Optimality and Greed in Dynamic Allocation

Peter Winkler
Bell Labs 2C–365
700 Mountain Ave.
Murray Hill, NJ 07974 U.S.A.
pw@lucent.com

October 24, 2001

Abstract

In dynamic allocation items arrive and depart randomly, and while present are stored in a limited facility; the job of an allocation algorithm is to decide whether and where to store an arriving item, while trying to minimize the cost incurred by rejections. Ordinarily, to prove the value of such an algorithm, one must either assume a specific arrival distribution (e.g. Poisson–λ), or try to obtain bounds relative to an adversary (as in competitive analysis).

We present here a novel method of proving precise optimality for certain kinds of dynamic allocation algorithms, without assuming a specific arrival distribution. We do need to assume memoryless departure for at least some item types, and most importantly, we must assume conditions are such that it is not right to reject arrivals unnecessarily.

The method is applied to two simple call assignment problems which arose, in quite different circumstances, at Lucent Technologies. In both cases the method was successful in showing that the intuitively best assignment algorithm really does minimize expected cost under fairly general, and essentially necessary, assumptions.

1 Introduction

It is notoriously difficult to prove optimality—indeed, even to have it—for an on-line algorithm, unless a specific probability distribution is assumed for the algorithm's input. In what follows we present a technique which can be used to show optimality for certain dynamic allocation algorithms. While far from general, the technique has already found two applications.

The following "call and trunk" problem was brought to the author's attention by Barnet Schmidt, of Lucent Technologies' Network Systems (some parameters have been generalized and unimportant details omitted in order to protect proprietary information). A certain facility consists of m bins containing k slots each, thus a total of $n = km$ slots. At random times "calls" and "trunks" arrive and depart (terminate). When a call arrives it may be put into an empty slot, where it remains until departure, or it may be rejected; similarly a trunk may be assigned to an empty bin (that is, a bin all of whose slots are unoccupied) where it remains until departure, or it may be rejected. Rejections may of course be necessary when slots and bins are occupied, but they should be minimized in general. The facility is indicated schematically in Fig. 1.

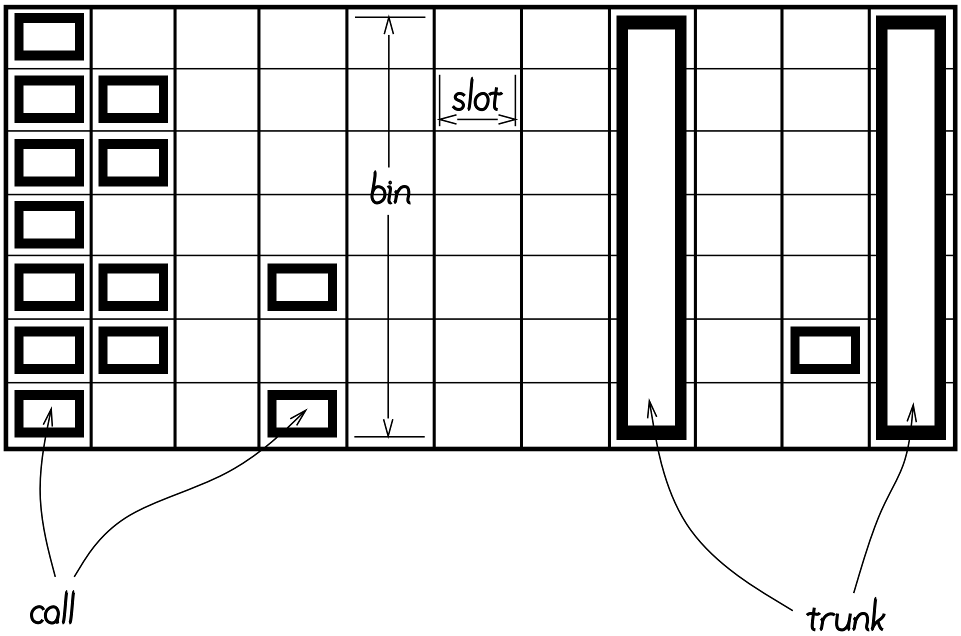

Figure 1: Slots, bins, calls and trunks

This problem is an example of what has been called in the literature *dynamic storage allocation* (see, e.g., [5, 6, 7]). There are actually two 'dynamic' aspects to dynamic storage allocation: the necessity for assigning space as requests arrive, and the spontaneous departure of items subsequent to storage. Without the latter, the problem is one of (on-line) *bin packing*, also a well-studied subject (see [8]).

Also related is on-line load balancing, in which the object is to minimize the maximum load on any machine (bin). This problem is tackled e.g. by Azar and Epstein [1], using competitive analysis to evaluate algorithms without making any explicit assumption about arrival distribution. Competitive analysis has proven useful and enlightening in many applications but is not a technique for proving optimality.

In on-line bin packing and dynamic storage allocation, two algorithms often considered are First Fit, in which each item is placed in the lowest-numbered bin in which it fits, and Best Fit, in which each item is placed in the fullest bin in which it fits. Since the number of bins is fixed in our problem, the latter algorithm, especially, is an obvious choice.

It is easy to imagine circumstances under which Best Fit is not optimal. For example, the relative cost of rejecting calls and trunks (considered in relation to arrival and duration statistics) may be such that all calls, or perhaps all trunks, should be rejected. Or, more subtly, one might wish to reserve one bin for trunks.

In the problem as it arose at Lucent Technologies, arrival rates are time-dependent and only general information known about their distribution. However, independent exponential waiting time had been observed to be a reasonable model for call departures. Trunk departure times, as it happens, play no direct role in the proof.

There is a cost for rejecting a call and (a larger one) for rejecting a trunk; surprisingly, we do not need to know what those costs are, only that it doesn't pay to reject calls or trunks unnecessarily. Even without assumptions about arrival times, it seems intuitively reasonable under these circumstances that Best Fit should be optimal. Indeed, we show below that a reasonable interpretation of the circumstances provides a sufficient, and in a sense necessary, set of conditions for Best Fit to be optimal.

The second problem arose in wireless telephony, using a call multiplexing scheme known as TDMA (time division multiple access). In TDMA up to n calls at the same radio frequency are handled simultaneously in a single cell, by dividing time into intervals of n time slots each and assigning one slot from each interval to a call in rotating fashion.

