Optimal Carrier Sharing in Wireless TDMA

S.C. Borst[*], E.G. Coffman, E.N. Gilbert, P.A. Whiting, and P.M. Winkler
Bell Labs, Lucent Technologies
700 Mountain Avenue
Murray Hill, NJ 07974
USA

June 19, 1999

Abstract

To improve efficiency, we allow a carrier in a TDMA (Time Division Multiple Access) network to be shared by adjacent cells. This sharing of time slots is seriously hampered by the lack of synchronization in distinct cells. We study packing algorithms that overcome this obstacle by clustering calls. The results suggest that even simple greedy algorithms are nearly optimal, and that little extra performance can be gained either by allowing the rejection of calls or by repacking.

1 Introduction

Consider a carrier (frequency band) allocated to a cell, say cell A, in a TDMA wireless network. The carrier is slotted and up to n calls can be time-multiplexed at one time, where n is given; values of 3 and 8 can be found in existing technologies [2, 3, 4]. When n calls are active, call requests are turned away; there is no queueing of call requests.

Typically, a carrier assigned to cell A will not also be assigned to a cell close enough to cell A to interfere with its calls. But this leads to an inefficient use of the carrier when the call traffic is low in cell A and idle time slots can be used by an adjacent cell in relatively high traffic. This paper studies a more efficient scheme whereby a neighboring cell, cell B say, is allowed to time-share the carrier with cell A. The sharing mechanism must account for the fact that the time slots in cells A and B are not perfectly synchronized.

[*]Current affiliation: Center for Mathematics and Computer Science, Amsterdam

To avoid the interference caused by overlapping time slots, the slots assigned to cell-A calls are not allowed to overlap slots assigned to cell-B calls.

To maximize the acceptance rate of calls, time-slot allocation should attempt to cluster separately the calls from cell A and those from cell B. For example, if the A calls in Figure 1 were clustered in adjacent slots, there would be 3 rather than 2 available slots for new B calls. This paper presents several clustering algorithms in Sections 2 and 3, and compares them in Section 4.

We distinguish classes of algorithms based on

1. the property that a new call must always be accepted if there is an available slot for it, and

2. the property that, in one or more states, calls in progress are repacked in new slots to make room for a new call.

Algorithms with the first property are said to be *greedy* and algorithms with the second property are said to be *repacking* algorithms. An algorithm in a given class is optimal if it maximizes the acceptance rate of calls among all algorithms in the class. This notion will be more precise after we define the stochastic model in which algorithms are compared. But first, we need some notation.

Whether the time slotting in cell A lags that in cell B or vice versa is not important, so a convenient graphic for the carrier state is the ring illustrated in Figure 2 for $n = 8$. The outside of the ring gives the time slots generated at the base station of cell A and the inside of the ring gives the time slots generated at the base station of cell B. These are called the A and B slots, respectively. The relative displacement has been fixed, as shown, at half a slot duration. This loses no generality, as the existence but not the extent of the overlap is significant. Segments shaded black and gray indicate the time slots assigned to calls in cells A and B, respectively. Any shading, such as any of those shown, that does not have overlapping slots carrying an A in the outer ring and a B in the inner ring, is a valid shading.

The linear representation of Figure 2 is obtained by moving clockwise around the ring and writing A's, B's, and dots as dark shadings, light shadings, and unoccupied half slots are encountered. The sequence is enclosed in angle brackets and the first character is an A, B, or dot according as the first half slot clockwise from the top of the ring overlaps an A call, overlaps a B call, or is unoccupied. The notation is cyclic in that the first and last positions are adjacent. The time slots assigned to an A and a B can not overlap, so in moving from an A to a B in either direction (cyclically) at least one intervening dot will be encountered. Since every A and B call occupies two half-slots, twice the total number of A's and B's plus the number of dots must be $2n$.

