# On Distributed Resource Handling: Dining, Drinking and Mobile Philosophers

Marina Papatriantafilou* Philippas Tsigas*†

November 1997

**Abstract**

Resource allocation is an essential class of problems, which includes many fundamental concurrency control problems. A significant amount of research has been devoted to the group of these problems, which share a common part in their name: *philosophers* (dining, drinking, mobile,...) We survey some of the main developments in this field.

## 1 Introduction

Taking into account that time, space, devices and control are all resources shared among the entities of a distributed system, the resource allocation problem class includes most of the fundamental concurrency control problems. Resources can be concrete or abstract entities, such as critical sections, printers, files, CPUs, critical sections, bandwidth, they can be reusable or not (e.g. seats and tickets in airline reservation systems, respectively), they can be identical or distinct. Depending on the type of the resources and combinations of them (e.g. devices *and* time, in the sense of real-time requirements), we have different instances (with variations of definitions and names) of the problem: e.g. mutual exclusion, $k$-exclusion [15] scheduling, processor allocation, load balancing [6, 9], real-time scheduling [33], non-reusable-resource allocation [28], banker-like resource allocation [21], are some examples.

Many of the types and aspects of the problem are well-studied and the existing literature on them is extensive. Moreover, the research interest in the field remains high, both because of its breadth, as well as because of the demand in the corresponding applications areas. In this short paper we can only cover a few of the many interesting results of the field. In particular, we survey some of the main developments in the literature on the dining, drinking and mobile types of philosophers formulation of the problem of reusable-resource allocation.

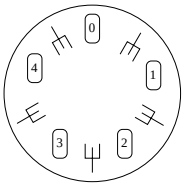
*Computing Sciences Department, Chalmers University of Technology and Göteborg University, S-412 96 Göteborg, Email: (ptrianta, tsigas)@cs.chalmers.se

†Department of Computer Systems, University of Uppsala, Sweden

# 2 The problems

The core of the problem is that processes ask for reusable) resources (maybe several of them concurrently, for bounded time use); each resource can be used by only one process at a time.

The problem of *dining philosophers* was described by Dijkstra [13]:

> "... the problem is the Five Dining Philosophers. The life of a philosopher consists of an alternation of thinking and eating:
>
> *cycle begin* think ;
> eat
> *end.*
>
> Five philosophers, numbered from 0 through 4 are living in a house where the table is laid for them, each philosopher having his own place at the table:
>
> 
> 
>
> Their only problem —besides those of philosophy— is that the dish served is a very difficult kind of spaghetti, that has to be eaten with two forks. There are two forks next to each plate, so that presents no difficulty: as a consequence, however, no two neighbours may be eating simultaneously."

In the *generalised dining philosophers* problem [26], each philosopher has an arbitrary number of neighbours, with each one of whom he shares one fork and he needs all of these forks in order to eat.

The *drinking philosophers* problem, introduced by Chandy and Misra in [11] is a generalisation of the dining philosophers. Each philosopher has, as before, an arbitrary number of neighbours with each one of whom he shares one *bottle*; each time that it becomes thirsty, it requires a (pre-specified) subset (which may differ each time) of its incident bottles in order to drink.

The integration of *mobility* introduces new issues and problems [7, 23]. Now the philosophers do not just sit, but are able to move. The ability of a philosopher (also called *mobile host* in this setting) to be reachable while moving requires some wireless form of communication, which supports broadcast communication within a specific region, called a *cell*. To provide communication between different cells, a specific host, called *cell agent* or *base station* is associated with each cell. The set of cell agents are interconnected and can communicate with each other in pairs. In order to satisfy a communication request of a mobile host, the respective cell agent should allocate, depending on the bandwidth required, a certain *number* of wireless channels[1] to it [22]. The wireless channels should be chosen by the cell agents in such a way that no interference between signals can occur. A solution to the mobile channel allocation problem should guarantee that if a channel is used for a communication session of a mobile host (philosopher) in a specific cell, it should

[1] Wireless channels are divisions of the frequency spectrum provided for broadcast communication; the signal carried on a wireless channel is carried on the respective frequency. Henceforth, the terms "channel" and "frequency" are used interchangeably.

not be used concurrently for other communication sessions, except in *distant cells*, outside the scope of the signal. Note that the set of frequencies that are potentially possible to be used in each cell are the same (i.e. frequencies are re-used in each cell). However, the *free frequencies* at a cell agent are the frequencies of the spectrum that are not in use in that cell or in any of the cells within the scope of the signal, and it is only from this set of frequencies channels that may be allocated to satisfy the requests at this cell. (Requests that cannot be satisfied can be dropped.) Note that the set of free frequencies for a base station changes over time. Furthermore, the same frequency could potentially belong to the set of free frequencies of two adjacent base stations and hence, even when a base station is picking channels from its own set, it needs to ensure that there is no interference.

To solve the philosophers problems, an algorithm must ensure the following:

**Exclusion** (safety ): no resource should be used simultaneously by more than one processes. The *important distinctions* between these three types of problems are with respect to exclusion and can be summarised as follows: The *dining* problem *disallows* neighbours to eat simultaneously. The *drinking* problem *allows* neighbours to drink simultaneously, provided that they *need* to drink from *different* (pre-specified) bottles. The *mobile* problem *allows* neighbours the possibility to *choose different* frequencies to use, so that they make it possible to operate simultaneously (i.e., situations in which adjacent base stations decide to pick the same frequency so that one of them has to wait for the other to release the frequency (thereby resulting in response times that depend on the duration that the frequencies are in use) can (and should) be avoided).