A difficulty arises when two adjacent cells are using the same frequency;

then interference considerations force them to share the same time slots, so that the total number of calls handled at that frequency is still only n. The actual situation is worse (but much more interesting) than that, because wireless base stations are unable to synchronize their clocks closely enough to make the schedules mesh. Thus one occupied time slot from Cell 1 knocks out *two* time slots from Cell 2's schedule (see e.g. Fig. 3). When calls in both cells are in progress at most $n-1$ can be handled at the same time, and even then only when calls in the same cell are in contiguous time slots. In this work we will concentrate on the $n = 6$ case, which is the most interesting case among those which do not require additional assumptions. Discussion of other cases, plus empirical and computational results and more background, is to be found in [3].

Our proof-of-optimality technique is best explicated by demonstration on a concrete problem, hence we will begin by applying it directly to the call and trunk problem above. We then extract from it a general theorem which is applied to the TDMA problem. We define optimality in Section 2 and spell out our two conditions in Sections 3 and 4. The proof for the call and trunk problem is presented in Section 5, and the general proof in Section 6. Finally the TDMA problem is tackled in Section 7.

2 Optimality

A dynamic allocation problem involves a finite facility in which an arriving object (from a finite list of types) can be stored until it departs. The state of a facility is a vector which describes which types of objects are stored in which places, at any given time; there is a finite set $\mathbf{S}$ of legal states, which is closed downward in the sense that if an object is removed from a legal state the result is another legal state.

Arrivals and departures take place during a fixed time interval $[0, T]$ (at the start of which the facility is in some fixed state, usually empty) and are governed by random processes in which the expected number of events is finite, and two events never occur at the same time.

The *cost* incurred in running the facility is a non-negative, non-decreasing function of the set of rejected arrivals, and has the property that the expected cost of rejecting all the arriving objects is finite. Typically the cost is a weighted sum of the numbers of rejected objects of each type; or the same but discounted over time. However, it is sometimes useful to consider a more general cost function, e.g. one which tries to encourage fairness by extracting a heavier penalty for rejecting too many objects of one type.

The decision of whether and where to store an arriving object is made by an *assignment algorithm* which (a) must not cause the facility to enter an illegal state, (b) cannot foresee the future, and (c) does not itself affect the arrival and departure processes. The algorithm *is* permitted to keep track of, and make decisions based on, the entire sequence of events up to the current time; its own past decisions; and (thus) the current state of the facility. We allow algorithms to consider the *times* at which events occur, as well as their order, but not to infinite precision (else we would have to worry about non-measurable algorithms). Accordingly we allow the event sequence maintained in the algorithm's memory to be an ordered list of arrivals and departures each of which is equipped with a time measured to the nearest whole unit (say, nanoseconds). We assume any event sequence is possible (although most will be highly improbable) even if it calls for 14,000 objects to arrive during the first nanosecond.

Every algorithm $\mathbf{A}$ generates a finite expected cost $\mathrm{E}\mathbf{A}$, the expectation taken over the probability space of event sequences. Let M be the infimum of $\mathrm{E}\mathbf{A}$ over all algorithms $\mathbf{A}$; $\mathbf{A}$ is *optimal* if $\mathrm{E}\mathbf{A} = M$.

Theorem 2.1. *Optimal algorithms exist.*

Proof. For any algorithm $\mathbf{A}$, let $\mathbf{A}\restriction j$ be the algorithm which behaves like $\mathbf{A}$ for the first j events and then rejects all subsequent arrivals. Note that for any j, there are only finitely many possible choices for $\mathbf{A}\restriction j$.

Let $\mathbf{A}_i$ be a sequence of algorithms for which $\lim_{i\to\infty} \mathrm{E}\mathbf{A}_i = M$. For each $j = 1, 2, \ldots$, choose an algorithm $\mathbf{B}_j$ so that $\mathbf{B}_j \restriction j-1 = \mathbf{B}_{j-1}$, and $\mathbf{A}_i \restriction j = \mathbf{B}_j$ for infinitely many values of i. We claim that $\mathbf{B} := \lim_{j\to\infty} \mathbf{B}_j$ is optimal.

To see this, assume otherwise and let $\varepsilon = \mathrm{E}\mathbf{B} - M > 0$. Choose i_0 so that $i > i_0$ implies $\mathrm{E}\mathbf{A}_i < M + \varepsilon/2$. Since the mean number of events and mean cost are finite, we may also choose j so that the expected additional cost of rejecting all arrivals after the jth event is less than $\varepsilon/2$. Let $i > i_0$ be such that $\mathbf{A}_i \restriction j = B_j$; then $\mathrm{E}\mathbf{B} < \mathrm{E}\mathbf{B}_j < \mathrm{E}\mathbf{A}_i + \varepsilon/2 < M + \varepsilon$, a contradiction. $\square$

3 Greed

An algorithm may choose to reject an arriving object even though it "has room", i.e. accepting it results in a legal state. This is known in the telephone business as *admission control.* In some real-world dynamic allocation problems, this behavior is not permitted; in others it is not only permitted but, in certain situations, advisable.

An algorithm that does not reject arrivals unnecessarily will be called *greedy*[1]. Since the algorithms whose optimality we intend to establish are in fact greedy ones, we must either forbid non-greedy algorithms or show somehow that they are inferior. We will see later that the former will not work, but the weakest possible manifestation of latter works surprisingly well: just knowing that *conditions are such that optimal algorithms are greedy* can help to tell us which greedy algorithm is the best.

The question of whether the "greed is good" assumption is justified, in a given situation, may be a delicate one. Intuition suggests that in sufficiently low traffic, this should be true. For example, suppose that in the call and trunk problem, call durations are exponential mean α and trunk durations exponential mean β; call arrivals are slowly-varying Poisson at rate bounded by $\lambda < 1/\alpha$, and trunks by $\mu < 1/\beta$; each call rejection costs an amount a, each trunk b. Then if

$$\frac{\beta\lambda}{1+\beta\mu} < b/a < \frac{1-\alpha\lambda}{\alpha\mu} ,$$

all optimal algorithms are greedy. How do we know this? Because the expected value of objects arriving during a call's lifetime is at most

$$\alpha(a\lambda + b\mu) < a$$

and the expected value of objects arriving during a trunk's lifetime is at most

$$\beta(a\lambda + b\mu) < b .$$

Thus if algorithm $\mathbf{A}$ rejects an object x arriving at time t, then the algorithm $\mathbf{A}'$ which accepts x, rejects *every* new object which arrives until x departs and then behaves like $\mathbf{A}$ thereafter, has lower expected cost than $\mathbf{A}$.

