# A Note On The Transaction Backout Problem

Mark W. Krentel
TR 86-737
February 1986

Department of Computer Science
Cornell University
Ithaca, NY 14853

# A Note on the Transaction Backout Problem

*Mark W. Krentel*

Department of Computer Science
Cornell University
Ithaca, New York 14853

## ABSTRACT

The transaction backout problem arises in the area of distributed databases. Suppose failures partition a data-redundant distributed database, and each partition continues to function as if it were the entire database. When the database is reconnected, the transactions executed by different partitions may not be serializable, and hence it may be necessary to back out some of the transactions. The transaction backout problem is to remove the smallest set of transactions that will leave the remaining ones serializable. The general problem is NP-complete, and in this paper we show that the special case of a fixed-size database can be solved in polynomial time by dynamic programming.

## I. Introduction.

An important problem in the field of distributed databases is dealing with failures. Suppose we implement a database with several sites and communication links between the sites. Site or link failures may partition the network into disconnected pieces, and we are then faced with the problem of coping with these failures. As long as every site can communicate with every other site, the database can guarantee a serializable schedule of transactions. However, if one site loses communication, it can no longer coordinate its actions with the rest of the database. This site must then either suspend actions on some data items or run the risk of losing serializability.

S. Davidson and H. Garcia-Molina propose a solution to this problem[DG]. In their solution, each site keeps a copy of each data item; therefore, when a failure occurs, each partition functions as if it were the entire database. A connected set of sites can guarantee that the transactions within its partition are serializable. However, if two sites are in disconnected partitions, then it is possible that their collective transactions are not serializable. For example, let $T_1$ write data item $a$ in one partition, and let $T_2$ also write $a$ in the other partition. If $T_1$ precedes $T_2$, then $T_2$ would have read the value written by $T_1$ when in fact $T_2$ reads the last value written before the database is partitioned. Thus we see that $T_1$ and $T_2$ cannot coexist in any serializable schedule.

Each site in Davidson and Garcia-Molina's solution keeps a log of its transactions and the data items that they read and write. When the network reestablishes communication, the partitions broadcast their actions and determine conflicts with other sites. We use serializability of the database as the criterion of correctness; therefore, we must back out the conflicting transactions to establish serializability. This solution thus requires some algorithm for deciding which transactions to remove and which to keep. We assume that we want to remove as few transactions as possible, so we state the transaction backout problem as finding the largest serializable subset of transactions.

Unfortunately, the problem of finding the largest serializable set of transactions is an NP-complete problem[AHU, DG, W]. Essentially, this problem is the same as finding the largest acyclic subgraph of a given graph. The general transaction backout problem assumes no upper bound on the number of items in the database. Davidson and Garcia-Molina leave the special case of assuming a fixed-size database as an open problem. Here we show that this problem is in polynomial time. The technique we use is very similar to dynamic programming[AHU]. We first show that a bound on the database gives a bound on the number of transactions needed to produce an unserializable result. The problem then reduces to finding the largest subset of transactions that does not satisfy a certain finite pattern. This last problem has a polynomial-time solution.

## II. Definitions.

The main result we wish to show is the polynomial-time algorithm for the transaction backout problem. We first need to state the problem and convert it to an equivalent problem in graph theory. We only consider the case of two partitions here.

**Definition.** An *instance* $I_{tbp} = (D, S_1, S_2)$ *of the transaction backout problem* is defined by:

(1) A set $D = \{a_1, \ldots, a_K\}$ of data items.

(2) Two ordered schedules of transactions $T_{11}, \ldots, T_{1n}$ and $T_{21}, \ldots, T_{2m}$. We assume that the two schedules are individually serializable and that we are given serial schedules for each of them.

(3) Two sets $Read(T_i)$, $Write(T_i) \subseteq D$ for each transaction $T_i$, which are the data items read and written by transaction $T_i$. We assume that $Read(T_i) \supseteq Write(T_i)$.

Suppose $I_{tbp} = (D, S_1, S_2)$ is an instance of the transaction backout problem, and $S_1$ and $S_2$ are the schedules of transactions in the two partitions. We use serializability as our notion of correctness, so assume that $S_1$ and $S_2$ are serial schedules. Although $S_1$ and $S_2$ are individually serializable, it is possible there there is no serial schedule containing both of them. The transaction backout problem is thus to find the largest serializable set of transactions for $S_1 \cup S_2$. We can define the problem more concretely by stating it in terms of graph theory.

**Definition.** Let $I_{tbp} = (D, S_1, S_2)$ be an instance of the transaction backout problem. The *precedence graph* $G = (V, E)$ associated with it is defined by:

(1) $V = \{T_{11}, \ldots, T_{1n}, T_{21}, \ldots, T_{2m}\}$.

