,n}) = n\ell - n + 1$.

Once we allow $n \geq 7$, the new subproblem that arises is one of optimally scheduling all $\mathcal{T}_{\langle \mathbf{x} \rangle}$, since $|\mathcal{T}_{\langle \mathbf{x} \rangle}| \geq 2$ for some $\mathbf{x} = \vec{T}$ and $T \in \mathcal{T}$. Notice that each $T \in \mathcal{T}_{\langle \mathbf{x} \rangle}$ is of the form:

$$T = \overbrace{\ldots}^{\delta_1} \mathtt{W}[\mathbf{x}^{(1)}] \overbrace{\ldots}^{\delta_2} \mathtt{W}[\mathbf{x}^{(2)}] \overbrace{\ldots}^{\delta_3} \mathtt{W}[\mathbf{x}^{(3)}] \overbrace{\ldots}^{\delta_4};$$

where, for all $j \in [4]$, the dots "…" represent $\delta_j \in [0, \ell - 3]$ read operations on unique objects, such that $\delta_1 + \delta_2 + \delta_3 + \delta_4 = \ell - 3$. We write these transactions as tuples $(\delta_1, \delta_2, \delta_3, \delta_4)$ if the $\mathbf{x}$ is irrelevant. To find a lower bound, we need assignments for all δ_j in all $T \in \mathcal{T}_{\langle \mathbf{x} \rangle}$,



(a) Schedule for $\mathcal{T}_2$. **(b)** Schedule for $\mathcal{T}_3$.

(c) Schedule for $\mathcal{T}_4$.

Figure 6 Schedules for $\mathcal{T}_{\langle \mathbf{x} \rangle}$ such that $\mathsf{cost}(\mathcal{T}_{\langle \mathbf{x} \rangle})$ is maximized, for $|\mathcal{T}_{\langle \mathbf{x} \rangle}| \in \{2, 3, 4\}$. Gray boxes with numbers denote there are that amount of read operations on unique objects.

such that $\mathsf{cost}(\mathcal{T}_{\langle \mathbf{x} \rangle})$ is maximal. We now give our best estimates. If $|\mathcal{T}_{\langle \mathbf{x} \rangle}| = 2$, then $\mathcal{T}_2 = \{(0, 0, \ell-3, 0), (0, \ell-3, 0, 0)\}$. If $|\mathcal{T}_{\langle \mathbf{x} \rangle}| = 3$, then this is $\mathcal{T}_3 = \{(0, \lfloor \ell - 3/2 \rfloor, 0, \lfloor \ell - 3/2 \rfloor)\} \cup \mathcal{T}_2$. And finally, when $|\mathcal{T}_{\langle \mathbf{x} \rangle}| = 4$ then $\mathcal{T}_4 = \{(0, \ell-4, 1, 0), (0, 1, \ell-4, 0)\} \cup \mathcal{T}_2$. To go beyond this, we can repeat any $T \in \mathcal{T}_4$ until $|\mathcal{T}_{\langle \mathbf{x} \rangle}| = n$. Figure 6 shows the worst optimal schedules we could find for these sets.

The above $\mathcal{T}_{\langle \mathbf{x} \rangle}$ are concatenated with a maximal total overlap of $n-1$ (*c.f.*, Theorem 24). Hence, the first 5 orders $\mathbf{x}$ each add $\ell - 2 + |\mathcal{T}_{\langle \mathbf{x} \rangle}|$ to the makespan, the final order $\mathbf{y}$ adds $\mathsf{cost}(\mathcal{T}_{\langle \mathbf{y} \rangle})$. This leads to the following lower bound.

▶ **Proposition 25.** *There exist $\mathcal{T} \in \mathbb{T}_{\ell,3,n}$ with $\ell \geq 3$, such that*

$$\mathit{cost}(\mathcal{T}) = \begin{cases} 6\ell + n - 11 & 7 \leq n \leq 17 \\ 6\ell + \lfloor \frac{\ell-3}{2} \rfloor + n - 11 & 18 \leq n \leq 23 \\ 7\ell + n - 15 & 24 \leq n. \end{cases}$$