In our stochastic model, calls arrive in independent Poisson streams at rates λ_A and λ_B in cells A and B. Call interarrival and holding times are independent, the latter being

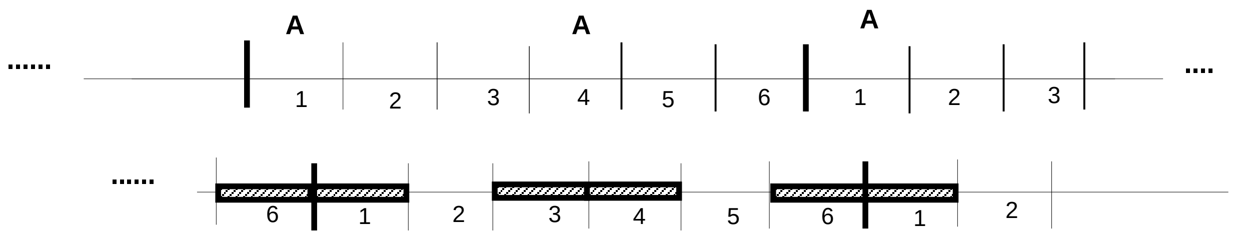
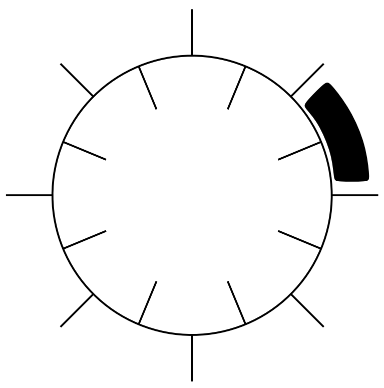


Figure 1: The nonoverlapping-slot constraint, $n = 6$. The upper and lower time lines apply to the time slotting at the base stations of cells A and B, respectively. The hatched intervals indicate the unused time slots in cell B that are not available to cell-B calls because of overlap with slots already assigned to cell-A calls.



independent exponentials with rate parameter μ in both cells. (Our results are easily extended to the case of different rates.) These assumptions yield a continuous-time, finite-state stochastic process Y on the set of carrier states defined above. A transition of the process occurs at an arrival or departure, the latter entailing the removal of a call from the current state. An arrival transition is determined by the call admission algorithm; the current state remains unchanged if the new call is not admitted; otherwise, the next state reflects the packing of the new call in some available slot of the current state. The admission algorithms studied here are defined by mappings from the current-state, new-call pair to an available slot, assuming the new call is to be admitted. Thus, because of the exponential assumptions, Y will be a Markov process for all of the algorithms presented in the next section.

By means of dynamic programming, optimal algorithms can be computed for given traffic parameters λ_A, λ_B, and μ; the class of algorithms can but need not be restricted to greedy algorithms or to algorithms not allowed to repack calls. Section 2 works out the details for the latter class of algorithms. These solutions are useful for comparisons, but they are unrealistic, since traffic parameters are not known in practice. Thus, the remainder of the paper focuses on greedy algorithms that make decisions not depending on offered traffic.

To see that greedy decisions are not always best, it is only necessary to consider the system with sufficiently heavy traffic in cell A. In these circumstances, it is optimal to reserve the carrier for the exclusive use of base station A; the unavoidable wasted time created by a state that mixes even one B with A's would decrease the overall call acceptance rate. Section 3 will give an example that helps to clarify the term 'sufficiently large traffic'.

In the class of greedy repacking algorithms, it is trivial to find an optimal algorithm that is insensitive to offered traffic; any algorithm that keeps the A calls and B calls in separate sequences of consecutive slots will do. Section 2 analyzes algorithms of this type. However, if greedy algorithms are not allowed to do repacking, then the problem of finding an optimal algorithm that makes decisions independent of the offered traffic is more difficult. Section 3 proposes two such clustering algorithms, one of first-fit type and the other of best-fit type, and compares them in Section 4 with the optimal algorithms of Section 2. The first-fit algorithm is denoted by FF_n and performs quite well even when compared to optimal repacking algorithms; and it has the advantage of simplicity. At the cost of somewhat more mechanism, the best-fit algorithm BF_n does better; indeed, it is optimal or very nearly optimal over a wide range of realistic values for the traffic parameters.

The paper concludes with two sections that bring out useful properties of BF_n. The first deals with symmetries. If a state x is carried into a state x' by rotations or reflections, then x and x' are said to be in the same *dihedral* class; such states are treated alike by the BF_n algorithm, i.e., BF packing decisions preserve dihedral symmetries.