For the problems described in this work, admission control is generally not practiced nor is there reason to believe that it should be. The usual wisdom is to take the "bird in hand". In these cases one cannot expect to *prove* that it is right to be greedy, but one can reasonably assume it, pending evidence to the contrary, and treat the results with the respect due the assumption.

4 Holding Times

If, in a dynamic storage facility, we have information about when certain items are due to depart, then very sophisticated planning may be called for

[1] sometimes called "fair", as in [4].

and we cannot expect any simple algorithm to be optimal. Thus our cavalier attitude toward arrival processes cannot be extended to departure times; here, for at least some kinds of items, we must assume no foreknowledge about departure times.

In particular, in the call and trunk problem we must assume that at any time all *calls* in the system are equally likely to depart (we do not care about trunk departure times, although of course they may affect the greediness assumption). In the TDMA problem we assume that at any time all calls of a given type are equally likely to depart.

These assumptions follow from independent, exponential holding times for the calls in question—the same fixed rate for each call type. This model is often reasonable in telephony and is frequently used for purposes of analysis and simulation. Since in the actual problem no record is kept of a given call's arrival time, it would be difficult to take advantage of non-exponential holding time distributions. In practice we are reduced to hoping that the effects of deviations from memoryless behavior are minor.

5 Optimality of Best Fit

We now show that our greediness and holding time assumptions suffice to prove optimality of Best Fit in the call and trunk problem.

Theorem 5.1. *Suppose that in the call and trunk problem*

1. *Every optimal assignment algorithm is greedy, i.e. accepts a call or trunk whenever possible;*

2. *At any time all calls currently in progress are equally likely to depart.*

Then Best Fit is optimal, and moreover it is the only optimal assignment algorithm.

Remark. A stronger (and perhaps more natural-looking) statement would be that given the equal-likelihood call termination assumption, Best Fit minimizes expected cost *among all greedy algorithms.* However, such a statement is far from being true. Imagine, for example, a situation in which calls are so frequent, and trunks so long-lived and cheap to reject, that it rarely pays to accept a trunk. Then, among greedy algorithms, Worst Fit will be far superior to Best Fit—calls should be spread over the bins as evenly as possible, maximizing the probability that no bin will be available when a trunk arrives. (We suspect that certain public facilities are operated in this manner.)

Proof. Note that it makes no difference which bin is used for a trunk, nor which slot in a given bin is used for a call, nor which of equally occupied bins is used for a call. For definiteness, we let **B** stand for the particular Best Fit algorithm which accepts all calls and trunks when possible, always putting a trunk into the highest-numbered empty bin, and always putting a call into the lowest-numbered empty slot in the lowest-numbered bin which has an empty slot, and has as few empty slots as any other non-full bin.

We describe the state of the facility at any given time by means of a vector $\bar{a} = \langle a_1, a_2, \ldots, a_m \rangle$ where a_i is the number of active calls in the ith bin; and we continually renumber the bins so that at all times $a_1 \geq a_2 \geq \cdots \geq a_m$. It will turn out not to be necessary to distinguish an empty bin from one containing a trunk.

We also continually renumber the calls $1, \ldots, w$ being serviced at a given time, so that $f(u) \leq f(v)$ for $u < v$ where $f(u)$ is the bin to which call u is assigned. Thus, for example, as long as at least one call is being carried, $f(1) = 1$; and the number of empty or trunk-carrying bins available is $m - f(w)$ where $w = \sum_{i=1}^{m} a_i$.

Let **B** be the Best Fit algorithm described above and assume that some other, non Best Fit algorithm **A** is optimal, with the intent of deriving a contradiction. To do this we run **A** and **B** side by side, with fixed call and trunk arrival times, fixed trunk departure times, and a fixed common initial state. The facility into which algorithm **A** assigns its calls and trunks will be called "System A" while **B** works on "System B."

However, we do not assume that a call which arrives at time t departs at the same time in **A**'s world as it does in **B**'s; instead, we couple the call departures by terminating calls *with the same current number* simultaneously in both worlds. Thus, at the moment call u terminates in System A, the call numbered u terminates in System B; even though, on account of renumbering, these calls may have arrived at different times. The validity of this coupling relies on the assumption that all calls in progress are equally likely to terminate. In effect, we have created a probability-preserving isomorphism between the space of **A** worlds and the space of **B** worlds (see Fig. 2).

The coupling cannot be executed, however, unless at every moment the same number of calls are being carried by the two systems. If, say, a trunk were to arrive and be accepted by one system and not the other, then later the system which took the trunk might be forced to reject calls accepted by the other, ruining the coupling.

We must therefore ensure that the same number of calls *and* trunks are being handled by the two systems, at all times. To do this we use not the algorithms **A** and **B** themselves, but alterations $\mathbf{A}'$ and $\mathbf{B}'$ which depend on each other as follows:

- Neither $\mathbf{A}'$ nor $\mathbf{B}'$ accepts a call or trunk unless, at the moment of arrival, there is adequate space in *both* systems;
- Otherwise $\mathbf{A}'$ behaves exactly like **A** and $\mathbf{B}'$ like **B**.