**No deadlock** (progress): among a set of non-failed processes which need to use the resources, some of them should be able to do so after bounded time.

**Starvation Freedom** (fairness): as long as a process does not fail, the response time for its requests is bounded.

The system is described with an *interference graph* (or *conflict graph*) $G = (V, E)$, where nodes represent processes (philosophers, or, in the mobile setting cell agents) and edges represent resource conflicts: an edge between two nodes implies conflict in case these processes try to access the respective resource(s) simultaneously. If a pair of philosophers in the drinking and dining setting have (potential) conflicts in more than one resources, the interference graph has multiple edges between them, one for each resource (for reasons that will be obvious in the next section). The maximum degree in the graph is denoted by $\Delta$. In the drinking philosophers, each process $v_i$ has an associated set of variables that specifies each time the bottles that it needs in order to drink. In the mobile philosophers, each cell agent $v_i$ has an associated number $r_i$, which denotes the number of pending requests at that cell each time.

Each process $v_i$ $(1 \leq i \leq n)$ is able to communicate (with messages or shared memory) with its neighbours in $G$. Usually no assumption is made on the speeds of the processes or on the communication delays.

The metrics of interest for the evaluation of solutions are (we are interested in the *worst-case*):

**Response time**: this is the delay between the time that a process develops the need to use the resources and the time it is able to do so. It is measured in

- units of the upper bound of the time that a process needs to use the resources (notice that in the mobile problem it is undesirable to relate the response time to this measure), and/or,
- depending on the underlying communication medium, in units of the maximum message transmission delay in the network or in units of the duration of access to each shared variable.

It should be stressed that these bounds are not assumed for the sake of the correctness of the solutions, nor for the sake of being used by the solutions, but only for the sake measuring the response times.

**Communication complexity**: again, depending on the underlying communication medium, this is measured in number of accesses to shared variables, or in number of messages exchanged.

**Fault Tolerance**: The concept of *failure locality* has been introduced [12] to measure the effect of process *stopping failures*. An algorithm has failure locality $m$ if a process is free from starvation even if some process outside its $m$-neighbourhood has failed by stopping. (The *$m$-neighbourhood* of a process is the set of processes within distance $m$ from it.)

**Request Satisfiability** (relevant to the mobile philosophers problem): Because the number of resources to be used is limited and the solution *selects* the resources to be used, here it is important that a solution also aim at maximizing the number of satisfiable requests; clearly this implies that the solution should adapt fast to temporal variations in channel demand in different cells.

# 3 No symmetric dining solution

First of all, it should be mentioned that there can be no (deterministic) solution [27] to the dining philosophers problem if the system is completely symmetric, i.e. if processes, initial local states, local variables and resources are identical. The idea is that in such a system, a continuous round-robin execution of one step by each process, will always result again in symmetric states, which implies that if one philosopher is eating, then all of them will be eating at the same time, hence the exclusion property will be violated.

Rabin and Lehmann in [32] gave a probabilistic solution to the dining philosophers problem, in which symmetry is broken with probabilistic choices. Each process' trying (to get its incident forks) section consists of iterations. In each iteration, it chooses a first fork randomly and waits as long as necessary to obtain it. After that, it just checks once to see if the second fork is available, in which case it picks it and proceeds to eat. If it is not available, it gives up on this iteration, puts the first fork down and proceeds to the next iteration.

**Theorem 1 (Rabin and Lehmann 1981 [32])** *The* Lehmann and Rabin *algorithm [32] is a solution for the dining philosophers problem; it guarantees exclusion and starvation freedom; it also guarantees progress with probability 1.*

# 4 Dining by collecting forks one by one

The basic line of this approach is that a hungry philosopher, in order to eat, has to compete to get the forks that it needs one by one, by executing for each one a mutual exclusion protocol with the other processes that may ask for that fork. Depending on the type of communication provided by the underlying system, this can be made using any mutual exclusion network protocol (e.g. the one by Ricard and Agrawala [17]), or semaphores, or *ReadModifyWrite* variables, or a resource controller for each resource (the controller can be centralised [17], or distributed [1]). In this way, exclusion is guaranteed by the underlying mutual exclusion protocol used. To guarantee absence of deadlock, the order in which resources (forks) are asked for is important.

Hence the approach dictates some ordering of the resources in some way. This can be done by random choices, as in the solution by Rabin and Lehmann [32], or deterministically, in a way that defines a partial order among the resources, such that the resources that can be asked by each process are totally ordered. The latter can be achieved by edge colouring the conflict graph (or, as in the approach taken by Lynch in [26], by vertex colouring the *resource graph*, in which the nodes are the resources in the system and pairs of nodes are connected by an edge if there is at least one processes that may ask for both them).

So, the *asymmetry* in this approach is based on *colouring the resources.* Assume $C$ colours are used. (Arbitrary graphs can be coloured in a distributed manner with $\Delta + 1$ colours [5, 27], while for planar ones 6 is sufficient [18].)