For example, BF_8 packs an arriving A into the second A slot of $\langle A..AA.....B.\rangle$, so it will pack an arriving A into the gap between the A's in state $\langle .B.A..AA....\rangle$, which is obtained by a rotation, and into the corresponding gap of $\langle .B.....AA..A\rangle$ which is obtained by reflection. BF_n also preserves *exchange* symmetries; two states are in the same exchange class if one can be obtained from the other by interchanging A's and B's. Combining all three symmetries gives the *dihedral exchange* symmetry classes. Thus, we can analyze BF_n by analyzing the smaller Markov process $\hat{Y}$ whose states are the dihedral exchange classes of the states of Y. The size of $\hat{Y}$'s state space is computed in Section 5; explicit functions of n are obtained for the count of rotational symmetry classes. A procedure for enumerating representatives of the dihedral exchange classes is also discussed.

Section 6, the last section, proves that BF_n is optimal for all $n \leq 6$ under the condition that, in the class of algorithms not allowed to repack calls, all optimal algorithms are greedy. Section 6 also illustrates that if $n \geq 7$, then optimal decisions depend in general on the offered traffic. On the other hand, we verify in the case of $n = 8$ that there are only a few states where optimal decisions depend on offered traffic; this accounts for the very small differences found in Section 4 between the performance of BF_8 and that of an optimal algorithm.

2 Optimal Algorithms

This section first shows how to compute optimal algorithms from dynamic programs, and then analyzes optimal greedy repacking algorithms.

Dynamic programming. Our stochastic setup is standard for the application of dynamic programming, since the arrival streams are independent and Poisson, call holding times are independently and exponentially distributed, and there are a finite number of states and a finite action space. In the class of algorithms studied here, the process Y induces a Markov decision process in which non-null actions can be represented by the available slot of the current state where a new arrival is placed. The rate at which reward is accumulated in a given state x is given by $n(x)$, the total number of calls carried. For given traffic parameters, we want to determine an algorithm that maximizes this quantity.

In the usual way, to bring the problem into the theory based on a discrete time parameter, we replace Y by a uniformized version Y^* in which events take place at a constant rate. It is convenient to take this rate $\nu = \nu(n)$ as the rate in Y maximized over all states. The maximum departure rate in any state is $n\mu$, so $\nu = \lambda_A + \lambda_B + n\mu$. In order that the relative rates of arrivals and departures remain unchanged, we introduce an artificial, or dummy, event in Y^* as follows. In a state with j active calls, the next event in Y^* is an arrival in cell A, an arrival in cell B, a departure from an occupied slot, or a dummy event which may be associated with a departure from some unoccupied slot. The probabilities of these events are, in the order given, λ_A/ν, λ_B/ν, $j\mu/\nu$, and $(n-j)\mu/\nu = 1-(\lambda_A+\lambda_B+j\mu)/\nu$.

Now let $\mathcal{X}$ denote the set of feasible states defined in Section 1, i.e., those states in which A and B calls are separated by at least one dot and are assigned to slots $(i, i+1)$ with i odd and even, respectively. Let $\mathcal{A}_x$ and $\mathcal{B}_x$ denote the sets of states that can be reached from x in a single cell-A and cell-B arrival event, respectively. To this point, we have tacitly assumed that the class of policies of interest can contain nongreedy policies. To limit the optimization to the subset of greedy policies, we simply require that $\mathcal{A}_x$ not contain x whenever x has a slot available for an A call. If x is a blocking state for A calls, then x is the only state in $\mathcal{A}_x$. Similar restrictions apply to $\mathcal{B}_x$.

Finally, let $\mathcal{D}_x$ contain just those states which can be reached from x via a single departure from either cell. For a given offered-traffic pair, the maximum steady-state reward rate is achieved by an algorithm satisfying the Bellman equation,

$$\gamma + f(x) = n(x) + \frac{\lambda_A}{\nu}\max_{y\in\mathcal{A}_x} f(y) + \frac{\lambda_B}{\nu}\max_{z\in\mathcal{B}_x} f(z) + \frac{\mu}{\nu}\sum_{w\in\mathcal{D}_x} f(w) + \frac{\nu-\lambda_A-\lambda_B-n(x)\mu}{\nu} f(x), \tag{1}$$