For the upper bound, we construct an order for $\mathcal{T} \in \mathbb{T}_{\ell,3,n}$ with $n \geq 7$. First, define a total order $(\mathcal{T}_{\langle a\,b\,c \rangle}, \leq_{\langle a\,b\,c \rangle})$ such that the T with the greatest $(\mathsf{cpos}(T, 2) - \mathsf{cpos}(T, 1))$ are placed first. And, we define another total order $(\mathcal{T}_{\langle c\,a\,b \rangle}, \leq_{\langle c\,a\,b \rangle})$ such that the T with the smallest $(\mathsf{cpos}(T, 3) - \mathsf{cpos}(T, 2))$ are placed first. For all remaining $\mathcal{T}_{\langle \mathbf{x} \rangle}$ with $\mathbf{x} \neq a\,b\,c$ and $\mathbf{x} \neq c\,a\,b$ we choose any total order $(\mathcal{T}_{\langle \mathbf{x} \rangle}, \leq_{\langle \mathbf{x} \rangle})$. We then define a total order $\Sigma_3 = \mathcal{T}_{\langle a\,b\,c \rangle} \parallel \mathcal{T}_{\langle b\,c\,a \rangle} \parallel \mathcal{T}_{\langle c\,b\,a \rangle} \parallel \mathcal{T}_{\langle b\,a\,c \rangle} \parallel \mathcal{T}_{\langle a\,c\,b \rangle} \parallel \mathcal{T}_{\langle c\,a\,b \rangle}$ for $\mathcal{T}$. P_{Σ_3} is illustrated in Figure 7.

▶ **Proposition 26.** *For $\mathcal{T} \in \mathbb{T}_{\ell,3,n}$ with $\ell \geq 3$ and $n \geq 7$: $\mathit{span}(\mathit{canonical}(\Sigma_3)) \leq 7\ell + n - 14$.*

6.4 Any Number of Contention Points

First, we notice that the upper bound for three contention points also holds for more than three contention points.

▶ **Proposition 27.** *If $m \geq 4$, $\ell \geq m$, and $3 \leq n < m!$, then for all $\mathcal{T} \in \mathbb{T}_{\ell,m,n}$ it is the case that $\mathit{cost}(\mathcal{T}) \leq n\ell - n + 1$.*

This is not a worst-case optimal bound, and it is not constructive. A better upper or lower bound requires a generalization of Proposition 23 for arbitrary m. It is possible to reduce the hamiltonian path problem–known to be NP-complete–to this problem, and it is therefore difficult to prove constructive results for.

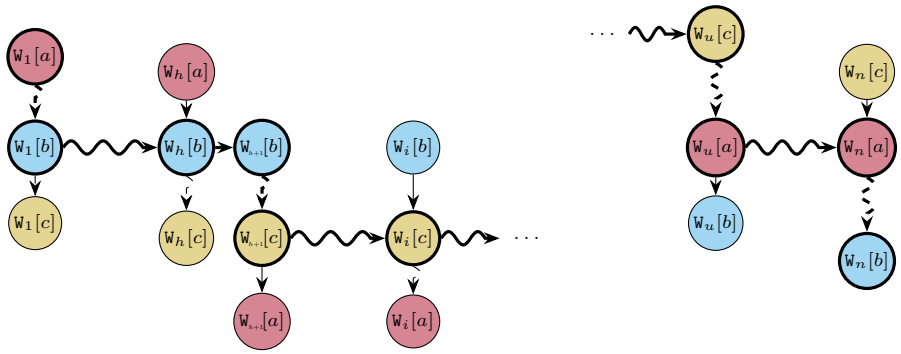


Figure 7 A longest path (bold line) in P_{Σ_3}. Waved lines represent one or multiple intermediate operations. Edges between conflicting operations that do not contribute to the longest path are omitted for clarity.

Whenever we have $m!$ transactions or more, we can make use of an interesting observation relating to object sequences. To this extent, we define the following concept.

▶ **Definition 28.** *Let* $\mathbf{x}$ *and* $\mathbf{y}$ *be equal-length sequences of* $z \geq 1$ *unique objects, then* $\mathbf{y}$ *is an adjacent transposition of* $\mathbf{x}$ *if* $\mathbf{y} = \mathbf{x}^{(1)}\ \mathbf{x}^{(2)}\ \ldots\ \mathbf{x}^{(i+1)}\ \mathbf{x}^{(i)}\ \ldots\ \mathbf{x}^{(z)}$ *for* $i \in [z-1]$.

The notation $\mathbf{x} \curvearrowleft \mathbf{y}$ denotes that $\mathbf{y}$ is an adjacent transposition of $\mathbf{x}$. Intuitively, $\mathbf{y}$ swaps precisely two adjacent elements in $\mathbf{x}$. We will write $\mathbf{x} \curvearrowleft \mathbf{y} \curvearrowleft \mathbf{z}$ to mean that both $\mathbf{x} \curvearrowleft \mathbf{y}$ and $\mathbf{y} \curvearrowleft \mathbf{z}$. Additionally, we call $\mathsf{seq}(m) = \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{m!}$ where each sequence has m operations an *adjacent transposition sequence for* m if $\mathbf{x}_1 \curvearrowleft \mathbf{x}_2 \curvearrowleft \ldots \curvearrowleft \mathbf{x}_{m!}$. Adjacent transpositions have the following property.