## 4.1 LeftRight algorithm and its generalisation

In a direct application of this approach, discussed and analysed by Lynch in [26, 27] (algorithm *Colouring*), all that each hungry process needs to do is to pick up its forks (resources) in an increasing colour order. In the original problem where the philosophers sit at a round table, this is the *LeftRight* solution [27]: if a philosopher is sitting on a seat with even number it picks up first its left and then its right fork, while if it is sitting on a seat with odd number it picks up first its right and then its left fork.

Note that waiting for resources introduces *waiting chains* among the processes. A process $v_i$ waiting for resource with colour 1 is essentially waiting for another process $v_j$ that holds that resource; $v_j$ may in turn be waiting for a resource with colour 2, and in this way, there may be waiting chains of processes of length equal to the number of colours, $C$. The maximum length of waiting chains is the maximum distance (in the interference graph) between two processes, such that one process can delay the other[2]. Note that the process in the high-colour end of the chain will be able to eat and this guarantees progress (i.e. deadlock avoidance) in the system. This (together with the respective liveness properties of the underlying mutual exclusion algorithms used) also guarantees liveness in the dining philosophers solution.

The analysis of the response time of the algorithm, which can be found in [26, 27], shows that its response time is dependent only on local measures of the network (the maximum *contention* $m$ at a resource, the number of colours and not on global

[2]Recall the definition of failure locality and note that the maximum length of the process waiting chains specifies also the failure locality of the algorithm.

measures (i.e. it is independent of the total number of processes and resources in the system). However, the response time is exponential in these measures.

**Theorem 2 (Lynch 1981 [26, 27])** *The* Colouring *algorithm [26, 27] is a solution for the dining philosophers problem; its response time is* $O(Cm^C)$*, its communication complexity is* $O(C)$ *times the communication complexity of the mutual exclusion algorithm used and its failure locality is* $C$*.*

## 4.2 Restricting the waiting chain length

Styer and Peterson in [34] came up with interesting ideas to reduce the length of maximum process waiting chains and hence, improve both the response time and the failure locality. They first propose a dynamic algorithm, which attempts to eliminate waiting chains by having each process, when it sees that a resource is unavailable, release all currently held resources and start competing for the first one again. Such a solution has maximum waiting chain length (and hence, failure locality) only 1, but may easily result in starvation. They balanced this behaviour by having the process release (in decreasing colour order) only at most half of the resources it has picked, if it encounters a reserved resource (and start competing from that "back-tracked" point. This ensures maximum waiting chain length at most $\log C$. To avoid starvation behaviour due to "back-tracking" and re-entering the competition from an earlier point, Styer and Peterson introduced some synchronisation mechanism, which they call *fence protocol*. This can be thought of as an *asynchronous synchronization barrier* (see Fig. 2), equivalent to each philosopher asking for permission to cross, and waiting until receiving one from each one of its neighbours. Provided that a philosopher that has already crossed it defers giving permission to its neighbours until it finishes eating, this rptocol guarantees that each philosopher will have to wait for each of its neighbours at most once before eating.

**Theorem 3 (Styer and Peterson 1988 [34])** *The* Styer and Peterson *algorithm [34] is a solution for the dining philosophers problem; its response time and communication complexity are* $O(\Delta^{\log C})$ *and its failure locality is* $\log C$*.*

# 5 Dining (drinking) by collecting forks (resp. bottles) concurrently

Here the processes do not ask for the resources one by one, but they request all of them concurrently. The approach implies some symmetry braking among the *processes* in order to resolve conflicts (as opposed to symmetry breaking among the *resources* in the previous section).

## 5.1 Approach based on dynamic conflict resolution

This is due to Chandy and Misra [11]. Consider the dining philosophers problem and assume an initial acyclic orientation of the conflict graph. (An acyclic orientation can be obtained by e.g. first executing a distributed vertex colouring protocol and then orienting each edge from the higher to the lower colour. Arbitrary graphs can be coloured in a distributed manner with $\Delta + 1$ colours [5], while for planar ones 6

is sufficient [18].) The orientation of an edge represents a priority relation between its endpoint-processes, pointing to the one with higher priority. This can be used to resolve conflicts between them. A node which is a sink in this orientation can have all the forks that it needs and hence can start eating. After finishing eating, it reverses the direction of all of its incident edges, thus giving priority (resp. the forks) to its neighbours. By reversing all incident edges concurrently, the acyclicity of the orientation is preserved (there will be new sinks, who will eat next and repeat the same procedure) and hence, deadlock is avoided. Clearly, exclusion is guaranteed, since no neighbouring processes can eat simultaneously. Also, the change of the direction of a relevant edge after each eating session of its endpoint-processes implies that they can only take fair turns in eating, which guarantees starvation freedom.

This procedure, except from solving the dining philosophers problem, provides a mechanism to maintain a *dymanic acyclic conflict resolution* scheme, which can be used to solve other synchronisation and concurrency control problems. Chandy and Misra demonstrated its use in solving the drinking philosophers problem: each time two thirsty neighbouring philosophers want to drink from the same bottle, they use the above conflict resolution scheme to break the symmetry. The final algorithm consists of tho algorithms, executed concurrently, one that manages forks (forks are used as auxiliary resources to resolve contention for the bottles) and one that manages bottles. A *thirsty* philosopher also becomes *hungry*, not for excluding its neighbours from drinking simultaneously (because they are allowed to if they do not need common bottles), but to get the forks (i.e. direct its incident edges towards it), in order to resolve conflicts with them in case such conflicts arise.

