On the Time Complexity of the Dining Philosophers Problem

Injong Rhee  
Department of Computer Science  
University of Warwick  
Coventry, CV4 7AL  
UK  
rhee@dcs.warwick.ac.uk

Jennifer L. Welch  
Department of Computer Science  
Texas A&M University  
College Station, TX 77843  
USA  
welch@cs.tamu.edu

Abstract

A tight bound on the response time for the dining philosophers problem in distributed systems is obtained. Also shown is that any (distributed) dining philosophers algorithm with the optimal response time is transformable in polynomial time into a sequential algorithm for an NP-complete problem. It suggests that an execution of any dining philosophers algorithm with the optimal response time may require a large (perhaps exponential) amount of local computation in acquiring resources.

Keywords: distributed systems theory, resource allocation, dining philosophers, lower bound, NP-completeness

1 Introduction

In operating systems, synchronizing simultaneous attempts to access system resources by concurrent processes without causing a deadlock or starvation is generally known as resource allocation. The response time, the time needed for a process to access all its required resources, is a typical efficiency metric for resource allocation. In this paper, we investigate the upper and lower bounds on the response time of a static version of the resource allocation problem, called the dining philosophers problem [6], in an asynchronous system where processes communicate by passing messages.

In the dining philosophers problem, each process requests an a priori known and fixed set of resources periodically, and has a code segment, called the critical region, where it uses the resources. A process enters the critical region after acquiring all its needed resources and then relinquishes them when it exits the region. If two processes require the same resource, we say that the two processes are in conflict. No two conflicting processes can be in the critical region at the same time. A conflicting graph can be constructed based on the conflict relation among processes, where a node represents a process and an edge represents the conflict relation between two processes.
For convenience in measuring the response time of an algorithm, it is assumed that a process is in conflict with at most \( \delta \) other processes (i.e., the maximum degree of the conflict graph is \( \delta \)); \( c \) is the maximum time that a process is in the critical region; and \( d \) is the maximum message delay between any two processes. These parameters are used only for analyzing time complexity, and are not used in specifying algorithms or proving the correctness of algorithms: we only consider asynchronous systems.

Despite much previous work on the upper bounds on the resource time for the dining philosophers problem \([6, 7, 4, 11, 5, 1, 10]\) (see [9] for review), little has been said about lower bounds on the response time. Only the obvious lower bound of \( \Omega(\delta(c + d)) \) is cited in the literature \([1, 5, 2, 8]\). This states that for every algorithm, there exists “one instance of the problem” on which the algorithm has an execution with a response time bigger than \( \Omega(\delta(c + d)) \).

In contrast, we show in this paper that for every algorithm and every instance of the problem, the algorithm has an execution with a response time bigger than \( \Omega(\chi \cdot (c + d)) \), where \( \chi \) is the chromatic number of the conflict graph for that instance. The previous bound is still correct, since \( \chi \leq \delta \) for every graph and there is a graph with \( \chi = \delta \). However, our bound is more informative and universally applicable in that it has something to say about every instance.

We also give a dining philosophers algorithm with the matching response time. The algorithm uses an optimal coloring algorithm as a subroutine, incurring a possibly exponential amount of local computation. However, we prove that such a large amount of local computation is inevitable (unless \( P = NP \)) by showing that any dining philosophers algorithm with the optimal response time is transformable in polynomial time into a sequential algorithm for an NP-complete problem.

A similar lower bound was presented by Barbosa and Gafni [3] on the maximum average number of times that a process runs in any given period, denoted by \( \gamma^*(G) \), in a synchronous system where conflicting processes need to run in an alternating order. They proved that the inverse of \( \gamma^*(G) \) is bounded in between the multi-chromatic number and the chromatic number of \( G \). The NP-completeness of obtaining the optimal solution with respect to \( \gamma^*(G) \) was also shown. Note that this result doesn't obviously imply our result. It might seem that obtaining an optimal response time would be easier than obtaining optimal \( \gamma^*(G) \). Nonetheless, we prove that it is not the case.