▶ **Lemma 29.** *If* $\mathbf{x}$ *and* $\mathbf{y}$ *are equal-length sequences of* $m \geq 3$ *unique objects, and* $\mathbf{x} \curvearrowleft \mathbf{y}$, *then* $\mathsf{max\text{-}ovl}(\mathbf{x}, \mathbf{y}) = m - 2$.

The next insight follows from the above lemma and the Steinhaus-Johnson-Trotter algorithm, which for a permutation of m objects, generates an adjacent transposition sequence for m [17, 28, 29]. This algorithm does not run in polynomial time, as it generates all possible sequences of m objects.

▶ **Lemma 30.** *If* $m \geq 4$, $\ell \geq m$, $n \geq m!$, *and* $\mathcal{T} \in \mathbb{T}_{\ell,m,n}$ *with* $K = \{\vec{T} \mid T \in \mathcal{T}\}$ *containing all equal-length sequences of* m *distinct objects, then* $\mathsf{max\text{-}tot\text{-}ovl}(K) = (n-1)(m-2)$.

The above results imply we can construct a transposition sequence $\mathsf{seq}(m) = \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{m!}$ such that $\mathcal{T}_{\langle \mathbf{x}_1 \rangle} \cup \mathcal{T}_{\langle \mathbf{x}_2 \rangle} \cup \ldots \cup \mathcal{T}_{\langle \mathbf{x}_{m!} \rangle} = \mathcal{T}$. From this, we construct a total order for any $\mathcal{T} \in \mathbb{T}_{\ell,m,n}$, with $n > m!$ and $\ell \geq m$. For $\mathcal{T}_{\langle \mathbf{x}_1 \rangle}$ we define the total order $(\mathcal{T}_{\langle \mathbf{x}_1 \rangle}, \leq_{\langle \mathbf{x}_1 \rangle})$ such that a transaction $T \in \mathcal{T}_{\langle \mathbf{x}_1 \rangle}$ appears earlier in $\leq_{\langle \mathbf{x}_1 \rangle}$ if $(\mathsf{cpos}(T, 2) - \mathsf{cpos}(T, 1))$ is larger. And, for $\mathcal{T}_{\langle \mathbf{x}_{m!} \rangle}$ we define the total order $(\mathcal{T}_{\langle \mathbf{x}_{m!} \rangle}, \leq_{\langle \mathbf{x}_{m!} \rangle})$ such that that $T \in \mathcal{T}_{\langle \mathbf{x}_{m!} \rangle}$ appears earlier in $\leq_{\langle \mathbf{x}_{m!} \rangle}$ if $(\mathsf{cpos}(T, m) - \mathsf{cpos}(T, m-1))$ is smaller. For all other $\mathcal{T}_{\langle \mathbf{x}_i \rangle}$ with $i \in [2, m! - 1]$, we choose an arbitrary total order $(\mathcal{T}_{\langle \mathbf{x}_i \rangle}, \leq_{\langle \mathbf{x}_i \rangle})$. Lastly, we define $\Sigma_{\mathsf{m}} = \mathcal{T}_{\langle \mathbf{x}_1 \rangle} \parallel \mathcal{T}_{\langle \mathbf{x}_2 \rangle} \parallel \ldots \parallel \mathcal{T}_{\langle \mathbf{x}_{m!} \rangle}$ to be a total order on $\mathcal{T}$. This then gives the next bounds.

▶ **Theorem 31.** *If* $m \geq 4$, $\ell \geq m$, *and* $n = m!$, *then* $\mathsf{wco\text{-}cost}(\mathbb{T}_{\ell,m,n}) = n\ell - (n-1)(m-2)$.

If $n > m!$ then the above scheduling method gives us the following upper bound.

▶ **Theorem 32.** *If* $m \geq 4$, $\ell \geq m$, *and* $n > m!$, *then for all* $\mathcal{T} \in \mathbb{T}_{\ell,m,n}$ *we have that* $\mathsf{span}(\mathsf{canonical}(\Sigma_{\mathsf{m}})) \leq (m! + m - 2)\ell + n - (m-1)(m! + m - 2)$.

Table 1 summarizes the results given in this section.