Because of its dynamic nature, the protocol avoids the exponential behaviour of [26], but on the other hand, the maximum waiting chain length can be of size $n$ and this can be seen as follows: The chains of the initial orientation get modified by becoming longer at their "tails" and shorter at their "heads" due to the edge-reversal mechanism. In an asynchronous system, due to differences in the speed of the processes and the communication lines, these modifications may result in waiting chains of arbitrary length.

The algorithm is very well presented in detail in [11]. Welch and Lynch gave a modular presentation of the above algorithm in [37]. Murphy and Shankar have also presented a quick rephrasement and some nice enhancements in [30].

**Theorem 4 (Chandy and Misra 1984 [11])** *The* Chandy and Misra *algorithm [11] is a solution for the dining and drinking philosophers problem; its response time is* $O(n)$*, its communication complexity is* $O(\Delta)$ *and its failure locality is* $n$*.*

Analysis of the concurrency attainable by such a scheme has been given by Barbosa and Gafni in [8]. A good source of reading more on acyclic orientations, labelling and efficient ways to get them is a paper by Attiya et.al. [3]. The drinking philosophers has also been solved by Ginat et. al. in [19], with the use of times-tamping [25] (thus avoiding to run the "dining layer" of Chandy and Misra) with the same worst case response time and communication complexity between zero and twice the number of bottles needed for drinking.

## 5.2 Approach based on distributed queues

The first solution in this approach is by Awerbuch and Saks in [6]. They consider a dynamic system, in which each process may request any resource in each session. In

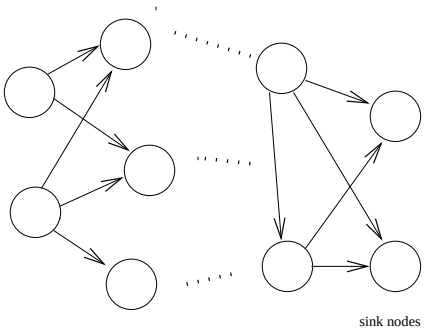



Figure 1: Solution using orientation

this dynamic version, the dining and drinking philosophers problems are then the same. The idea is to maintain a distributed queue that consists of a totally ordered set of positions, each of which can hold one or more *jobs* (the term "job" is used in the paper by Awerbuch and Saks [6] and the problem is treated as a *dynamic job scheduling problem*; each job is a collective request by a philosopher for all the resources it needs). New jobs are placed in the queue and progress through it in a way that ensures that no two conflicting jobs can reach the head of the queue at the same time. The algorithm in [6] is first described under the assumption that requests come sufficiently separated in time, in a way such that two or more jobs cannot enter the queue simultaneously. Then it is sketched how to cope with the opposite with the use of randomness and last it is shown that randomness can be removed, showing the following:

**Theorem 5 (Awerbuch and Saks 1990 [6])** *The* Awerbuch and Saks *algorithm [6] is a solution to the (dynamic) dining/drinking philosophers problem; the worst case response time and the communication complexity are* $O(\Delta^2 \log U$ *(where* $U$ *is the range of the set from which the identities of the processes are selected); its failure locality is* $O(\Delta)$.

A study of this technique under parametrisation of the process computation power and the knowledge of the job execution times has been given by Bar-Ilan and Peleg in [9]. The results are given for the synchronous model of computation and communication and can be transformed to the asynchronous one modulo the overhead of a global synchroniser [4], which also implies failure locality $n$.

## 5.3 Approach based on synchronisation barriers

This approach is based mainly on the idea of synchronisation barriers, such as the fence protocol, introduced by Styer and Peterson in [34]. Choy and Singh extended it and introduced an efficient protocol for the dining philosophers in [12]. The protocol assumes a priority (conflict resolution) scheme, which can be a static acyclic orientation or a node colouring. Actually, as also mentioned before in this paper, an acyclic orientation can be obtained by first executing a distributed vertex colouring protocol and then orienting each edge from the higher to the lower colour (arbitrary graphs can be coloured in a distributed manner with $\Delta + 1$ colours [5], while for

planar ones 6 is sufficient [18]). So assume an initial vertex colouring of the graph that will serve as a priority scheme. Assume $C$ colours are used. The core of the idea is that between neighbouring philosophers that are hungry at the same time, the one with smaller colour should have priority. This is as in the Chandy and Misra [11] idea, but in this case it is static. As such, it might lead to starvation of a philosopher with high colour, due to preemption by its lower coloured neighbours. To avoid this, some synchronisation (double doorway protocol) is used (see Fig. 2).