where $n(x)$ is the number of calls in state x, γ is the long-term average reward rate, and $f(x)$ denotes the usual differential cost or potential associated with state x [8]. As shown in [1], the decisions of an optimal algorithm satisfying (1) can be determined by solving the recursion,

$$\begin{aligned} F^{(k+1)}(x) = n(x) + \frac{\lambda_A}{\nu}\max_{y\in\mathcal{A}_x} f^{(k)}(y) \quad &+ \quad \frac{\lambda_B}{\nu}\max_{z\in\mathcal{B}_x} f^{(k)}(z) \\ &+ \quad \frac{\mu}{\nu}\sum_{w\in\mathcal{D}_x} f^{(k)}(w) + \frac{\nu-\lambda_A-\lambda_B-n(x)\mu}{\nu} f^{(k)}(x) \end{aligned} \tag{2}$$

with $f^{(k+1)}(x) = F^{(k+1)}(x) - F^{(k+1)}(0)$ and $f^{(0)}(x) \equiv 0$ for all states x. We have that $F^{(k)}(0) \to \gamma$ and $f^{(k)}(x) \to f(x)$ as $k \to \infty$. The application of (2) to specific cases is illustrated in Section 4.

Repacking algorithm. Consider the greedy repacking algorithms mentioned in Section 1 which maintain the A and B calls in separate consecutive sequences of A and B slots. An analysis of any such algorithm leads to explicit results. Let $a(t), b(t)$ be the respective numbers of A and B calls in the system at time t. We note immediately that $\{(a(t), b(t)), t \geq 0\}$ is a Markov process on the set of states $\{(0,n),(n,0)\} \cup \{(i,j) : 0 \leq i,j; i+j \leq n-1\}$. Were it not for the states $(0,n),(n,0)$, the process would be the classical one of an Erlang queue with two classes of customers, a process that has local balance, as is easily verified. In spite of the added peculiarity of states $(0, n), (n, 0)$, the process continues to have local balance and a product form stationary distribution

$$q(i,j) := \lim_{t\to\infty} \Pr\{(a(t), b(t)) = (i,j)\}$$

If we let $\rho_A = \lambda_A/\mu$ and $\rho_B = \lambda_B/\mu$ denote the cell-A and cell-B traffic intensities, then

$$q(i,j) = G^{-1}\frac{\rho_A^i}{i!}\frac{\rho_B^j}{j!}, \tag{3}$$

where the normalization constant $G = \sum_{i,j}\frac{\rho_A^i}{i!}\frac{\rho_B^j}{j!}$ is a sum over all states. Blocking probabilities are computed over the states

- $(n-1, 0)$ which blocks only B calls,
- $(0, n-1)$ which blocks only A calls, and
- $(n, 0), (n-2, 1), \ldots (1, n-2), (0, n)$, which block both A and B calls.

Thus, the nonblocking states are just those states (i,j) with $i + j < n - 1$.

3 First Fit and Best Fit

Under FF_n, a call arriving at a time when the current state has no slot available to the call's cell is rejected. Otherwise, with reference to Figure 2, FF_n packs the call in the first available slot found in a scan of the current state clockwise from the topmost A slot, if the call is in cell A, and counterclockwise from the topmost B slot, if the call is in cell B.

The best-fit algorithm BF_n is chosen from the class of *weighting algorithms* defined as follows. First, we augment the state so that each slot carries a weight. According to a weighting algorithm, a new call is packed in a slot of the same type with maximum non-negative weight, if one exists; otherwise, if all slots of the same type as the new call have negative weights, the call is rejected.

A specific weighting algorithm is defined by two weight-increment vectors $(\delta_0, \ldots, \delta_{\lfloor n/2\rfloor})$ and $(\delta_1', \ldots, \delta_{\lfloor n/2\rfloor}')$. Weights are changed at arrival and departure events as follows. Suppose we have just assigned a new A call to an available slot; the case for B calls is entirely symmetric. The weight of the slot where the new call is assigned is incremented by δ_0, the slots to each side of that slot have their weights incremented by δ_1, the slots at distance 2 have their weights incremented by δ_2, and so on with the slots a distance $\lfloor n/2\rfloor$ incremented by $\delta_{\lfloor n/2\rfloor}$. If n is even, the slots encountered going a distance $\lfloor n/2\rfloor$ clockwise and counterclockwise are one and the same antipodal slot.