(2) $E$ = {ripple edges} ∪ {precedence edges} ∪ {interference edges} where

   (i) There is a ripple edge $T_{ai} \rightarrow T_{aj}$ if $i<j$ and $\exists\ d \in D$ s.t. $d \in Write(T_{ai}) \cap Read(T_{aj})$.

   (ii) There is a precedence edge $T_{ai} \rightarrow T_{aj}$ if $i<j$ and $\exists\ d \in D$ s.t. $d \in Read(T_{ai}) \cap Write(T_{aj})$.

   (iii) There is an interference edge $T_{1i} \rightarrow T_{2j}$ if $\exists\ d \in D$ s.t. $d \in Read(T_{1i}) \cap Write(T_{2j})$, and similarly for $T_{2i} \rightarrow T_{1j}$.

Let $G = (V, E)$ be a precedence graph. In general, an edge $T_i \rightarrow T_j$ implies that $T_i$ must precede $T_j$ in any serial schedule. If $T_i$ precedes $T_j$ and $T_i$ reads a data item that $T_j$ later writes, then $T_j$ would change the value that $T_i$ reads. Thus $T_i$ must precede $T_j$ in any equivalent serial schedule, and we use a precedence edge to indicate this relationship. Interference edges are similar. If $T_i$ precedes $T_j$ and $T_i$ writes a value later read by $T_j$, then clearly $T_i$ must come before $T_j$ in any serial schedule. Furthermore, since $T_j$ acts on information provided by $T_i$, if we undo $T_i$ we must also undo $T_j$. We indicate this relationship by a ripple edge. We say that $H \subseteq V$ is *ripple closed* if $A \in H$ and a ripple edge $A \rightarrow B$ imply that $B \in H$. We can now state the transaction backout problem in terms of graph theory.

**Definition.** Let $G = (V, E)$ be the precedence graph for $I_{tbp} = (D, S_1, S_2)$. We say that $I_{tbp}$ *has a solution of size k* if there exists $H \subseteq V$ such that (i) $|H| \leq k$, (ii) $H$ is ripple closed in $G$, and (iii) $G \setminus H$ is acyclic.

That the graph-theory version of the transaction backout problem is equivalent to the previous definition follows from a result of Davidson and Garcia-Molina.

**Theorem.** *Let* $I_{tbp} = (D, S_1, S_2)$ *be an instance of the transaction backout problem, and let*

*$G=(V,E)$ be its precedence graph. Then $S_1 \cup S_2$ is serializable iff $G$ is acyclic.*

Our notion of a precedence graph is slightly different from the definition given by Wright[W] or by Davidson and Garcia-Molina[DG]. They use a ripple edge $T_i \to T_j$ only when $d \in Write(T_i) \cap Read(T_j)$ and $\forall\ i<k<j\ \ d \notin Write(T_k)$, and similarly for interference edges. However, we can easily show that our change in the definition does not change the problem. In particular, the ripple closed sets are identical, and if $H \subseteq V$ is ripple closed, then $G \backslash H$ is acyclic in one definition iff it is acyclic in the other. Thus we are at liberty to use our notion of a precedence graph without any change in the problem.

## III. Main Results.

Davidson and Garcia-Molina give a proof of the following result, and Wright gives a reduction from the feedback vertex set problem.

**Theorem.** *The transaction backout problem is NP-complete.*

The general problem assumes no upper bound on the number of items in the database. Our main result is that a bound on the size of the database puts the problem in polynomial time. The first task in proving this result is to establish a bound on the length of a cycle in the precedence graph. It turns out that if the precedence graph has a cycle, then it has a cycle of bounded length. With no bound on the number of data items, the shortest cycle in the graph may be arbitrarily long. We are able to implement the dynamic programming in polynomial time because of this bound on the length of the cycles and the fact that there are a bounded number of transaction types. The careful reader will observe that we need our notion of a precedence graph for this lemma, and in fact, the lemma does not hold for Davidson and Garcia-Molina's definition.