7 Related Work

For years, serializability has been the gold standard for ensuring data consistency. In brief, a transaction schedule is serializable if it is equivalent (for some notion of equivalence) to some serial execution of these transactions. One straightforward notion of equivalence is view equivalence, which states that two schedules are equivalent if all read operations observe the same values and the final database state is the same for both schedules. A famous result by Papadimitriou [24] states that view serializability is NP-complete. Although view serializability is a natural notion of serializability, this negative result has led to the adoption of conflict serializability as the de facto standard for serializability in practice, and is frequently simply referred to as "serializability". Contrasting view equivalence, conflict equivalence puts a more direct restriction on the order of operations. In particular, all pairs of conflicting operations must be ordered the same in both schedules. It is well known that conflict serializability of a schedule can be decided in polynomial time, and that every conflict serializable schedule is view serializable as well [23]. *The serializability problem is orthogonal to the scheduling problem, as the former is concerned with deciding serializability for a given schedule, while the latter is concerned with finding the most efficient serializable schedule for a given set of transactions.*

To the best of our knowledge, the only prior results on scheduling are also from Papadimitriou [24], who established that a scheduler which allows precisely all view serializable schedules will likely be inefficient unless P = NP. This result is to be expected, given the NP-completeness of deciding view serializability. Contrasting these results by Papadimitriou, we restrict the search space of schedules to those that are conflict serializable, a property which can be tested in polynomial time. *Our NP-completeness result therefore strengthens this earlier work, and shows that a tractable notion of serializability does not necessarily lead to tractable scheduling algorithms.*

Orthogonal to finding more efficient schedules that guarantee (conflict) serializability, database systems frequently offer lower levels of isolation, such as read committed or snapshot isolation. In fact, many commercial database systems offer these lower isolation levels by default, and some do not even support conflict serializability [3]. The relaxed consistency guarantees of these lower isolation levels allow for more concurrency, and therefore higher throughput. This performance increase comes at a cost, however, as these lower isolation levels no longer guarantee serializability. Recent work from the database theory community has looked into the transactional robustness problem [11, 18, 19, 30]. The robustness problem is concerned with the following question: given a set of transactions and a specific isolation level, is every allowed schedule (i.e., consistent with the isolation level) a serializable schedule? This robustness property guarantees that the set of transactions can be safely executed under the lower isolation level, thereby increasing performance, while still ensuring serializability. The allocation problem [11, 31] extends upon this problem by considering a setting where isolation levels can be mixed (i.e., different transactions are allocated to different isolation levels), and tries to identify the optimal allocation that still guarantees serializability for the given workload. We emphasize that this line of work relies on an existing scheduler. A key assumption is that, although this scheduler is guaranteed to produce a schedule in line with the chosen isolation levels, the produced output schedule cannot be chosen. Instead, robustness involves verifying serializability of every possible output schedule. *The robustness and allocation problems are orthogonal to the scheduling problem, as they rely on an existing scheduler to execute transactions, rather than directly constructing an efficient serializable schedule over the provided workload.*

Outside of the database literature, the job shop scheduling problem [4, 25] is the closest relative to the transaction scheduling problem. Indeed, one can see objects as machines, transactions as jobs, and operations as always lasting a single unit of time. The associated decision problem is similarly known to be NP-complete, even when limited to 3 machines [27], known to be approximable only if the operations are unit-time [20, 22], and has a polynomial time approximation scheme (PTAS) if the maximal job length and the number of machines is fixed [13]. However, the results do not carry over. *The job shop scheduling problem is distinct because its resulting schedules are not conflict serializable.*

The only systems doing explicit scheduling based on makespan in practice, to the best of our knowledge, are the system from Cheng et al. [6] and a prototype system discussed in a workshop article [2]. The former system [6] relies on a heuristic that clusters transactions based on the most frequently occuring contention point, and has seen promising performance. *This demonstrates the viability of scheduling algorithms for transactions, and of utilizing contention points to perform workload analysis.*

8 Discussion

We return to our central questions. First, *"What is the complexity of finding a conflict serializable schedule of minimal makespan?"*. We have shown there is a factorial time algorithm for this problem; that the decision variant is NP-complete; and lastly, that there is an instance optimal algorithm for single contention point transaction sets.

Secondly, "*Is there a bound on the optimal makespan?*". Our bounds in Table 1 indicate that if the number of transactions is low, then worst-case optimal schedules are close to serial. This gets progressively better once $n \geq m!$, as the increased number of transactions enables more parallellism. Moreover, the most sensitive parameter is m. Hence, workloads with many contention points are likely to make all transaction scheduling approaches struggle.

Open Problems

Many interesting problems remain open. There is still a gap between the number of contention points needed in the construction used for Theorem 10, and those needed for the algorithm in Section 5. A possibility would be extending techniques for job shop scheduling's proofs. Given that the problem is known to be NP-hard for 3 machines, it may lead to similar results.