The rest of the paper is organized as follows. Sections 2 and 3 give the description of the formal model and problem definition. In Section 4, we show the lower bound, and in Section 5, we present the optimal algorithm and prove that the dining philosophers problem can be transformable to a sequential NP-complete problem in a polynomial time. We conclude in Section 6. Proofs of some key lemmas are only sketched because of the space limit, and are left to the full paper.

## 2 The system model

There exists a finite set of processes \( P = \{p_1, p_2, p_3, \ldots \} \). Processes communicate by passing messages. There are three types of process steps: send, receive and local steps. Send and receive are communication primitives and a local step changes local variables.
of processes. Each process \( p_i \) is modeled by a finite state automaton with state set \( Q_i \). The state set \( Q_i \) includes an initial state \( q_{0,i} \).

The automaton for each process is specified by a single guarded command set \([B_1 \rightarrow A_1 \blacklozenge B_2 \rightarrow A_2 \blacklozenge \ldots \blacklozenge B_m \rightarrow A_m]\). Each \( B_i \rightarrow A_i \) is a guarded command, where a guard \( B_i \) is a boolean expression or a message reception (receive step) or both. A finite list of action statements \( A_i \) that consist of multiple local steps or one send step or both.

We model communication by a special process, called network, which schedules the delivery of messages sent among the other processes. Each process owns a buffer of infinite size. A buffer is an atomic read-modify variable, so that a process can add a message to the contents of the buffer atomically. A send step adds a set of messages to the buffer of the network. A receive step of a process removes some messages from the buffer of the process. The receive step of the network removes some messages from its buffer, and the send step of the network adds the messages to the buffers of their destination processes. (In this case, we say that the messages are delivered to the destination.) A process can send the same message to a finite set of processes in one send step.

A configuration is a vector \( C = \{q_1, q_2, \ldots, \} \) where \( q_i \) is the local state of \( p_i \) for each \( p_i \in P \) (including the network). A guarded command is enabled in a configuration if its associated boolean expression is true and associated receive, if any, can return nonempty messages, i.e., the messages specified in the receive are in the buffer of its process. An execution of a guarded command involves an atomic execution of all the steps in the action statements of the guarded command. It results in simultaneous changes to the state of the process of the guarded command based on the state of the process, and possibly the state of the network if the guarded command involves a send step. A guarded command enabled in a configuration \( C \) can be applied to \( C \) to yield a new configuration \( C' \) as a result of the execution of the guarded command.

A system is specified by describing \( P \), an initial configuration \( C_0 = (\emptyset, 1, \emptyset, 2, \ldots) \), and the automaton of all processes in \( P \).

An execution sequence of a system is an infinite alternating sequence of configurations and enabled guarded commands \( \alpha = C_0, \pi_1, C_1, \pi_2, \ldots, C_i, \pi_i, \ldots \), where \( \pi_i \) is an enabled guarded command in \( C_{i-1} \); and \( C_i \) is obtained by applying \( \pi_{i-1} \) to \( C_{i-1} \). We say that in an execution sequence, a guarded command is continually enabled from \( C_i \) to \( C_j \), \( i < j \) if the guarded command is enabled in every configuration from \( C_i \) to \( C_j \) and is not applied to any configuration in between \( C_i \) and \( C_{j-1} \).

We also say that in an execution sequence \( \sigma \), a guarded command \( g \) is enabled before a guarded command \( g' \) in \( C_k \) if there exists a sequence of configurations \( C_i, \ldots, C_k \) in \( \sigma \) such that \( g \) is continually enabled from \( C_i \) to \( C_k \), but \( g' \) is not continually enabled from \( C_i \) to \( C_j \).

An execution is an execution sequence satisfying the following fairness conditions: (1) if \( \pi_i \) is a guarded command of process \( p \) (that is not the network) and applied to \( C_i \), then there is no guarded command of \( p \) that is enabled before \( \pi_i \) in \( C_i \); (2) all enabled guarded command will be executed eventually; and (3) messages are received in the order that they are sent (i.e., FIFO).