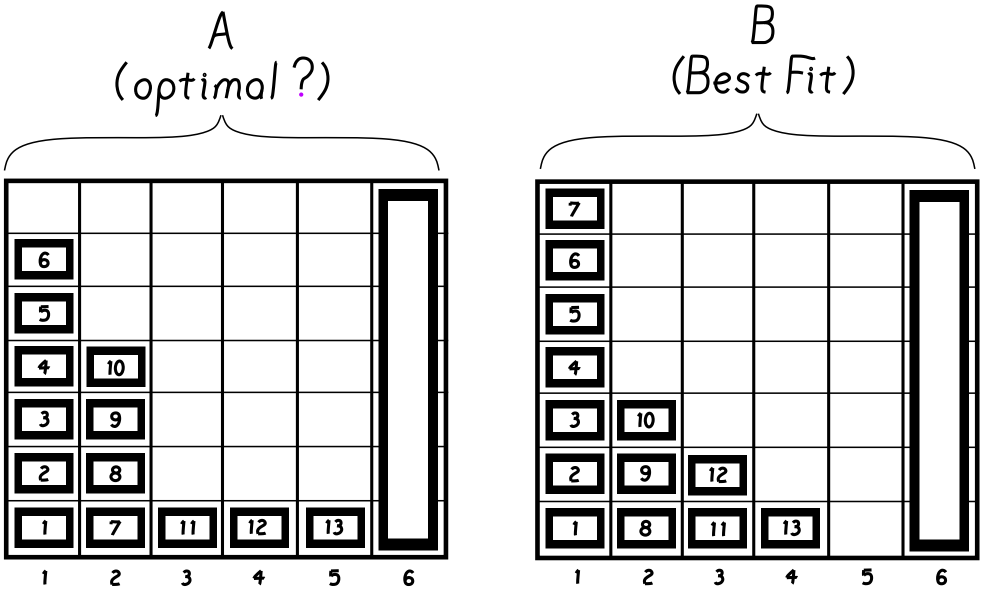

Figure 2: Coupling of algorithms

It may seem as if there is little to be gained by comparing $\mathbf{A}'$ and $\mathbf{B}'$ since, perforce, they reject the same calls and trunks. Our strategy, however, is to show that in fact $\mathbf{A}'$ is no different from **A**: all calls and trunks rejected by the former would have been rejected anyway by the latter. On the other hand, we will see that $\mathbf{B}'$ can reject a trunk for which adequate space exists in System B. Thus, by the greediness assumption, $\mathbf{B}'$ cannot be optimal. Then neither is $\mathbf{A}'$, whose performance is the same as $\mathbf{B}'$, but this contradicts the optimality of **A**, and the theorem will then follow.

We use $\bar{a}$ to denote the state of System A and $\bar{b}$ for System B. We define a partial order on the state space by majorization: $\bar{a} \preceq \bar{b}$ just when

$$\sum_{i=1}^{s} a_i \leq \sum_{i=1}^{s} b_i$$

for each s, $1 \leq s \leq m$.

Suppose that calls in each system are numbered left to right, as described above, with $f(u)$ denoting the bin number of call u in system A and $g(u)$ in system B. Then it is immediate that $\bar{a} \preceq \bar{b}$ implies $f(u) \geq g(u)$ for every u. Observe, however, that the converse also holds. Let $\bar{e}_i$ be the vector with 1 in coordinate i and 0's elsewhere; then for each call u, $f(u) \geq g(u)$ implies $\bar{e}_{f(u)} \preceq \bar{e}_{g(u)}$. Thus

$$\bar{a} = \sum_{u} \bar{e}_{f(u)} \preceq \sum_{u} \bar{e}_{g(u)} = \bar{b}$$

since majorization is preserved by summing.

We now begin the two systems in any state $\bar{a} = \bar{b}$ (with, also, the same number of trunks present) and prove by induction that at all subsequent times, $\bar{a} \preceq \bar{b}$.

To see that this would suffice to complete the proof, note first that if $\bar{a} \preceq \bar{b}$ then System B has at least as many empty bins as System A. Hence no trunk is ever rejected by $\mathbf{A}'$ solely on account of the state of System B; and the number of slots for calls will always be the same in both worlds.

On the other hand, since $\mathbf{A}$ is not a Best Fit algorithm then we can set up a history in which, at some point, $\mathbf{A}$ assigns (with positive probability) the next call to a less-than-best-fit bin. Immediately after that call we will have

$$\sum_{i=1}^{s} a_i < x = \sum_{i=1}^{s} b_i$$

for some $s < m$. If all calls numbered higher than x now depart, we will have $a_{s+1} > 0 = b_{s+1}$, thus there will be an extra open bin in System B. A sequence of trunk requests will now cause $\mathbf{B}'$ to reject a trunk request on $\mathbf{A}$'s account. Then $\mathbf{B}'$ is not optimal, but its performance is equal to that of $\mathbf{A}'$ (via the isomorphism of probability spaces) and $\mathbf{A}'$ is the same algorithm as $\mathbf{A}$. This is our desired contradiction.

We have finally reached the combinatorial heart of the proof: we need to show that if $\bar{a}$ and $\bar{b}$ are the system states before an arrival or departure, and $\bar{a}'$ and $\bar{b}'$ the new system states, then $\bar{a} \preceq \bar{b}$ implies $\bar{a}' \preceq \bar{b}'$. Arrivals and departures of trunks have no effect on the state vector, so we begin with call arrivals.

We may assume that the arriving call is placed by $\mathbf{A}'$ in the leftmost of a string of bins with given occupancy, and likewise for $\mathbf{B}'$, so that the bins will not need to be renumbered. It is then clear that the partial sums $\sum^s a_i$ and $\sum^s b_i$ can only remain the same or increase by 1.

In fact, the new call is placed by $\mathbf{B}'$ in bin j, where $b_i = k$ (the total number of slots in a bin) for $i < j$, and $b_j < k$. Then $\sum^s b'_i = 1 + \sum^s b_i$ for $s \geq j$ so we are in no danger there, and $\sum^s b'_i = sk$ remains maximal for $s < j$. Thus there is no place where majorization can be lost.

The situation for call departures is even simpler. Prior to the event the calls are numbered left to right, thus $f(u) \geq g(u)$ for each call u by the induction assumption. But the same numbered call departs in both systems, hence majorization is preserved vector by vector. □

6 A General Theorem

The combinatorial heart of the above proof is the majorization argument, from which we conclude that $\mathbf{A}' = \mathbf{A}$. In the general case majorization is replaced by a quasi-ordering of the state set $\mathbf{S}$, together with a coupling of departures. A quasi-ordering "$\leq$" (see e.g. [2]) is a transitive, reflexive binary relation; it differs from a partial ordering in permitting ties, that is, we can have $x \leq y \leq x$ without $x = y$. In this case we say that x and y are equipotent, and write "$x \simeq y$".