Furthermore, there are missing worst-case optimal bounds for the sets $\mathbb{T}_{\ell,3,n}$ with $7 \leq n$, and $\mathbb{T}_{\ell,m,n}$ in case $n \neq m!$ and $3 \leq n$. We suspect that the upper bounds can be lower than the ones provided. Moreover, the upper bound provided in the case of $3 \leq n < m!$ for arbitrary m is not constructive.

The work around bounds is necessary to investigate the approximation complexity of transaction scheduling. This domain has many interesting results for job shop scheduling (*c.f.*, Mastrolilli et al. [22] and Fishkin et al. [12, 13]). It involves constructing approximation-preserving reductions (*e.g.*, L-reductions [9, 15]) from existing problems in discrete optimization, which enable us to decide membership in the class of approximable optimization problems (APX), or the class that admits a polynomial time approximation scheme (PTAS). The former would show there are tractable algorithms for which the output schedule has a makespan within a constant factor of its optimal makespan. The latter implies that we can find a tractable algorithm that gets arbitrarily close to the optimal makespan.

Lastly, the algorithms provided in this work require knowledge of the transaction set before execution. To bridge this gap, queuing theory may be an inspiration [14]. This means entry of transactions into the system becomes a stochastic process, increasing complexity.

References

1 Dana Van Aken, Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudré-Mauroux. Benchpress: Dynamic workload control in the oltp-bench testbed. In Timos K. Sellis, Susan B. Davidson, and Zachary G. Ives, editors, *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 1069–1073. ACM, 2015. doi:10.1145/2723372.2735354.

2 Tim Baccaert and Bas Ketsman. Cascade: Optimal transaction scheduling for high-contention workloads. In *PhD Symposium of the 40th IEEE International Conference on Data Engineering, ICDE 2024, Utrecht, The Netherlands, May 13-16, 2024*, pages 5634–5638. IEEE, 2024. doi:10.1109/ICDE60146.2024.00452.

3 Peter Bailis, Aaron Davidson, Alan D. Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. Highly available transactions: Virtues and limitations. *Proc. VLDB Endow.*, 7(3):181–192, 2013. doi:10.14778/2732232.2732237.

4 Peter Brucker. *Scheduling algorithms (4. ed.)*. Springer, 2004.

5 Gerard J. Chang, Li-Da Tong, Jing-Ho Yan, and Hong-Gwa Yeh. A Note on The Gallai-Roy-Vitaver Theorem. *Discret. Math.*, 256(1-2):441–444, 2002. doi:10.1016/S0012-365X(02)00386-2.

6 Audrey Cheng, Aaron N. Kabcenell, Jason Chan, Xiao Shi, Peter D. Bailis, Natacha Crooks, and Ion Stoica. Towards optimal transaction scheduling. *Proc. VLDB Endow.*, 17(11):2694–2707, 2024. doi:10.14778/3681954.3681956.

7 Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with YCSB. In Joseph M. Hellerstein, Surajit Chaudhuri, and Mendel Rosenblum, editors, *Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC 2010, Indianapolis, Indiana, USA, June 10-11, 2010*, pages 143–154. ACM, 2010. doi:10.1145/1807128.1807152.

8 Transaction Processing Performance Council. Tpc benchmark c standard specification revision 5.2, 2006.

9 Pierluigi Crescenzi. A short guide to approximation preserving reductions. In *Proceedings of the Twelfth Annual IEEE Conference on Computational Complexity, Ulm, Germany, June 24-27, 1997*, pages 262–273. IEEE Computer Society, 1997. doi:10.1109/CCC.1997.612321.

10 Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudré-Mauroux. Oltp-bench: An extensible testbed for benchmarking relational databases. *Proc. VLDB Endow.*, 7(4):277–288, 2013. doi:10.14778/2732240.2732246.

11 Alan D. Fekete. Allocating isolation levels to transactions. In Chen Li, editor, *Proceedings of the Twenty-fourth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 13-15, 2005, Baltimore, Maryland, USA*, pages 206–215. ACM, 2005. doi:10.1145/1065167.1065193.

12 Aleksei V. Fishkin, Klaus Jansen, and Monaldo Mastrolilli. On minimizing average weighted completion time: A PTAS for the job shop problem with release dates. In Toshihide Ibaraki, Naoki Katoh, and Hirotaka Ono, editors, *Algorithms and Computation, 14th International Symposium, ISAAC 2003, Kyoto, Japan, December 15-17, 2003, Proceedings*, volume 2906 of *Lecture Notes in Computer Science*, pages 319–328. Springer, 2003. doi:10.1007/978-3-540-24587-2_34.