1. The first (asynchronous) doorway is very much like the fence protocol of Styer and Peterson [34], which was introduced to avoid starvation. It is equivalent to each philosopher asking for permission to cross, and waiting until receiving one (and once) from *each* one of its neighbours. Provided that a philosopher that has already crossed the doorway defers giving permission to its neighbours until it finishes eating, this doorway guarantees that each philosopher will have to wait for each of its neighbours at most once before eating. On the other hand, because of its asynchronous nature, "staggered" crossing of the doorway can preempt the fork acquisition process of a philosopher multiple times per trying session, and can result in some exponential response time [12, 34].
2. The second (synchronous) doorway is equivalent to the philosopher waiting until it finds an instant when none of its neighbours is past this second doorway. This doorway alone guarantees that after the time instant that a philosopher $v_i$ crosses it (modulo the message transmission delay for its neighbours to receive the respective synchronisation message), none of its neighbours can cross it until $v_i$ finishes eating. On the other hand, it can result in starvation, depending on the neighbours' timing.
3. However, the two doorways combined cancel each other's deficiencies and guarantee at most $O(C\Delta)$ waiting time for a philosopher to cross them (Fig. 2).
4. After having crossed both doorways, a process $v_i$ has to wait for its neighbours (only those that are also past both doorways), to pass to it the $pr_{ij}$ privileges. The way that privileges are requested and passed from one node to the other is such that the maximum length of any process waiting chain is bounded by four at any time instant in any execution [12]: A node first requests the privileges from its higher priority neighbours and then from the lower priority ones. A competing node $v_i$, which has crossed both doorways, gives the privilege to a lower priority neighbour only if it has not collected the privileges from all its higher priority neighbours; otherwise it defers answering until it finishes eating. On the other hand, $v_i$ gives the privilege to a higher priority neighbour if it has not collected all the other privileges. This mechanism guarantees that the maximum length of any process waiting chain is bounded by four (two for the doorways and two for the forks) at any time instant in any execution; hence the failure locality is also four.

**Theorem 6 (Choy and Singh 1992 [12])** *The* Choy and Singh *algorithm [12] is a solution to the dining philosophers problem; the worst case response time and the communication complexity are $O(C\Delta)$ (where $C$ is the length of the longest directed path under an initial acyclic orientation or the number of colours of an initial vertex colouring of the conflict graph) and its failure locality is 4.*

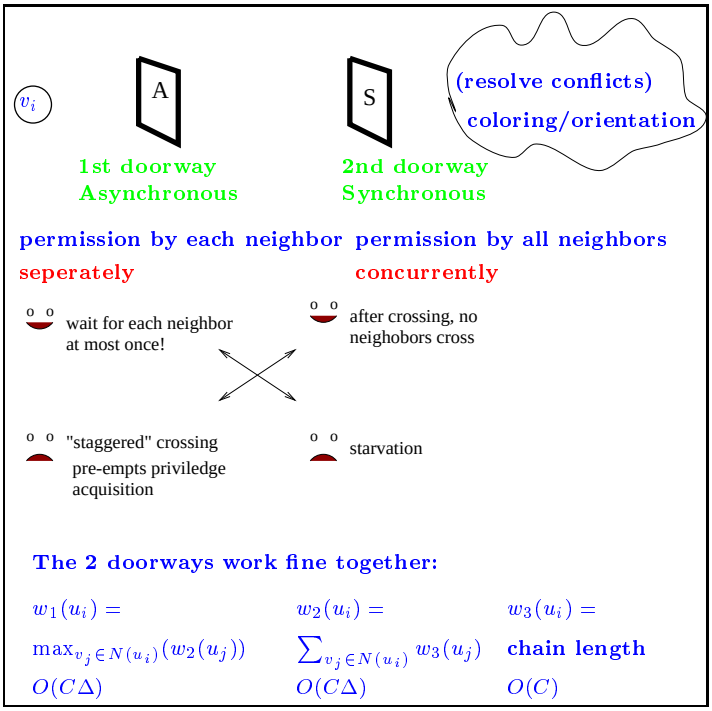



Figure 2: The double doorway synchronisation

Tsay and Barodia in [36] suggested a combination of the dynamic conflict resolution protocol of Chandy and Misra [11] and the preemptive fork collection strategy of Choy and Singh [12] (without the synchronisation barriers) that solves the problem with failure locality 2, but at the cost of having response time $0(n^2)$,

# 6 Solving the mobile philosophers problem

Recall that in this setting the philosophers move between cells. Within each cell broadcast communication is supported. Inter-cell communication is supported via the cell-agents (base stations). The frequencies to be allocated to the philosophers should be chosen by the cell agents in such a way that guarantees that if a channel is used for a communication session of a mobile host (philosopher) in a specific cell, it should not be used concurrently for other communication sessions, except in *distant cells*, outside the scope of the signal. The *free frequencies* at a cell agent are the frequencies of the spectrum that are not in use in that cell or in any of the cells within the scope of the signal, and it is only from this set of frequencies channels that may be allocated to satisfy the requests at this cell. (Requests that cannot be satisfied can be dropped.) Note that the set of free frequencies for a base station changes over time. Furthermore, the same frequency could potentially belong to the set of free frequencies of two adjacent base stations and hence, even when a

base station is picking channels from its own set, it needs to ensure that there is no interference.

Here, besides efficiency in time and communication, since the number of resources to be used is limited and the solution *selects* the resources to be used, here it is important that a solution also aim at maximizing the number of satisfiable requests (*request satisfiability*); this implies that the solution should adapt fast to temporal variations in channel demand in different cells.