The two B slots straddled by the A slot just assigned have their weights incremented by δ_1'; the weights of the B slots at equal distance are then incremented by $\delta_2', \ldots, \delta_{\lfloor n/2 \rfloor}'$ as one moves away from the new A call. Finally, whenever a call departs, the increments made at the call's arrival time become decrements of the same magnitude.

The system can be set up to begin with fractional weights associated with its slots, which we assume are all initially empty. These weights will break ties in symmetric situations. However, without these fractional values (all weights are initially 0), the increments are designed so that there will never be a tie between two slots of the same type unless those slots are in the same dihedral class. Thus, the assignment algorithm preserves symmetry classes.

Since weighting algorithms are greedy, the increment vectors must be such that in every reachable state with an available slot, the weight of some available slot must be non-negative. Moreover, there can be no reachable state in which the slot with maximum non-negative weight is unavailable. An example for $n = 8$ that we shall analyze in detail has

$$\delta_0 = -41, \quad \delta_1 = 12, \quad \delta_2 = 6, \quad \delta_3 = 2, \quad \delta_4 = 0$$

$$\delta_1' = -41, \quad \delta_2' = 0, \quad \delta_3' = 2, \quad \delta_4' = 3$$

Figure 3 illustrates BF_8 in action with a four-state sample path in which there are two arrivals and a departure. Note that a δ_i sequence decreasing for $i \geq 1$ and a δ_i' sequence increasing for $i \geq 1$ promote clustering and an antipodal positioning of the clusters.

As A's, say, build up to the left and right of an empty slot, the slot's weight can increase by at most $2 \cdot 12 + 2 \cdot 6 + 2 \cdot 2 = 40$, so setting $\delta_0 = \delta_1' = -41$ guarantees that an unavailable slot must always have negative weight. Section 4 will show that the algorithm always makes a reasonable choice for the slot to be assigned a new call. In most cases, it is obvious which slot (or equivalence class of slots) should be selected, and in those cases the algorithm chooses correctly. In a few states the correct slot choice depends on traffic intensities, but in all such states, the algorithm opts to make a large contiguous block of ongoing calls of the same type, which is reasonable and consistent behavior even under those circumstances when it may not be optimal.

For example, in the state $\langle .AA..A..A.B \rangle$ it is unclear whether an arriving A call should be placed in the slot to the left or right of the isolated A (a calculation shows that these slots have weights 32 and 26, respectively). Similarly, when in the state $\langle ....AAA.B.A \rangle$ the question is whether to put a new A call into the available slot (of weight 29) next to the string of 3 A's or into the available slot (of weight 22) next to the remaining A. The dynamic programming solution can be used to investigate these states. For example, a computation based on (2) shows that, if traffic is equal and there is an A call arriving to the state $\langle .AA..A..A.B \rangle$, then adding the call to the left of the