A timed execution \((\alpha, T) = C_0, (\pi_1, t_1), \ldots, (\pi_j, t_j), \ldots\) satisfies the following: (1) \( \alpha =
$C_0, \pi_1, C_1, \pi_2, \ldots C_i, \ldots$ is an execution; (2) $T$ is a mapping from guarded commands to real numbers that associates a real time with each guarded command in the execution. The sequence $t_0, t_1, \ldots t_i$ is nondecreasing and unbounded; and (3) all messages sent are received in finite time.

If $V$ is a state variable of a process and $t$ is a real number, $V(t)$ denotes the value of $V$ in the configuration $C_j$ where $T(\pi_j) \leq t \leq T(\pi_{j+1})$, i.e., a configuration $C_j$ represents the states of the system during time interval $[T(\pi_j), T(\pi_{j+1})]$.

Note that the system model here is completely asynchronous because there are no constraints on relative timing of process steps and message delays.

3 The dining philosophers problem

We now specialize the general system for resource allocation problem. Since we don’t put any restriction on the behavior of resources and we treat them as passive entity\(^1\), we don’t model resources as physical objects in our system. We simply treat them as an additional system parameters. Let $R$ be the set of all resource IDs in the system. There exists a set of processes $U$ ($\subseteq P$) called users that need a subset of $R$ for their execution at various times. Let $R_i$ be a subset of $R$ and denote the resource requirement of a user $i$. If $R_i \cap R_j \neq \emptyset$ for users $i$ and $j$, then we say that user $i$ conflicts with user $j$.

The states of each user are partitioned into four regions. In the trying region, the user requests its required resources. On acquiring the resources, the user enters the critical region. It remains in the region for finite time using the resources. When the user is through with the resources, it enters the exit region, where it relinquishes the resources. Otherwise, the user is in the remainder region. Initially, every user is in the remainder region. To specify the above, we assume that each user has a local variable called region whose value is set to constant Trying, Critical, Exit, or Remainder if and only if the user is in the trying region, the critical region, the exit region or the remainder region respectively.

A user code also contains the statements that are not related to resource allocation such as the statements that are executed in the critical region. A resource allocation algorithm is specified by only describing the statements that are related to resource allocation, so that it can be used as a subroutine for any users.

To formalize this concept, we define a mapping $M_i$ for a user $i$ that labels each guarded command in the code of $i$ to be either resource allocation part or non-resource allocation part. We call the set of process steps labeled resource allocation the resource allocation part of $i$, and call the set of process steps labeled non-resource allocation the non-resource allocation part of $i$.

A user code is well-formed if there exists $M_i$ for any $i$ in $U$ such that (1) all the steps in the non-resource allocation part are executed only when user $i$ is in the critical or remainder region; (2) region is the only variable that is used (i.e., read or written) both in the resource allocation part and in the non-resource allocation part; (3) in the resource allocation part, region is updated only either from Trying to Critical or from

\(^1\)The exclusion property of a resource can also be modeled by restricting the behavior of processes.
Exit to Remainder; and (4) in the non-resource allocation part, region is updated only either from Critical to Exit or from Remainder to Trying.

The dining philosophers problem is a system with a finite and fixed set $U$. The algorithm of each user is well-formed, and satisfies the following two conditions even when the non-resource allocation part of user $i$ (labeled by $M_i$) is replaced by any set of guarded commands: (1) (exclusion) in any timed execution of the algorithm, if users $i$ and $j$ are both in their critical regions at the same time, then $i$ and $j$ do not conflict with each other; this is the property that guarantees the exclusive access to the resources, and (2) (no-lockout) in any timed execution of the algorithm, if a user is in the trying or exit region, then it leaves its current region in finite time assuming no user remains in the critical region forever.

The response time of an algorithm is defined to be the maximum time period between when a user enters the trying region and when the user enters the critical region subsequently, for all timed executions of the algorithm.

## 4 Lower Bound on the response time

Before we prove the lower bound, we first define a few terms. We define a coloring assignment $C$ of an arbitrary graph $G = (V, E)$ to be a function $C : V \rightarrow \mathbb{N}^2$ such that if $(v_i, v_j) \in E$, $C(v_i) \neq C(v_j)$. The chromatic number of a graph $G$ is the minimum number of colors used for any coloring assignments of $G$. Finding a coloring assignment with the minimum number of colors for a given graph $G$ is called the coloring problem.