The problem of allocating frequencies to mobile philosophers by the base stations (cell agents) can be viewed and analysed as a *generalised* version of the *list colouring* problem, as presented by Garg, Papatriantafilou and Tsigas in [16]. In the list colouring problem, every node of the graph has associated with it a list of colours which in our case are the set of free frequencies. The requirement is to find a proper vertex-colouring of the graph such that each vertex is coloured with one of the colours in its list. The list colouring problem was introduced in [14]; sequential protocols for solving it can be found in [2, 24, 35]. The relation between the classical list colouring and the channel allocation problems has also been noticed in [29]. In the generalised version [16], each node $v_i$ needs to be coloured with a certain number of colours from its list — the number of colours required for a node being the number of pending requests ($r_i$) at the base station at that time. If each colour is viewed as a frequency, solving the mobile philosophers channel allocation problem reduces to solving the *long-lived distributed generalised* version of the list colouring. Thus any protocol for the list colouring problem that is *long-lived*, i.e. it can be invoked multiple times, with lists and requests that vary in time, can be used for solving the channel allocation problem in a dynamic manner.

The *list chromatic number* $\chi_l(G)$ of $G$ is the smallest number $k$ for which, for any assignment of a list $L_i$ of size at least $k$ to every node $v_i \in V(G)$ it is possible to colour $G$ so that every vertex gets one colour from its list. If $\chi_l(G) \leq k$ then $G$ is said to be *k-choosable*. The list sizes that are necessary/sufficient to list colour $G$ give a way to measure the efficiency of a solution with respect to *request satisfiability* as the size of the free spectrum which is sufficient to ensure that the node properly satisfies its pending requests.

## 6.1 Sequential solutions for list colouring (bounds for the request satisfiability)

List colouring a graph generalises the vertex colouring problem and is hence **NP**-hard [10]. The best known achievable (with sequential algorithms) bounds with respect to list sizes are:

**"$\Delta+1$" bound**: Given a graph of maximum degree $\Delta$, one straightforward way of colouring it with $\Delta+1$ colours is to consider vertices one at a time and assign them a colour different from their neighbours. This simple strategy can be adapted to list-colouring – for each vertex we pick a colour from its list that is different from the colours of its neighbours. Therefore, if each list is of size at least $\Delta+1$ we can always list-colour the graph.

**"Outgoing-neighbours requests" bound**: If we have an initial acyclic orientation of the edges of the graph then we could first colour all vertices that are sinks (no two sinks are adjacent) by picking an arbitrary colour from their lists. We then remove these sinks and colour the new sinks created taking care

to assign them a colour different from their (already coloured) neighbours. It is now easy to see that a list colouring is always possible if each vertex has a list of size more than its out-degree in the initial orientation, which is much better than the upper bound given in the previous paragraph. Alon and Tarsi [2] use linear algebraic techniques to argue that lists of these sizes are sufficient, even when the initial orientation of the edges is not acyclic, provided that it satisfies some properties with respect to the number of Eulerian subgraphs.

**Special case: planar graphs**: An elegant argument due to Thomassen [35] shows that every planar graph is 5-choosable. This is optimal since there are planar graphs known which are not 4-choosable [38].

## 6.2 Distributed solutions for the mobile philosophers

The problem has been studied in the distributed setting initially by Prakash et.al. [31]. It has been analysed as a multiple mutual exclusion problem and has been solved with the use of timestamps as in [19]; that algorithm gives response time and failure locality of $O(n)$ and no guarantees for the request satisfiability.

Garg et.al [16] have shown how to derive the generalised distributed equivalent of the solution of the previous section that list colours a graph given an initial acyclic orientation. An acyclic orientation can be obtained by first vertex colouring $G$ (in a distributed manner, with $C$ colours, where $C$ can be $\Delta + 1$ for arbitrary graphs [5] or 6 for planar ones) and then orienting each edge from the higher to the lower colour.

Starting from such an acyclic orientation, let the sinks concurrently and independently list-colour themselves first (cf. Fig. 1). There is no conflict between them when they choose colours since no two sinks are adjacent. The sinks communicate to their neighbours the colours they chose and reverse the orientation of the incident edges, by e.g. using a privilege token for each edge, similarly as forks are used in the Chandy and Misra dynamic conflict resolution algorithm [11]. The resulting orientation is again acyclic, so the new sinks can list-colour themselves next. By repeating this procedure, in time proportional to the length of the longest possible directed path, all the nodes of the graph will be list-coloured, provided that for each node, the sum of the number of its requests and of those of its outgoing-edge neighbours in the initial orientation does not exceed its list size. This scheme not only works for the case when each node is interested in choosing colours once (one-shot protocol), but it can also offer as a *long-lived* solution, as a token-passing-like protocol [8, 11]; any process that becomes a sink at some point holds a privilege to choose frequencies if it has pending requests. However, this approach has a serious drawback, as the chains of the initial orientation get modified in time and in an asynchronous system they can become of length up to $n$. This implies the need of some form of synchronization (as local as possible). The synchronisation mechanism of [12] is shown to be appropriate to solve the new synchronisation issues, too. The resulting protocol (DET-DLC in [16]) uses the same initial acyclic orientation as above and employs the double doorway synchronisation (Fig. 2) and the privilege release mechanisms. Besides, in order that the neighbours of a process $v_i$ have a correct view of their free frequency sets when they are choosing, $v_i$ informs them about the frequencies it picked and waits for acknowledgements before it signals them for the doorways (so that only after consistency of views has been attained

will they be able to compete to make their own choices). When a frequency is not used any more, $v_i$ informs its neighbours to update their sets.