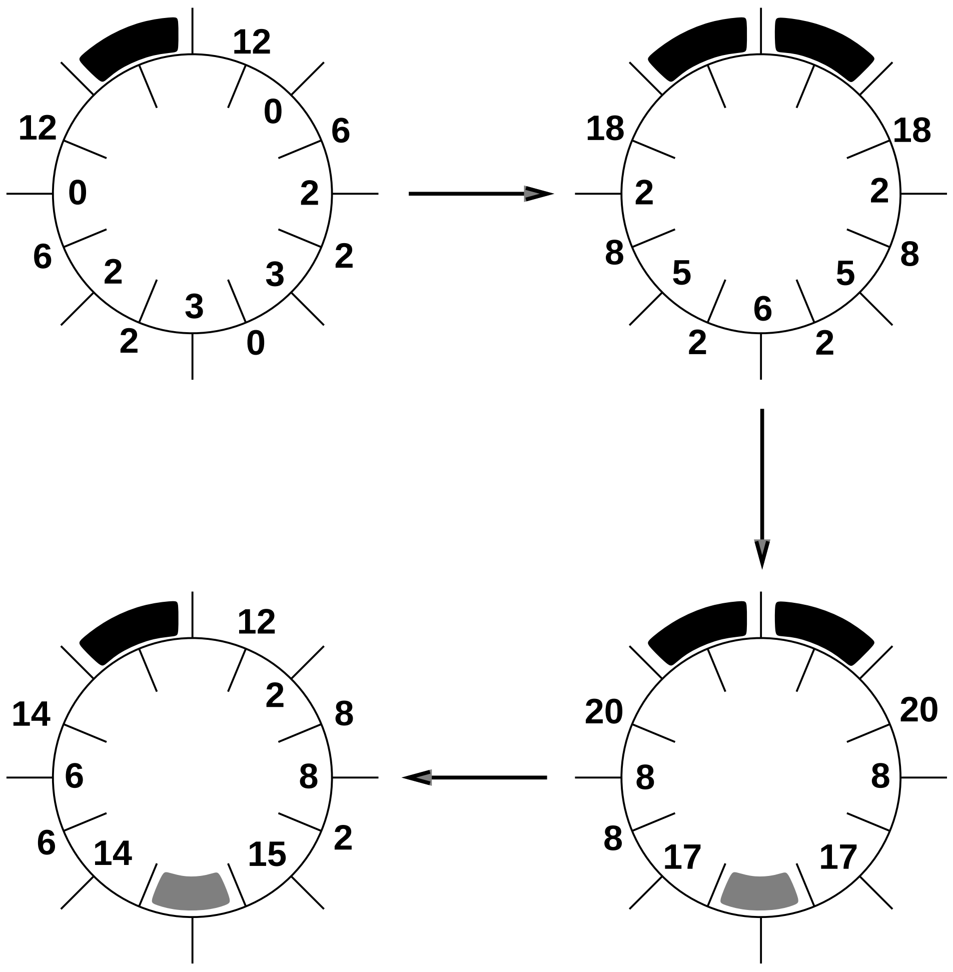


Figure 3: A sample path under algorithm BF_8. The negative weights of unavailable slots are not shown.

third A is optimal for $\rho_A = \rho_B \geq 2.4$, approximately, and adding it to the right is optimal otherwise.

4 Comparisons

For our purposes, i.e., for small n, numerical comparisons of the various algorithms can be obtained by computational as well as simulation methods. Computations are based on routine numerical solutions of finite Markov chains and are only feasible for n up to about 12. Simulations were easily implemented and gave insights into transient behavior as well as the stationary regime. The loss in accuracy was small enough to be of no concern, so we used simulation results except in those cases where explicit formulas were available. However, we did use numerical solutions of Markov chains to verify the (probable) correctness of our simulation program for algorithm BF_8; the results of the computations matched very closely the simulation results (in particular, those in Table 1 below).

Our first comparison is between FF_n and BF_n for $n = 8$. The simulation results appear in Table 1, where p_A, p_B are the blocking probabilities for cell-A and cell-B calls, respectively; the overall fraction of lost calls is then

$$p = \frac{\lambda_A p_A + \lambda_B p_B}{\lambda_A + \lambda_B}.$$

Results are given for all integer traffic intensities summing to 6, and range from about 1 rejection out of every 200 calls to about 1 out of every 5.

As expected, uniformly over all parameter values, BF_8 gives lower overall lost traffic than does FF_8. Note, however, that the same can *not* be said of the individual blocking probabilities p_A and p_B, when the difference between traffic intensities is sufficiently large. In these cases, the lower overall blocking probability is being achieved at the expense of larger blocking probabilities for the cell with higher traffic. On the other hand, it can also be seen that the differences in all cases are rather small.

The results from the dynamic programming algorithm DP_8 are for the greedy case with repacking disallowed, and indicate that BF_8 is for all practical purposes optimal for the cases tabulated. The very slight discrepancies between the DP_8 and BF_8 result from variability in the simulation results. The close agreement between the two reflects the noncritical nature of the choice in the few cases where the choice for BF_8 is unclear. We also tested the benefits of nongreedy decisions; the results for the dynamic program in which call rejection was allowed gave very slight improvements over the greedy case.

A property supported by our numerical studies, one that could have been anticipated, is that for a fixed total intensity $\rho_A + \rho_B$, the lost traffic decreases as the difference

Table 1: Blocking probabilities for FF_8, BF_8 and DP_n.