The lower bound studied here is based on the conflict graph $G$, which is given as input to the dining philosophers problem. Based on all resource requirements of users in dining philosophers problem, the conflict graph $G$ is constructed. Let $\chi$ be the chromatic number of $G$. In this section, we present the lower bound $\Omega(\chi \cdot (c + d))$ on the response time for the given $G$.

The structure of the lower bound proof is as follows. Given a dining philosophers algorithm $A$, we define a specific execution of the algorithm in which all users enter the trying region at the start of the algorithm; every user stays in the critical region exactly $c$ time; and once a user return to its remainder region, it stays there forever.

In this execution, we first prove that there exists a sequence of users $u_1, u_2, \ldots, u_\chi$ where for any $i$, $u_i$ conflicts with $u_{i+1}$ and $u_i$ enters the critical region before $u_{i+1}$ (see Lemma 4.1). Then, we prove that at least one communication delay (which is at most $d$ time) elapses between when $u_i$ leaves the critical region and when $u_{i+1}$ enters it (see Lemma 4.2). Thus, the worst case time period that $u_\chi$ is in the trying region is $(\chi - 1)(c + d)$. This proves the lower bound. Now we provide the details.

Let $(\beta, T)$, called the synchronized execution, be the timed execution of $A$ where all users enter the trying region at time $0$; after a user enters the remainder region, it remains in the region (i.e., no more requests for resources); all message delays are exactly $d$ and all critical region times are exactly $c$ where $d$ and $c$ are positive integers bigger than 1 and for every steps $g$ in $\beta$, $T(g)$ is an integer.

---

$\mathbb{N}$ is the set of natural numbers.
Lemma 4.1 In \((\beta, T)\), there is a sequence \(u_1, u_2, \ldots, u_k\) where \(k \geq \chi\), such that for all \(j, 1 \leq j \leq k\), user \(u_j\) leaves the critical region before user \(u_{j+1}\) enters it and user \(u_j\) conflicts with user \(u_{j+1}\).

Proof: (Sketch) By way of contradiction, given that there is no such sequence \(u_1, \ldots, u_k\) with \(k \geq \chi\) in \(\beta\), it can be proved that there exists a coloring assignment of \(G\) with less than \(\chi\) colors. This is a contradiction.

Lemma 4.2 In \((\beta, T)\), if user \(i\) enters the exit region at time \(t\) and conflicting user \(j\) hasn't entered the critical region before time \(t\), then \(j\) enters the critical region only after \(t + d\).

Proof: (Sketch) By way of contradiction, we assume that \(j\) enters the critical region at time \(t + x\) where \(x < d\). Then, we prove that there exists an execution \((\beta', T')\) where users \(i\) and \(j\) are in the critical region together. This contradicts the exclusion condition of \(A\). We obtain execution \((\beta', T')\) by retiming and reordering \((\beta, T)\) in such a way that all the steps of \(j\) executed during time interval \((\beta - 1, t + x]\) in \((\beta, T)\) happen during time interval \((\beta - 1, t]\) in \((\beta', T')\). We also retime all the actions of the network that deliver messages to \(j\) during time interval \((\beta - 1, t + x]\) of \((\beta, T)\) in the same way.

Fix \(\epsilon > 0\). For all guarded commands \(g\) in \(\beta\),

\[
T'(g) = t - 1 + \frac{1}{x + \epsilon} (T(g) - (t - 1)) \quad \text{if } T(g) \text{ is in } (t - 1, t + x] \text{ and } g \text{ is either a guarded command of } j \text{ or a guarded command of the network that adds messages to } j\'\text{s buffer.}
\]

\[
T'(g) = T(g) \quad \text{otherwise.}
\]