**Request Satisfiability of DET-DLC [16]**: A node, after having crossed the first doorway, will, in the worst case, have to wait for all its neighbours to select frequencies. Since this does not happen more than once, it follows that $v_i$ is guaranteed to get $r_i$ frequencies if, for the set $f$ of its free frequencies at that time it holds:

$$|f| \geq r_i + \sum_{v_j \in N(v_i)} r_j$$

Therefore, the solution guarantees the corresponding (for the distributed generalised long-lived setting) version of the *"Outgoing-neighbours requests" bound* (i.e. the best known to be achievable) regarding the request satisfiability. To achieve this, the protocol uses some extra synchronisation, which implies a small overhead in the response time. On the question whether it is possible to achieve that good request satisfiability without the need for extra synchronisation, it has been shown in [16], that the use of randomisation can help, and achieves the equivalent of the *"$\Delta + 1$" bound* presented in the previous section (i.e. the second best known upper bound for the request satisfiability achievable in a sequential manner).

The idea of the randomised protocol (RAND-DLC in [16]) is for each node $v_i$ to pick randomly, an $\epsilon$ fraction of the colours (frequencies) in its free set; thus each colour in the list is picked with a probability $\epsilon$. The node then runs a *handshaking* phase to check with its neighbours to ensure that some colour it picked is not picked by a neighbour; the colour is dropped if such is the case. Thus a colour picked by two neighbours could potentially be dropped by both of them. Since no answers are deferred the maximum waiting chain length is one.

**Request Satisfiability of RAND-DLC [16]**: Since the colour is retained when none of the neighbours picks it and since the number of neighbours who could be picking colours is at most $\Delta$, the probability that a colour is retained is at least $(1-\epsilon)^{\Delta}$. Let $X_j$ be a random variable that is 1 if the $j^{\text{th}}$ colour in the list (of some $v_i$) is acquired (and is 0 otherwise). Then the probability that $X_j = 1$ is at least $\epsilon(1-\epsilon)^{\Delta}$. Let further $X = X_1 + X_2 + \ldots + X_f$ be the sum of these random variables. The expected value of the random variable $X$ is at least $f \cdot \epsilon(1-\epsilon)^{\Delta}$ which is maximised when $\epsilon = 1/(\Delta+1)$. For this choice of $\epsilon$, the expected number of colours acquired is at least

$$\mu = \frac{f}{\Delta}\left(1 - \frac{1}{\Delta+1}\right)^{\Delta+1} \geq \frac{f}{4\Delta}$$

Since $X$ is the sum of independent Bernoulli trials, Chernoff's bounds [20] bound the probability that the number of colours acquired is at least $(1-\delta)\mu$ to give for $\delta = \sqrt{\frac{2}{\mu}}$, that the probability that we acquire $\mu - \sqrt{2\mu}$ colours is at least $1 - e^{-1}$.

**Theorem 7 (Garg, Papatriantafilou and Tsigas 1996 [16])** *Algorithms DET-DLC and RAND-DLC [16] are solutions to the mobile philosophers problem.*
*Algorithm DET-DLC guarantees worst case response time is $O(C\Delta)$ (where $C$ is the length of the longest directed path in $G$ under an acyclic orientation), the communication complexity is $O(C\Delta)$ and the failure locality is 4.*
*Algorithm RAND-DLC has failure locality 1. With the exchange of $3\Delta$ messages and*

*in 3 rounds of communication a node $v_i$ gets at least $r = \mu - \sqrt{2\mu}$ frequencies with probability at least $1 - e^{-1}$, where $\mu = \frac{f}{4\Delta}$ and $f$ is the number of free frequencies for $v_i$ at that point of the execution.*

# References


[1] Y. Afek, B. Awerbuch, S. Plotkin and M. Saks Local Managment of a Global Resource in a Communication Network. *Proc. of IEEE FOCS 1987*, pp. 347-357.

[2] N. Alon and M. Tarsi. Colorings and Orientations of Graphs. *Combinatorica* 12 (2) (1992), pp. 125–134.

[3] H. Attiya, H. Shachnai and T. Tamir Local Labeling and Resource Allocation Using Preprocessing. *Proc. of WDAG 1994*, pp. 194–208.

[4] B. Awerbuch Complexity of network synchronization. *Journal of the ACM*, 32, pp.804–823, 1985.

[5] B. Awerbuch, A. Goldberg, M. Luby and S. Plotkin. Network Decomposition and Locality in Distributed Computation. *Proc. of IEEE FOCS 1989*, pp. 364–369.

[6] B. Awerbuch and M. Saks A Dining Philoshopher Algorithm with Polynomial Response Time. *Proc. of IEEE FOCS 1990*, pp. 65-74.

[7] B.R. Badrinath, A. Acharya and T. Imielinski. Impact of Mobility on Distributed Computations. *Operating Systems Review*, Vol. 27, No. 2, Apr. 1993.

[8] V. Barbosa and E. Gafni. Concurrency in Heavily Loaded Neighborhood-Constrained Systems. *ACM TOPLAS*, Vol. 11, No. 4, pp. 562-584, Oct. 1989.

[9] J. Bar-Ilan and D. Peleg Distributed Resource Allocation Algorithms. *Proc. of WDAG 1992*, pp. 276–291.