		Algorithm FF_8			Algorithm BF_8			DP_8		
ρ_A	ρ_B	p_A	p_B	p	p_A	p_B	p	p_A	p_B	p
1	1	.0048	.0047	.0048	.0043	.0042	.0043	.0045	.0045	.0045
1	2	.0353	.0234	.0274	.0333	.0226	.0262	.0319	.0227	.0257
1	3	.1060	.0572	.0694	.0950	.0574	.0668	.0954	.0567	.0664
1	4	.2107	.1003	.1224	.1871	.1012	.1184	.1890	.1010	.1186
1	5	.3233	.1473	.1766	.2978	.1493	.1741	.2968	.1495	.1740
2	2	.0772	.0784	.0778	.0743	.0744	.0744	.0737	.0737	.0737
2	3	.1605	.1297	.1420	.1495	.1250	.1348	.1502	.1242	.1346
2	4	.2581	.1771	.2041	.2411	.1753	.1972	.2408	.1755	.1972
3	3	.2103	.2113	.2108	.2028	.2034	.2031	.2034	.2034	.2034

$|\rho_A - \rho_B|$ increases, i.e., as more and more traffic is being concentrated in one of the cells. (For example, see the (1,5), (2,4), and (3,3) rows of Table 1.)

Algorithm BF_8 and the optimal greedy repacking algorithm R_8 are compared in Table 2, where the results for R_8 are computed from (3) applied to the appropriate aggregates of blocking states. The table verifies that, in terms of the overall blocking probability, R_8 is uniformly better than BF_8. On the other hand, it can be seen that the table shows the same effects that appeared in Table 1: When the difference in the traffic intensities is large enough, the lower overall blocking probability is balanced by a greater blocking probability for the calls in the cell with higher traffic. And for a fixed total traffic intensity, the equal-intensity case $\rho_A = \rho_B$ maximizes the overall blocking probability.

We observed earlier that, with traffic intensities sufficiently large, a non-sharing algorithm (i.e., non-greedy algorithm confined to the calls of just one of the cells) becomes optimal. To get some idea of what 'sufficiently large' means, we computed the overall blocking probability p in the equal-intensity case and found that freezing out one of the cells completely gives lower p only for $\rho_A = \rho_B \geq 10$. But in this case, over 2/3 of the calls are rejected on average. In a regime where ρ_A and ρ_B are nearly the same, only non-sharing or greedy sharing algorithms are likely to be of practical interest. With the two loads very different, non-greedy algorithms might be superior to non-sharing algorithms, but probably not by much.

Figure 4 brings out relative performance graphically for $n = 8$; the curves are for the balanced case $\rho_A = \rho_B$, but they are typical of other choices as well. A curve for DP_8 is not shown, as it is not distinguishable from the BF curve. Note that the repacking algorithm gives only marginal improvements. We conclude that other issues, e.g., ease of

Table 2: Blocking probabilities for R_8 and BF_8

		Algorithm R_8			Algorithm BF_8		
ρ_A	ρ_B	p_A	p_B	p	p_A	p_B	p
1	1	.0034	.0034	.0034	.0043	.0042	.0043
1	2	.0221	.0209	.0213	.0333	.0226	.0262
1	3	.0657	.0573	.0596	.0950	.0574	.0668
1	4	.1315	.1065	.1115	.1871	.1012	.1184
1	5	.2105	.1605	.1688	.2978	.1493	.1741
2	2	.0625	.0625	.0625	.0743	.0744	.0744
2	3	.1215	.1183	.1196	.1495	.1250	.1348
2	4	.1894	.1787	.1823	.2411	.1753	.1972
3	3	.1845	.1845	.1845	.2028	.2034	.2031

implementation, will likely determine whether repacking algorithms are preferable, and if not, whether the better performance of BF_8 relative to FF_8 is worth the additional mechanism.

The U curve for unrestricted carrier sharing (no adjacency constraints among the A's and B's in a state) shows the potential performance gains under perfect synchronization.