13 Aleksei V. Fishkin, Klaus Jansen, and Monaldo Mastrolilli. Grouping techniques for scheduling problems: Simpler and faster. *Algorithmica*, 51(2):183–199, 2008. doi:10.1007/s00453-007-9086-6.

14 Donald Gross, John F. Shortle, James M. Thompson, and Carl M. Harris. *Fundamentals of Queueing Theory, Fourth Edition*. Wiley Series in Probability and Statistics. Wiley, 2008. doi:10.1002/9781118625651.

15 Juraj Hromkovic. *Algorithmics for Hard Problems - Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2001. doi:10.1007/978-3-662-04616-6.

16 S. M. Johnson. Optimal Two- and Three-stage Production Schedules with Setup Times Included. *Naval Research Logistics Quarterly*, 1(1):61–68, 1954.

17 Selmer M. Johnson. Generation of permutations by adjacent transposition. *Mathematics of Computation*, 17(83):282–285, 1963.

18 Bas Ketsman, Christoph Koch, Frank Neven, and Brecht Vandevoort. Concurrency control for database theorists. *SIGMOD Rec.*, 51(4):6–17, 2022. doi:10.1145/3582302.3582304.

19 Bas Ketsman, Christoph Koch, Frank Neven, and Brecht Vandevoort. Deciding robustness for lower SQL isolation levels. *ACM Trans. Database Syst.*, 47(4):13:1–13:41, 2022. doi:10.1145/3561049.

20 Frank Thomson Leighton, Bruce M. Maggs, and Andréa W. Richa. Fast algorithms for finding o(congestion + dilation) packet routing schedules. *Comb.*, 19(3):375–401, 1999. doi:10.1007/s004930050061.

21 J.K. Lenstra and A.H.G. Rinnooy Kan. Computational complexity of discrete optimization problems. In P.L. Hammer, E.L. Johnson, and B.H. Korte, editors, *Discrete Optimization I*, volume 4 of *Annals of Discrete Mathematics*, pages 121–140. Elsevier, 1979.

22 Monaldo Mastrolilli and Ola Svensson. Hardness of approximating flow and job shop scheduling problems. *J. ACM*, 58(5):20:1–20:32, 2011. doi:10.1145/2027216.2027218.

23 Christos Papadimitriou. *The theory of database concurrency control*. Computer Science Press, Inc., USA, 1986.

24 Christos H. Papadimitriou. The serializability of concurrent database updates. *J. ACM*, 26(4):631–653, 1979. doi:10.1145/322154.322158.

25 Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 2022.

26 Luyi Qu, Yuming Li, Rong Zhang, Ting Chen, Ke Shu, Weining Qian, and Aoying Zhou. Application-oriented workload generation for transactional database performance evaluation. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*, pages 420–432. IEEE, 2022. doi:10.1109/ICDE53745.2022.00036.

27 Yuri N. Sotskov and Natalia V. Shakhlevich. Np-hardness of shop-scheduling problems with three jobs. *Discret. Appl. Math.*, 59(3):237–266, 1995. doi:10.1016/0166-218X(95)80004-N.

28 Hugo Steinhaus. *One hundred problems in elementary mathematics*. Basic Books, Inc., Publishers, New York, 1964.

29 H. F. Trotter. Algorithm 115: Perm. *Commun. ACM*, 5(8):434–435, August 1962. doi:10.1145/368637.368660.

30 Brecht Vandevoort. *Optimizing Concurrency Control: Robustness Against Read Committed Revisited*. PhD thesis, Hasselt University, Belgium, 2021. URL: https://hdl.handle.net/1942/35460.

31 Brecht Vandevoort, Bas Ketsman, and Frank Neven. Allocating isolation levels to transactions in a multiversion setting. In Floris Geerts, Hung Q. Ngo, and Stavros Sintos, editors, *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2023, Seattle, WA, USA , June 18-23, 2023*, pages 69–78. ACM, 2023. doi:10.1145/3584372.3588672.

32 Gerhard Weikum and Gottfried Vossen. *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. Morgan Kaufmann, 2002.

33 Siyang Weng, Qingshuai Wang, Luyi Qu, Rong Zhang, Peng Cai, Weining Qian, and Aoying Zhou. Lauca: A workload duplicator for benchmarking transactional database performance. *IEEE Trans. Knowl. Data Eng.*, 36(7):3180–3194, 2024. doi:10.1109/TKDE.2024.3360116.

34 Huidong Zhang, Luyi Qu, Qingshuai Wang, Rong Zhang, Peng Cai, Quanqing Xu, Zhifeng Yang, and Chuanhui Yang. Dike: A benchmark suite for distributed transactional databases. In Sudipto Das, Ippokratis Pandis, K. Selçuk Candan, and Sihem Amer-Yahia, editors, *Companion of the 2023 International Conference on Management of Data, SIGMOD/PODS 2023, Seattle, WA, USA, June 18-23, 2023*, pages 95–98. ACM, 2023. doi:10.1145/3555041.3589710.