Assume that items belong to types $1, \ldots, \tau$. The set $\mathbf{S}$ of allowable states, which is required to be closed under substates, is naturally partitioned into "state classes" $\mathbf{S}(n_1, \ldots, n_\tau)$ consisting of those states containing precisely n_j items of type j. We denote by $S + j$ the set of states reachable from state S by accepting an item of type j, and by $S - j$ the set of states which result when an item of type j departs. If $\mathbf{A}$ is an assignment algorithm then we denote by $\mathbf{A}(S + j)$ the particular state in $S + j$ chosen by $\mathbf{A}$ when an item of type j arrives to find the facility in state S; in general $\mathbf{A}(S + j)$ may depend on time and history as well as on j and S.

We will assume that departures of items of types $1, \ldots, \mu$, $\mu \leq \tau$, are memoryless; that is, at any time, every item in the facility of type j is as likely to depart as any other, for $j \leq \mu$. This assumption will be called the "departure assumption for types 1 through μ".

Finally, we assume every optimal assignment algorithm is greedy. This will be called the "greediness assumption."

Theorem 6.1. *Suppose that a dynamic allocation facility satisfies the greediness assumption and the departure assumption for types* 1 *through* μ*, and* *suppose there is a greedy assignment algorithm* $\mathbf{B}$ *and a quasi-ordering "$\leq$" on the state space* $\mathbf{S}$ *which satisfy the following for all states* $S \leq T$*:*

(i) *If* $S + j \neq \emptyset$ *then* $T + j \neq \emptyset$*, for every* $j \leq \tau$*.*

(ii) $S' \leq \mathbf{B}(T + j)$ *for every* $S' \in S + j$*, and every* $j \leq \tau$*.*

(iii) *For each* $j \leq \tau$ *there is a bijection* π_j *from the items of type* j *in* S *to the items of the same type in* T*, such that if the departure of item* i *from* S *results in state* S' *and the departure of item* $\pi_j(i)$ *from* T *results in state* T'*, then* $S' \leq T'$*. Moreover, for* $j > \mu$ *the bijections* $\{\pi_j\}$ *can be chosen arbitrarily.*

Then algorithm $\mathbf{B}$ *is optimal.*

Roughly speaking, we think of state T as being "better than" state S if $S \leq T$. In the call and trunk problem, this relation was realized by majorization "$\preceq$" between state vectors; in the TDMA problem to follow, the relation is more *ad hoc* (Fig. 7). Condition (i) ensures that any arrival acceptable by S can also be accepted by T. Condition (ii) says that algorithm $\mathbf{B}$ not only picks the best state in $T + j$, but that this state dominates anything in $S + j$. Condition (iii) implies, first of all, that S and T are in the same class $\mathbf{S}(n_1, \ldots, n_\tau)$; secondly, that departures can be coupled so as to preserve the partial order; and thirdly, that for items not covered by the departure assumption, any coupling will do.

Proof. Let $\mathbf{B}$ satisfy the conditions of the theorem statement, and suppose it is not optimal. Using Theorem 2.1 we choose some optimal algorithm $\mathbf{A}$. We put $\mathbf{B}$ up against $\mathbf{A}$ from the same starting state S_0, and alter them to $\mathbf{B}'$ and $\mathbf{A}'$ by forbidding either to accept any item that cannot be accepted by the other.

We prove by induction that at each time t, if S is the state of the facility under $\mathbf{A}'$ and T the state under $\mathbf{B}'$, then $S \leq T$. Of course $S_0 \leq S_0$, and condition (ii) preserves the quasi-ordering through arrivals. Departures of items of types $\mu+1$ and higher cause no difficulties, and the departure assumption for types 1 through μ allows us to couple departures so as to preserve the quasi-ordering through all departures.

Now condition (i) shows that in fact $\mathbf{A}' = \mathbf{A}$. Since the cost function depends only on rejections, $\mathbf{A}'$ and $\mathbf{B}'$ have the same expected cost; thus $\mathbf{B}'$ is optimal. Then the greediness assumption says $\mathbf{B}'$ is greedy, therefore no different from $\mathbf{B}$. Thus $\mathbf{B}$ is optimal after all. $\square$

In the call-trunk example we had $\tau = 2$ and $\mu = 1$, and there **B** was the only algorithm which satisfied conditions (i) through (iii) above relative to majorization. Thus, with the assumption that all events are possible, we could show that **B** was the *unique* optimal algorithm. Theorem 6.1 can easily be extended to this stronger form.

7 The TDMA Problem

For the TDMA problem $\tau = \mu = 2$, that is, there are two types of items (calls in Cell 1 and calls in Cell 2) and both types are assumed to have memoryless departures. It is not necessary that the mean lifetimes, or rejection costs, be the same for calls of type 1 and calls of type 2; nevertheless the arguments below are symmetric. We tackle the $n = 6$ case, where n is the number of time slots, because when $n \geq 7$ additional assumptions are required and when $n \leq 5$ the problem becomes less interesting.

We represent the six time slots available to Cell 1 by partitioning the outer perimeter of a circle and filling in the segment to represent a call; for Cell 2 we partition the inner perimeter instead, in such a way that two inner segments overlap each outer segment. Fig. 3 shows how the system might appear on a time line, and the corresponding circular representation.

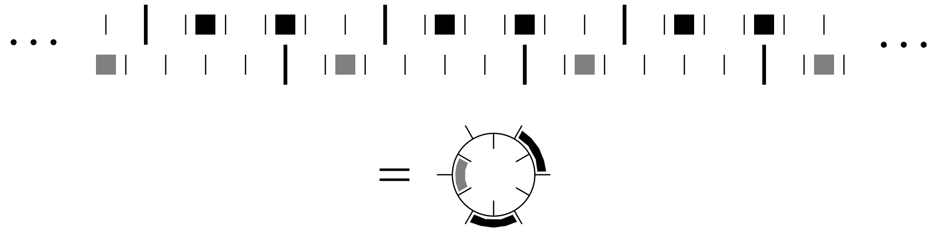

Figure 3: Circular representation of slots in the TDMA problem

For purposes of analysis it is useful to identify states which differ only by rotation, reflection or interchange of the two call types. Fig. 4 gives examples of these three types of symmetry, which result in equivalence classes of size up to 24. Thus identified, there are only 26 different states, each of which has a representative in one of the following state classes: $S(0,0)$, $S(1,0)$, $S(2,0)$, $S(3,0)$, $S(4,0)$, $S(5,0)$, $S(6,0)$, $S(1,1)$, $S(2,1)$, $S(3,1)$, $S(4,1)$, $S(2,2)$ or $S(3,2)$.