[10] M. Biró, M. Hujter and Z. Tuza. Precoloring Extensions. I. Interval Graphs. *Discrete Math.* **100**, pp. 267–279, 1992.

[11] K.M. Chandy and J. Misra. The Drinking Philosophers Problem. *ACM TOPLAS*, Vol. 6, No. 4, pp. 632-646, Oct. 1984.

[12] M. Choy and A. Singh. Efficient Fault Tolerant Algorithms for Resource Allocation in Distributed Systems. *ACM TOPLAS*, Vol. 17, No. 3, pp. 535-559, May 1995. (Also in *Proc. of ACM STOC 1992*, pp. 593-602).

[13] E.W. Dijkstra Hierarchical Ordering of Sequential Processes. *Acta Informatica*, 1(2), pp. 115–138, 1971 (also in *Operating Systems Techniques*, Academic Press 1972).

[14] P. Erdös, A.L. Rubin and H. Taylor. Choosability in Graphs. *Proc. of the West Coast Conference on Combinatorics, Graph Theory and Computing*, (Congr. Num 26), pp. 125–157, 1979.

[15] M. Fisher, N. Lynch, J. Burns and A. Borodin Distributed FIFO Allocation of Identical Resourses Using Small Shared Space *ACM TOPLAS*, Vol. 11, No. 1, Jan. 1989, pp. 91–114.

[16] N. Garg, M. Papatriantafilou, Ph. Tsigas Distributed List Coloring: How to Dynamically Allocate Frequencies to Mobile Base Stations. *Proc. of IEEE SPDP 1996*, pp. 18-25.

[17] V. Garg *Principles of Distributed Systems.* Kluwer Academic Publishers, 1996.

[18] S. Ghosh and M.H. Karaata. A self-stabilizing algorithm for coloring planar graphs. *Distributed Computing* **7**, pp. 55-59, 1993.

[19] D. Ginat, A. U. Shankar and A.K. Agrawala An Efficient Solution to the Drinking Philoshophers Problem and its extensions. *Proc. of WDAG 1989*, pp. 83-93.

[20] T. Hagerup and C. Rüb. A Guided Tour of Chernoff Bounds. *Information Processing Letters* **33** (1989/90), pp. 305-308.

[21] D. Hauschidt and R. Valk Safe States in Banker-like Resource Allocation Problems. *Information and Computation*, **75**, 232-263 (1987).

[22] S.G. Hild. A Brief History of Mobile Telephony *Technical Report No. 372, University of Cambridge, Computer Laboratory*, Jan. 1995.

[23] G. Ioannidis. Mobile Computing. *PhD Thesis, Columbia University*, 1992.

[24] T.R. Jensen and B. Toft. Graph Coloring Problems. *Wiley-Interscience Series in Discrete Mathematics and Optimization*, 1995.

[25] L. Lamport Time, Clocks, and the Ordering of Events in a Distibuted System. *Communications of the ACM*, Vol. 21, No. 7, July 1978, pp. 558-565.

[26] N. Lynch Upper Bounds for Static Resourse Allocation in a Distributed System. *Journal of Computer and System Science*, **23**, 254-278 (1981).

[27] N. Lynch Distributed Algorithms. *Morgan-Kaufmann*, 1995.

[28] N. Lynch, N. Griffeth, M. Fisher and L. Guibas Probabilistic Analysis of a Network Resourse Allocation Algorithm. *Information and Control*, **68**, 47-85 (1986).

[29] E. Malesinsca. An Optimization Method for the Channel Assignment in Mixed Environments. *Proc. of ACM MobiCom '95.* Also available as TR. No. 458/1995, Fachbereich Mathematik, Technische Universität Berlin.

[30] S. Murphy and A. U. Shankar Technical Correspondence: A Note on the Drinking Philosopher Problem. *ACM TOPLAS*, Vol. 10, No. 1, pp. 178-188.

[31] R. Prakash, N. G. Shivarati and M. Singhal. Distributed Dynamic Channel Allocation for Mobile Computing. *Proc. of ACM PODC 1995*, pp. 47–56.

[32] M. O. Rabin and D. Lehman On the Advantages of Free Choice: A Symmetric And Fully Distributed Solution to the Dining Philosophers Problem. *Proc. of ACM PoPL 1981*, pp. 133-138.

[33] K. Ramamrithan, J. A. Stankovic, and W. Zhao Distributed Scheduling of Tasks with Deadlines and Resource Requirements. *IEEE Transactions on Computers*, Vol. 38, No. 8, August 1989, pp. 1110-1123.

[34] E. Styer and G. Peterson Improved Algorithms for Distributed Resource Allocation. *Proc. of ACM PODC 1988*, pp. 105-116.

[35] C. Thomassen. Every Planar Graph Is 5-Choosable. *Journal of Combinatorial Theory*, Series B **62**, pp. 180-181, 1994.

[36] Y. Tsay and R. Bargodia An Algorithm with Optimal Failure Locality for The Dining Philosopher Problem. *Proc. of WDAG 1994*, pp. 296–310.

[37] J. Welch and N. Lynch A Modular Drinking Philoshopher Algorithm. *Distributed Computing*, (1993) **6**:233-244.

[38] M. Voigt. List Colorings of Planar Graphs. *Discrete Mathematics* **120**, pp. 215-219, 1993.