A Enumeration Scheduling

To construct an optimal serializable schedule, we can enumerate all possible linear orders $\Sigma = (\mathcal{T}, \leq_{\mathcal{T}})$ for $\mathcal{T}$ and pick one such order with a minimal $\mathsf{max\text{-}path}(P_\Sigma)$. This is precisely the idea behind the *enumerative scheduling* algorithm EnumSchedule.

Algorithm EnumSchedule(*transaction set* $\mathcal{T}$) : *schedule for* $\mathcal{T}$.

```
1  if T = ∅ then
2      return empty schedule
3  let Σ = (T, ≤_T) a linear order
4  let m = max-path(P_Σ)
5  foreach linear order (T, ≤_T) do
6      let k = max-path(P_Σ)
7      if k < m then
8          let m = k
9          let Σ = (T, ≤_T)
10 return canonical(Σ)
```

▶ **Proposition 33.** *Let $\mathcal{T}$ be a set of transactions, then $\mathsf{EnumSchedule}(\mathcal{T})$ computes a serializable schedule of minimal makespan in $O(|\mathcal{T}|! + (\sum_{T\in\mathcal{T}} |T|)^2)$ time.*

Proof. First, observe that $\mathsf{EnumSchedule}(\mathcal{T})$ is trivially serializable due to Proposition 8.

Furthermore, $\mathsf{span}(\mathsf{EnumSchedule}(\mathcal{T}))$ is minimal. Indeed, we consider all possible linear orders $\Sigma = (\mathcal{T}, \leq_{\mathcal{T}})$, hence, by Proposition 8 the order Σ for which $\mathsf{span}(\Sigma) = \mathsf{cost}(\mathcal{T})$ will be considered by the algorithm.

Finally, the time complexity follows from the fact that we consider all total orders on $\mathcal{T}$, of which there are precisely $|\mathcal{T}|!$. For each order Σ, we compute the longest vertex path in its accompanying precedence graph $P_\Sigma = (V, E)$. Since P_Σ is acyclic, we can compute this by doing a depth-first post-order traversal which has a time complexity of $O(|V| + |E|)$. However, we know $|E| \leq |V| \cdot (|V| - 1)$ hence this is equivalent to $O(|V|^2)$. Furthermore, $|V|$ is equal to $\sum_{T\in\mathcal{T}} |T|$, which gives an overall bound of $O(|\mathcal{T}|! + (\sum_{T\in\mathcal{T}} |T|)^2)$. ◀

B Missing Proofs for Section 3

B.1 Proof for Proposition 8

▶ **Proposition 34.** *Let $\mathcal{T}$ be a set of transactions. For every serializable schedule s for $\mathcal{T}$, we have that $\mathsf{max\text{-}path}(P_s) \leq \mathsf{span}(s)$. If s is dense, then $\mathsf{span}(s) = \mathsf{max\text{-}path}(P_s)$.*

Proof. Let $s = A_1, \ldots, A_k$ be some serializable schedule. By definition, $\mathsf{span}(s) = k$. The equality $\mathsf{span}(s) \geq \mathsf{max\text{-}path}(P_s)$ follows from a simple inductive argument showing that for the longest directed path $v_1, \ldots v_m$ in P_s, the set A_i cannot contain an operation v_j with $i < j$, thus implying $\mathsf{max\text{-}path}(P_s) = m \leq k = \mathsf{span}(s)$.

By definition of dense schedule, for every $o \in A_i$, with $i > 1$, there is a $p \in A_{i-1}$ such that (p, o) is an edge in the precedence graph. The latter immediately implies that there is a directed path with k nodes in P_s, which means $\mathsf{span}(s) \leq \mathsf{max\text{-}path}(P_s)$. ◀

▶ **Proposition 35.** *Let $\mathcal{T}$ be a set of transactions. There exists an ordering $\leq_{\mathcal{T}}$ over $\mathcal{T}$ such that $\mathsf{max\text{-}path}(P_\Sigma) = \mathsf{cost}(\mathcal{T})$, with $\Sigma = (\mathcal{T}, \leq_{\mathcal{T}})$.*

Proof. Let s be some serializable schedule for $\mathcal{T}$ with $\mathsf{span}(s) = \mathsf{cost}(\mathcal{T})$. We assume that s is dense. The latter is w.l.o.g., because otherwise we can repeatably pick a problematic operation and move it one step to the left, until the resulting schedule is dense. Clearly, the resulting schedule s' is equivalent to s and thus remains conflict-serializable. We have $\mathsf{span}(s') \leq \mathsf{span}(s)$ and thus $\mathsf{span}(s') = \mathsf{span}(s)$ by choice of s.