**Lemma.** *Let $G$ be a precedence graph, and let $K=|D|$. If $G$ contains a cycle, then it contains a cycle of length at most $4K$.*

**Proof.** Suppose $A_1\ A_1 \to A_2 \to \cdots \to A_m \to A_1$ is the shortest cycle in $G$. Since two read-only transactions are not connected by an edge, at least $\frac{m}{2}$ of the transactions write some value. If

$m>4K$ then the cycle contains more than $2K$ writes, and some data item is written at least 3 times. So let $1 \le i<j<k \le m$ where $d \in Write(A_i) \cap Write(A_j) \cap Write(A_k)$. $A_i$, $A_j$, and $A_k$ must all be in the same partition, for otherwise $G$ contains a 2-cycle. Without loss of generality, assume $A_i$ precedes $A_k$ in their partition. Then $A_1 \rightarrow \cdots \rightarrow A_i \rightarrow A_k \rightarrow \cdots \rightarrow A_m \rightarrow A_1$ is a shorter cycle, contradicting the assumption that $m>4K$. Therefore $m \le 4K$. □

**Theorem.** *The transaction backout problem can be solved in $Time(2^{2^{p(K)}} n^2)$ for a polynomial $p$.*

**Proof.** Let $I_{tbp} = (D, S_1, S_2)$ be an instance of the transaction backout problem with $K = |D|$. Let $T_1, \ldots, T_n$ be the sequence of transactions in $S_1$ followed by $S_2$, where each transaction is labelled by its read and write sets, and the partition containing it. Let $S$ be the set of all sequences of transaction types of length at most $4K$. With $K$ items in the database, a transaction may ignore, read, or write a given item. So there are less than $2 \cdot 3^K$ distinct types of transactions, and $|S| \le (2 \cdot 3^K)^{4K} \le 2^{p(K)}$ for some polynomial $p$. We build the dynamic programming table as follows. For $1 \le i,j \le n$ and for $\alpha \subseteq S$ define:

$A_{i,l,\alpha}$ = **true** iff $\exists$ a subsequence, $\beta$, of $T_i, \ldots, T_n$ of length $l$ s.t. $<T_i, \ldots, T_n> \backslash \beta$ is ripple closed and $\alpha$ is the set of all sequences from $\beta$ of length at most $4K$.

The key to dynamic programming is that we can efficiently compute each entry in the table from earlier ones. In this case, we make crucial use of the facts that there are a bounded number of transaction types, and that a bounded number of transactions define a cycle in the precedence graph. For $\alpha_1, \alpha_2 \subseteq S$, define $\alpha_1 | \alpha_2$ as the collection of all subsequences of $s_1 s_2$ of length at most $4K$ where $s_i \in \alpha_i$. We can compute $A$ as:

$A_{i,l,\alpha}$ = **true** iff

(*i*) $A_{i+1,l,\alpha}$ = **true** and backing out $T_i$ does not violate ripple closure, or

(*ii*) $A_{i+1,l-1,\alpha_1}$ = **true** and $\alpha = <T_i> | \alpha_1$

We compute $A$ in the order $A_{n,0,\alpha}$, $A_{n,1,\alpha}$, $A_{n-1,0,\alpha}$, $A_{n-1,1,\alpha}$, $A_{n-1,2,\alpha}$, etc. Computing $A$ in backwards order makes checking ripple closure easier. Part (ii) is straightforward, and in part (i),

$T_i$ violates ripple closure if and only if there is a transaction type $T_j$ from $\alpha$ where $T_i \rightarrow T_j$ is a ripple edge.

The transaction backout problem has a solution of size $l$ if and only if $\exists\, \alpha \subseteq S$ and $\exists\, k \geq n-l$ such that $A_{1,k,\alpha}$ = **true** and $\alpha$ contains no cycle in the precedence graph. The lemma implies that it is sufficient to check all sequences of length $4K$ to show that the resulting set of transactions is indeed acyclic and hence serializable. Thus by computing the dynamic programming table, we have an algorithm for the transaction backout problem. As with other dynamic programming solutions, we can keep the sequence $\beta$ with the table $A_{i,l,\alpha}$ and compute which transactions to remove in the solution.

To analyze the running time of the algorithm, we first observe that $|S| \leq 2^{p(K)}$ so there are at most $2^{2^{p(K)}}$ choices for $\alpha$. Also $1 \leq i,j \leq n$, so there are only $2^{2^{p(K)}} n^2$ entries in the dynamic programming table. We see by the formula for computing the entries that each new entry can be computed in constant time, hence the algorithm runs in $Time(2^{2^{p(K)}} n^2)$ for some polynomial $p$. □

**Corollary.** *The transaction backout problem for a fixed-size database is in polynomial time.*

## IV. References.

[AHU] Aho, A.V., J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Reading, Mass., Addison-Wesley Publishing Co., 1974.

[DG] Davidson, S., and H. Garcia-Molina, "Protocols for Partitioned Distributed Database Systems," TR 280, Princeton University, Dept. of EECS, March 1981.

[W] Wright, D.D., "The Transaction Backout Problem," Cornell University, Dept. of Computer Science, July 1982.