We reorder the guarded commands in \(\beta\) according to the new time assignment \(T'\). Using \(C_0\) and the reordered sequence of guarded commands, we construct a new execution sequence \(\beta'\). We prove that \(\beta'\) is an execution (whose proof is omitted). Note that this retiming moves the guarded command \(g\) of \(j\) that enters the critical region to occur at some time in time interval \((\beta - 1, t]\) because \(T(g) = t + x\) and \(T'(g) = t - 1 + \frac{(\chi + 1)}{x + \epsilon}\). This means that user \(j\) enters the critical region in the interval \((\beta - 1, t]\). Therefore, since user \(i\) is in the critical region at time \((\beta - 1, t]\), users \(i\) and \(j\) are in the critical region at the same time (see Figure 1). This is a contradiction because \(i\) and \(j\) are in conflict.

Corollary 4.3 The worst case response time of any algorithm for the dining philosophers problem is \((\chi - 1) \cdot (c + d)\).

Proof: Lemma 4.1 states that in \(\beta\) there is a sequence of \(\chi\) users that enter the critical region consecutively. Recall that all users enter the trying region at time 0 by definition of \(\beta\). Since each user spends \(c\) time in the critical region and at least \(d\) time elapses between when one user leaves the critical region and the next conflicting user enters it, the result follows.
Figure 1: An example of the retiming. (a) \((\beta, T)\), (b) \((\beta', T')\). An arrow represents message delivery and delay; a tick mark represents an execution of a guarded command and time increases from left to right. In \((\beta', T')\), \(\tau\) occurs before \(\sigma\) which means users \(i\) and \(j\) are in the critical region at the same time.

5 An optimal algorithm and NP-Hardness

The following theorem shows that \((\chi - 1) \cdot (c + d)\) is the optimal response time for the dining philosophers problem.

**Theorem 5.1** There exists a dining philosophers algorithm whose response time is \((\chi - 1) \cdot (c + d)\).

We can prove theorem 5.1 by providing a simple algorithm. We only give an informal description of the algorithm.

**Algorithm:** Given a conflict graph of a dining philosophers problem, we can color the conflict graph \(G\) with \(\chi\) colors by using any known coloring algorithm. Therefore, each user knows the colors of all users. The basic idea of the algorithm is to use token-passing. Tokens are passed among users with different colors in such a way that if a user with color \(i\) holds a token, to pass the token to other users, it sends a message to all users with color \((i + 1 \mod \chi)\). If a user with color \((i + 1 \mod \chi)\) receives a message from each user with color \(i \mod \chi\), it now has a token. Initially, all users with color 1 have a token. If a user is in the trying region while holding a token, it enters the critical region and when it enters the exit region, it passes the token. If not, it simply passes the token.

It is easy to see that the response time of the described algorithm is \((\chi - 1) \cdot (c + d)\) because the users with color \(\chi\) may have to wait for \(\chi - 1\) users to enter the critical region and pass the token in series.

Obviously, this algorithm is very expensive because it uses an optimal coloring algorithm and all the known optimal coloring algorithms require an exponential amount of (local) computation. However, we prove that this cost is inevitable (unless P=NP) by showing that any dining philosophers algorithm is transformable in polynomial time into a sequential algorithm for an NP-complete problem.

**Theorem 5.2** Any algorithm for the dining philosophers problem with response time less than or equal to \((\chi - 1)(c + d)\) is transformable in polynomial time into a sequential algorithm for an NP-complete problem.
To prove the theorem, we first define the relevant NP-complete problem. Then we show how to transform a distributed dining philosophers algorithm with response time \((\chi - 1)(c + d)\) into a sequential algorithm for the NP-complete problem.

**Definition 5.1** The optimal resource scheduling problem is, given a process set \(P = \{p_1, p_2, \ldots, p_n\}\), a resource set \(R = \{r_1, r_2, \ldots\}\) and a resource requirement set \(\text{Req} = \{R_1, R_2, \ldots, R_n\}\), \(R_i \subseteq R\), find a function \(S : P \rightarrow N\) such that if \(R_i \cap R_j\), \(S(p_i) \neq S(p_j)\) and \(\max_{i} \{S(p_i)\}\) is as small as possible.