It now follows from Proposition 34 that $\mathsf{cost}(\mathcal{T}) = \mathsf{span}(s) = \mathsf{max\text{-}path}(P_s)$ and from Proposition 6 that $\mathsf{max\text{-}path}(P_s) = \mathsf{max\text{-}path}(P_\Sigma)$ for some $\Sigma = (\mathcal{T}, \leq_\mathcal{T})$. ◀

C Missing Proofs for Section 4

C.1 Proof for Lemma 13

▶ **Lemma 36.** *For any total order $\leq_V$ on V, the longest path in $\mathsf{OG}(G, \leq_V)$ counts at most two edges if and only if $\mathsf{max\text{-}path}(P_\Sigma) \leq 3|E| + 3$, with $\Sigma = (\mathcal{T}, \{(T_u, T_v) \mid u \leq_V v\})$.*

Proof. To follow along with the proof, it will be helpful to have some visual intuition about the structure of P_Σ. Therefore, we recall that the nodes of P_Σ coincide with the operations of transactions in $\mathcal{T}$ and that an edge (o, p) in P_Σ either means that p is the direct successor of o in some transaction, or that o and p are conflicting operations from different transactions. More precisely, in the latter case it follows from the construction of Σ that $o \in T_v$ and $p \in T_u$ with $v \leq_V u$ and with $o = T_v[i]$ and $p = T_u[i]$ for some integer i. We call the latter edge a cross transaction edge from T_v to T_u.

(If.) The proof is by contraposition, i.e., we show that if $\mathsf{max\text{-}path}(P_\Sigma) > 3|E|+3$ than there is a path $\mathbf{p}$ in $\mathsf{OG}(G, \leq_V)$ with three edges. For this, first recall that $\mathsf{max\text{-}path}(P_\Sigma) \geq 3|E|+3$ implies existence of a directed path $\mathbf{q}$ in P_Σ with $3|E| + 3$ nodes, and hence with a length of $3|E| + 2$. Now, making use of the earlier developed intuition about P_Σ, we observe that every path $\mathbf{q}$ of length k contains at least $k - 3.|E| - 1$ cross transaction edges (since every individual transaction is represented in P_Σ by precisely $3|E| - 1$ edges, and by the fact that cross transaction edges are always from transactions T_v to T_u with $v \leq_V u$.). Since in our case $k \geq 3|E| + 2$ it follows that there are at least three cross transaction edges. Hence, we can assume that there is an edge (o_1, o_2), (p_1, p_2) and (q_1, q_2) with $o_1 \in T_v$, $o_2, p_1 \in T_u$, $p_2, q_1 \in T_w$ and $q_2 \in T_s$, which implies that a directed path $(u, v)\,(v, w)\,(w, s)$ in G exists, which is clearly of length three.

(Only-if.) This direction is by contraposition as well, hence, we show that if there exists a directed path in $\mathsf{OG}(G, \leq_V)$ of length ≥ 3 then there exists a directed path in P_Σ of length $\geq 3|E| + 2$. For this, let $\mathbf{p} = (u, v)\,(v, w)\,(w, s)$ be such a path of precisely length three, with $e_{j_1} = \{u, v\}$, $e_{j_2} = \{v, w\}$, and $e_{j_3} = \{w, s\}$. Then, we construct a directed path $\mathbf{q}$ in P_Σ as follows: first traverse the consecutive operations of transaction T_u up to (and including) its operation $\mathtt{W}_u[x_{j_1}]$. Then traverse the cross transaction edge $(\mathtt{W}_u[x_{j_1}], \mathtt{W}_v[x_{j_1}])$, followed by the consecutive operations of transaction T_v consecutive to $\mathtt{W}_v[x_{j_1}]$ till (and including) operation $\mathtt{W}_v[y_{j_2}]$. Then analogously as before, traverse the cross transaction edge $(\mathtt{W}_v[y_{j_2}], \mathtt{W}_w[y_{j_2}])$ followed by consecutive operations of $\mathtt{W}_w[y_{j_2}]$ in T_w till (and including) $\mathtt{W}_w[z_{j_3}]$ after which yet another cross transaction edge $(\mathtt{W}_w[z_{j_3}], \mathtt{W}_s[z_{j_3}])$ is traversed, followed by the remaining operations of T_s consecutive to $\mathtt{W}_s[z_{j_3}]$. It can be verified that the constructed path has a length of precisely $3|E| + 2$, which concludes the proof. ◀