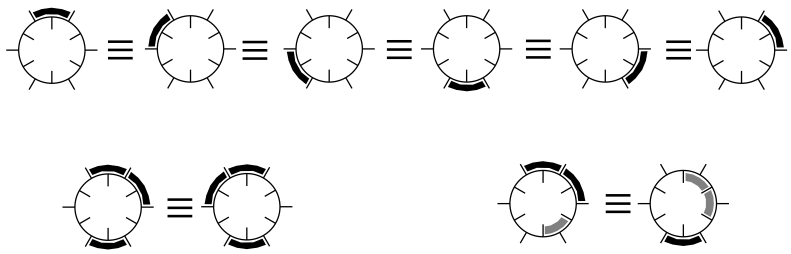

Figure 4: State symmetries in the TDMA problem

Fig. 5 shows a possible sequence of states as calls arrive and depart. This particular sequence illustrates two undesirable effects: inefficiency of use brought about by discontiguous occupation of slots, and “lockout” of calls of one type. One can easily imagine high-traffic situations in which, in order to avoid these effects, it might pay to refuse some calls even though a slot is available. As usual, however, we assume this is not the case; instead, conditions are such that all calls should be accepted.

It is intuitively clear that calls of the same type should be packed contiguously when possible, and that states where they are so packed are preferable to those in which they are not. But in many cases it is not obvious which of two states (containing the same number of calls of each type) one would rather be in. Fig. 6 shows six pairs of states; in each case, which of the two is better? Or are they equally good?

In fact the quasi-ordering of Theorem 6.1 answers that question. For the two leftmost pairs, the upper state will turn out to be strictly preferable; thus, for example, if there is only one call in the system and a second of the same type arrives, it must be placed in an adjacent slot. In the third and fourth pairs, the states are equipotent.

The two rightmost pairs are more interesting. The state with three separated slots occupied looks worse than the one with two adjacent and a third separated, since the latter has room for a call of the second type. But the wrong departure can leave the latter state with oppositely opposed calls, which we have decided is the worst state of that type; this cannot happen to the former state.

In fact in each of those pairs the two states are incomparable. What

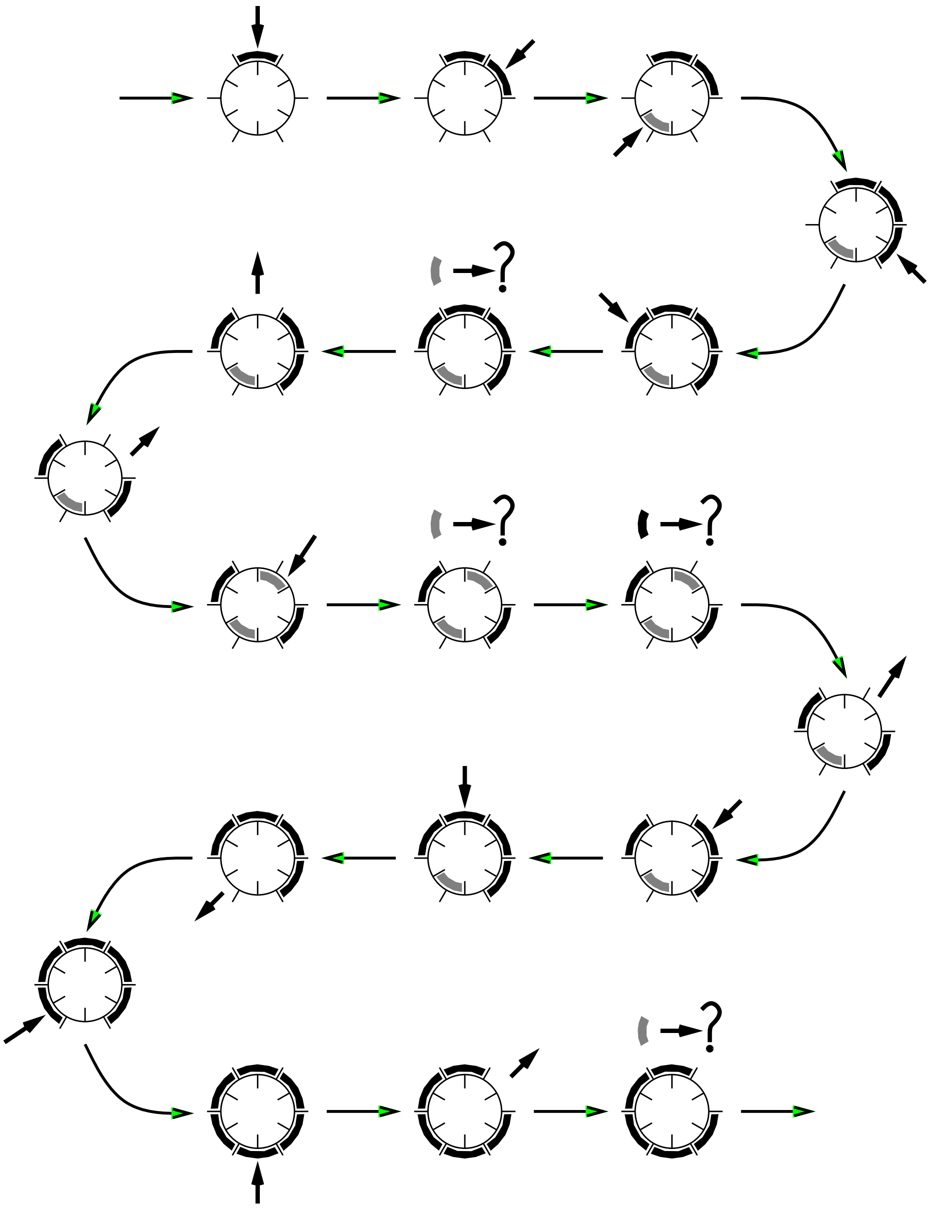

Figure 5: A possible state sequence in the TDMA problem

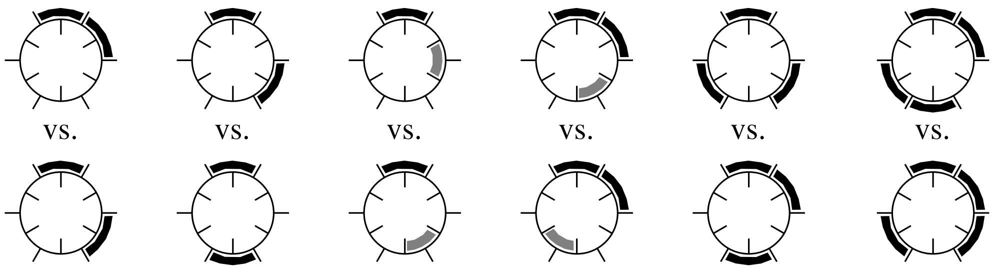

Figure 6: Six pairs of states: from each pair, which state is preferred?

saves the proof is that when placing an arrival, the assignment algorithm never has to decide between two incomparable states; there is always a third alternative which is better than both.