We call the mapping \(S\) a schedule, and \(\max_{i} \{S(p_i)\}\) the response time of \(S\).

**Lemma 5.3** The optimal resource scheduling problem is NP-complete.

**Lemma 5.4** Any (distributed) dining philosopher algorithm with response time \((\chi - 1)(c + d)\) can be transformable into a (sequential) optimal resource scheduling algorithm in a polynomial time.

**Proof:** Let \(A\) be a dining philosophers algorithm with the optimal response time. First, since \(A\) is a dining philosophers algorithm, the user code of \(A\) must be well-formed and there exists a mapping \(M_i\) for each \(i\) that labels each step either with resource allocation or with non-resource allocation satisfying the conditions specified in Section 3. Let \(E\) be the set of action statements executed guarded by \(\text{region} = \text{Exit}\) (if there is any).

We first remove all the non-resource allocation part from \(A\) and all the guarded commands associated with \(E\), and replace the remainder and critical regions by the following code.

**Remainder region:**

- \(\text{region} = \text{Remainder}, \neg \text{done} \rightarrow \text{region} = \text{Trying};\)
- \(\text{region} = \text{Remainder}, \text{done} \rightarrow \text{done} = \text{true};\)

\(\text{done}\) is a boolean variable and initially set to false, and \(\text{user}_id\) is the ID of the user.

The new algorithm is still a dining philosophers algorithm with the optimal response time because any dining philosophers algorithm should work correctly for any given remainder and critical regions, and the resulting code is still well-formed.

The code states that all users enter the trying region as soon as they are in the remainder region (at time 0 because users is in the remainder region initially); in the critical region, they print out their IDs and then executes the exit region (note this is done in one step because of the atomic execution of the guarded command); and once entering the remainder region, users remain there forever.

Then, we simulate the new dining philosophers algorithm in a sequential machine as follows. The local variables, buffer and state of each process are maintained in a separate memory space. Each process in \(P\) is executed in round robin order, i.e., at each round, one enabled guarded command of each process, if there is any, is executed
based on the local state of the process which is maintained in its memory space, while observing the fairness conditions given in Section 2. At the end of each round, the network checks its buffer and delivers all the messages in the buffer to the destination process’s buffer. This ensures that messages are delivered in the same round they are sent. It is easy to see that this transformation adds only polynomial time overhead (in fact, a factor of $n$ if $n$ is the total number of processes in $P$).

Let $A_{seq}$ be this sequential version of the new algorithm. We can now use $A_{seq}$ as a subroutine to solve the optimal resource scheduling problem. We feed $Req$, which is given for the optimal resource scheduling problem, as input to $A_{seq}$ (i.e., let $R_i' = R_i$) and run $A_{seq}$. $A_{seq}$ prints the IDs of the users that enter the critical region at each round (see the code of the critical region). All users must enter the critical region once by round $\chi$ because users finishes in one round, messages are delivered in the same round they are sent, and the original dining philosophers algorithm has the response time of $((\chi - 1)(c + d)$. To come up with an optimal schedule $S$, for each user $i$, let $S(i)$ be the round that the user enters the critical region. It is easy to see that $\max_{\chi}(S_{\chi}) = \chi$ and it is the minimum because the dining philosophers algorithm has worst case response time $(\chi - 1)(c + d)$.

Theorem 5.2 implies that any dining philosophers algorithm with the optimal response time must have a large (possibly exponential) amount of local steps. This justifies the use of an optimal coloring algorithm in our optimal dining philosophers algorithm.

6 Conclusion

In this paper, we presented a tight bound of $(\chi - 1)(c + d)$ on the response time for the dining philosophers problem. Although our optimal dining philosophers algorithm requires a large amount of local computation, we showed that the cost is inevitable (unless $P=NP$) by proving that the dining philosophers problem can be transformable to an NP-complete problem in a polynomial time. However, we note that the algorithm is not sensitive to contention, i.e., the response time of the algorithm is always at most $(\chi - 1)d$, even if only one process is contending. Finding contention-sensitive distributed dining philosophers algorithm with the optimal response time is an open problem.

References