The full quasi-ordering is shown in Fig. 7, for all the state classes containing more than one element. Condition (i) of Theorem 6.1 is easily checked from this figure. From Fig. 7 one can also deduce which choices must be made to by an optimal algorithm in order to satisfy condition (ii); we have identified one such algorithm, which we call **B**, and chosen bijections which satisfy condition (iii).

In Fig. 8 all 26 (equivalence classes of) states are represented. From each state which can accept a type-1 (outer) call, a heavy solid arrow departs to the new state which results when algorithm **B** makes its assignment. Similarly, from each state which can accept a type-2 (inner) call, a heavy dashed arrow indicates **B**'s choice.

The light arrows indicate departures; from each state S there will be one light solid arrow for each departure of a type-1 call, and one light dashed arrow for each departure of a type-2 call.

Conditions (ii) and (iii) for Theorem 6.1 can be checked on the figure. For example, to check condition (ii) for a given state T, it must be ascertained that the set of states achievable via arrival of a type-1 call from state T or any lesser state $S \leq T$ has a maximum state in the quasi-ordering, and that the heavy solid arrow from T points to that state. For condition (iii), if $S \leq T$ then there must be a one-to-one correspondence between out-going light solid arrows which preserves the quasi-ordering.

All these conditions are in fact satisfied. Thus, assuming memoryless call departure, and assuming that the traffic distribution, rejection costs

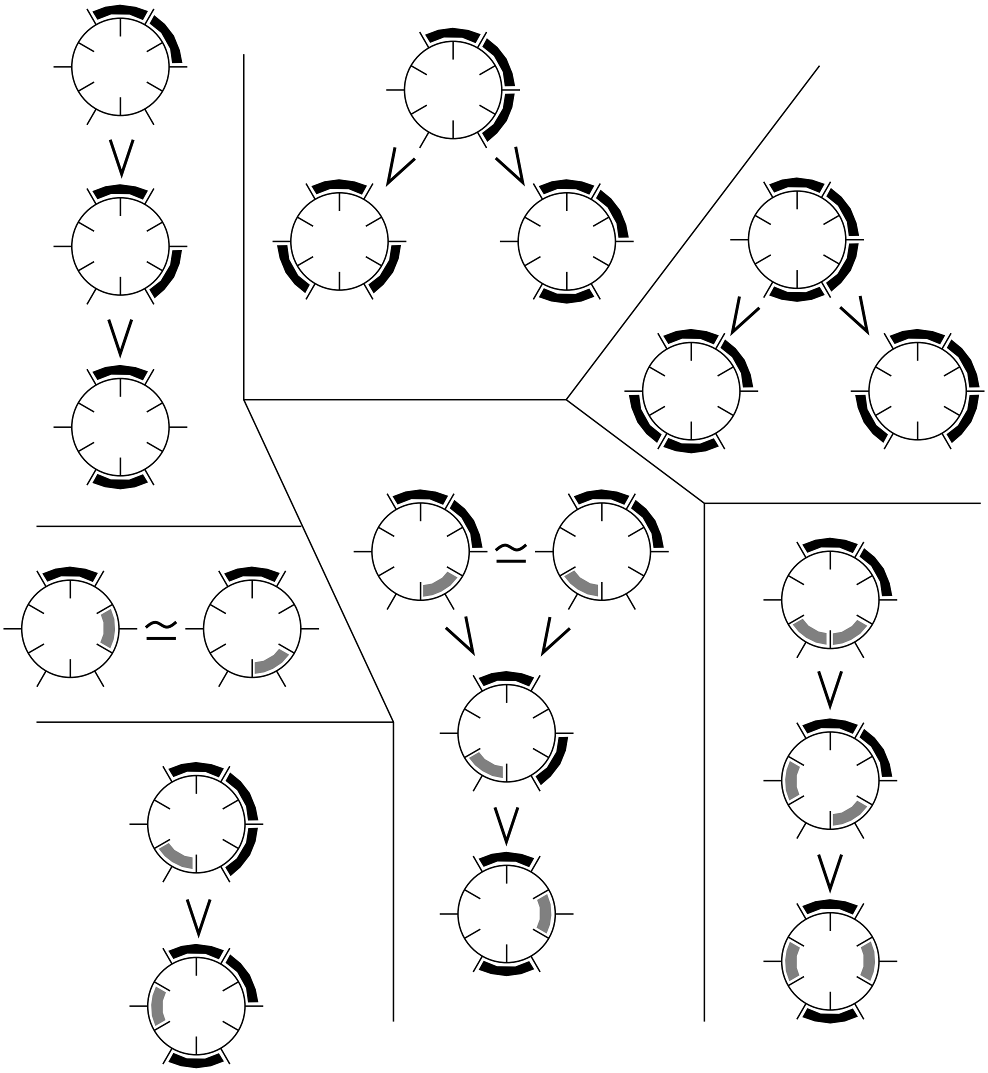

Figure 7: The quasi-ordering in classes $S(2,0)$, $S(3,0)$, $S(4,0)$, $S(1,1)$, $S(2,1)$, $S(2,2)$, and $S(3,1)$

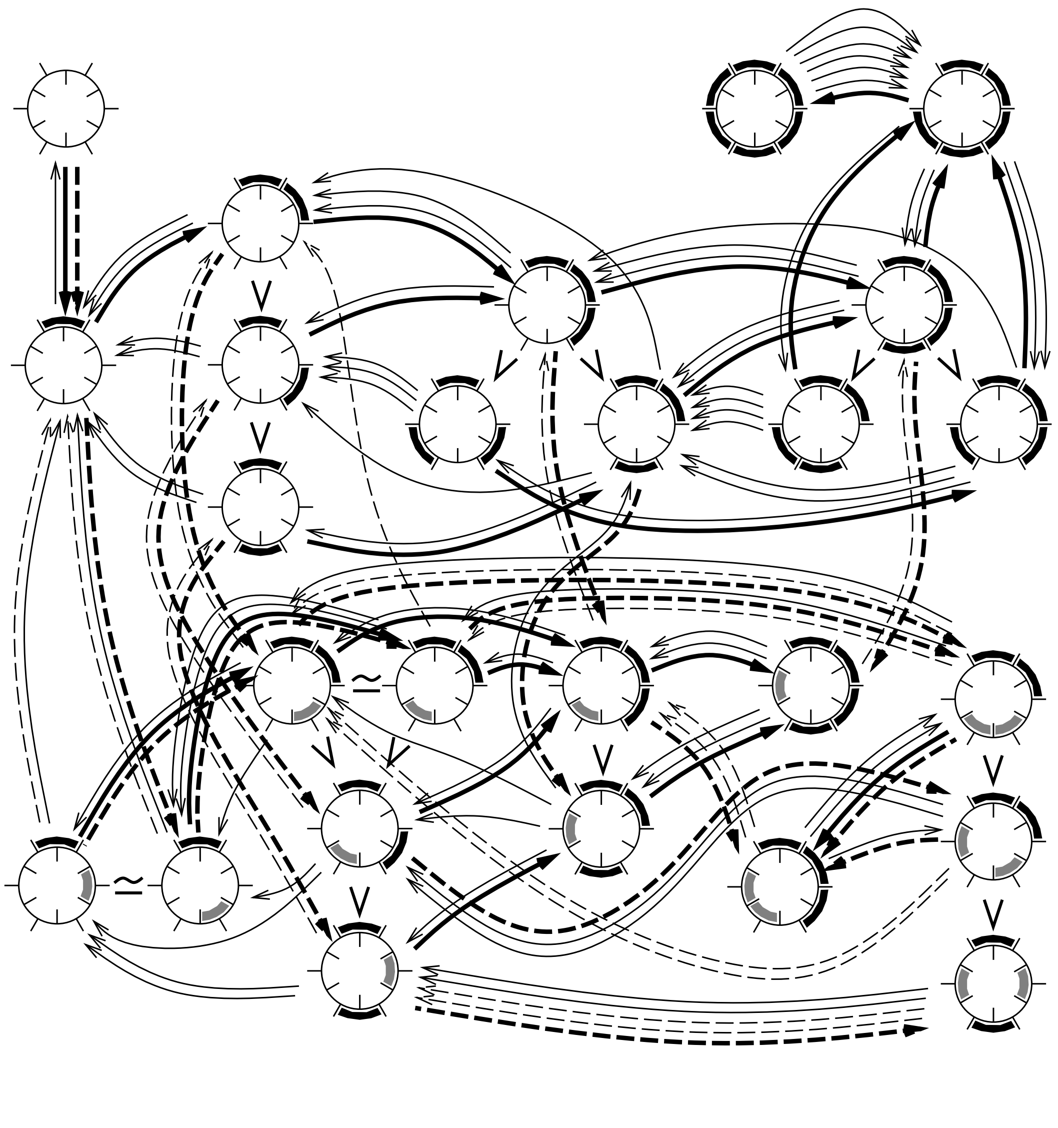

Figure 8: The complete state transition diagram for algorithm **B**

and mean departure times are such that all optimal assignment algorithms are greedy, we have:

Theorem 7.1. *The algorithm* **B** *pictured in Fig. 8 is optimal for the TDMA problem, with* $n = 6$.

8 Final Remarks

It is apparent that the method presented in this work is not applicable to every dynamic allocation problem, by a long shot. Memoryless departure times and the "greediness assumption" are essential, and even then there is no guarantee that an algorithm and quasi-ordering satisfying the conditions of Theorem 6.1 exist.

Indeed, for the TDMA problem with $n > 6$, the method cannot be applied directly. However, results can be salvaged by making additional assumptions e.g. about the superiority of one state over another. Indeed, any dynamic allocation problem can be narrowed for the purposes of analysis by making assumptions of the form "no optimal algorithm takes action x when faced with state S".

One particular approach which the authors of [3] found useful is based on the idea that the greediness assumption tends to become more reasonable as arrival traffic decreases. Hence, it is consistent to base new preference decisions between states on performance in low traffic conditions. Even in cases where Theorem 6.1 is far from being applicable, it may be possible to use a form of the method of this paper to show that a particular assignment algorithm is optimal in sufficiently low traffic.

Acknowledgment. The author is indebted to Dana Randall for a clever simplification of the majorization argument, to Kavita Ramanan for helpful discussions about random processes, and to Bell Labs colleagues Barnet Schmidt and Ed Coffman for bringing the two pretty dynamic allocation problems to the author's attention.

References

[1] Y. Azar and L. Epstein, On-line load balancing of temporary tasks on identical machines, *ISTCS '97.*

[2] G. Birkhoff and T.C. Bartee, *Modern Applied Algebra*, McGraw-Hill, New York (1970).

[3] S. Borst, E.G. Coffman, E. Gilbert, P. Whiting and P. Winkler, Optimal carrier sharing in wireless TDMA, *J. Interconnection Networks* **2** #2 (2001), 189–211.

[4] J. Boyer and K.S. Larsen, The seat reservation problem, *Algorithmica* **25** #4 (1999), 403–417.

[5] E.G. Coffman, Jr., An introduction to combinatorial models of dynamic storage allocation, *SIAM Review* **25** #3 (July 1983), 311–325.

[6] E.G. Coffman, Jr., T.T. Kadota and L.A. Shepp, A stochastic model of fragmentation in dynamic storage allocation, *SIAM J. Comput.* **14** #2 (May 1985), 416–425.

[7] E.G. Coffman, Jr. and F.T. Leighton, A provably efficient algorithm for dynamic storage allocation, *J. Comp. Syst. Sci.* **38** #1 (February 1989), 2–35.

[8] E.G. Coffman, Jr., M.R. Garey and D.S. Johnson, Approximation algorithms for bin packing: A survey, in *Approximation Algorithms for NP-hard Problems*, Dorit S. Hochbaum, ed., PWS Pub. Co., Boston MA (1997), 46–93.