5 State Space

This section begins by counting states and then counts their symmetry classes; the procedure will be useful in the enumeration of symmetry classes that we require in the optimality proof of Section 6. Further motivation is provided by the fact that, for the analysis of symmetry preserving algorithms like BF_6 and BF_8, the Markov process Y can be reduced to a Markov process $\hat{Y}$ on the set of dihedral exchange symmetry classes, thus giving a substantial reduction in complexity, as this section shows. General background for the combinatorics below can be found in Riordan [5] and Stanley [7].

The number S_n of states has an interesting relation to the Fibonacci numbers. Although S_n can be computed in various ways, we choose a technique that brings out this relation explicitly.

Theorem 1 *Let F_m denote the m^{th} Fibonacci number, where $F_0 = 0$, $F_1 = 1$, and $F_{m+2} = F_{m+1} + F_m$, $m \geq 0$. Then*

$$S_n = F_{2n+1} + F_{2n-1},\ n \geq 1.$$

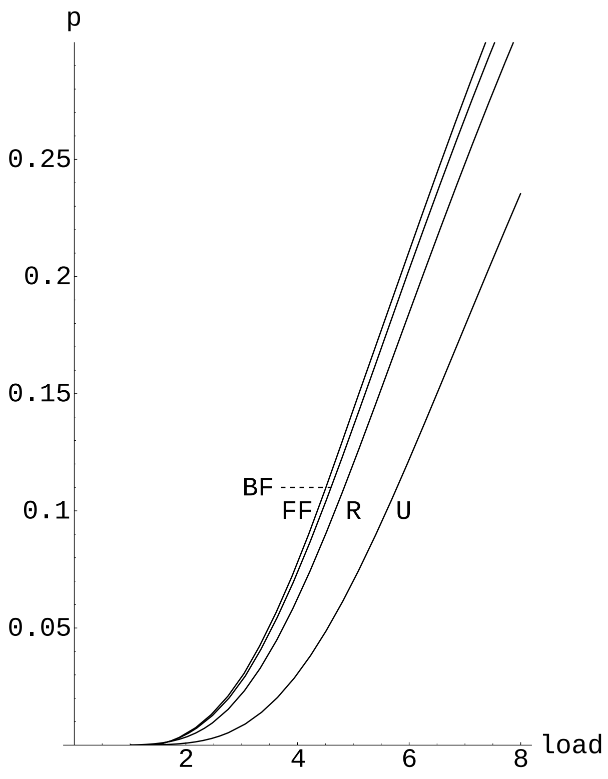


Figure 4: Blocking probabilities p vs. load $\rho = \rho_A + \rho_B$ for the case $\rho_A = \rho_B$ ($n = 8$). The curve labels denote the FF = FF_8 and BF = BF_8 clustering algorithms, the R = R_8 repacking algorithm, and the unrestricted sharing algorithm U = U_8 which operates free from any adjacency constraints. Under U_n, each of the 2^n possible length-n sequences of A's and B's is a valid state, and the system becomes a two-class Erlang queue.

Proof: We start by considering linear arrangements of half-slots numbered $\ldots, 0, 1, 2, \ldots$, where A's and B's can be inserted into pairs of half-slots of the form (odd,even) and (even,odd), respectively. Thus, half-slot 1 is either empty or it contains the right half of a B or the left half of an A. Suppose that half-slot 1 does not contain the right half of a B. Under this condition, S'_m counts the number of possible arrangements in half-slots $2, 3 \ldots, m$ with half-slot m either empty or containing the right half of a packed call. The packed call is an A call if m is even and a B call if $m > 1$ is odd. A single half-slot can only be empty so $S'_1 = 1$. Next, $S'_2 = 2$, since a single full slot ($m = 2$) can only be an A call or two empty half-slots. For larger m, there are S'_{m-1} ways that half-slot m can be empty and S'_{m-2} ways that it can be the right half-slot of a call. Then

$$S'_m = S'_{m-1} + S'_{m-2}$$

from which we conclude that S'_m is the $(m+1)^{st}$ Fibonacci number.

There are F_{2n+1} linear arrangements of half slots $1, \ldots, 2n$ that end in an empty half-slot or the right half of a packed call, necessarily an A call since $2n$ is even. Each of these arrangements can be folded into a cycle of $2n$ slots with the contents of half-slot $2n$ compatible with the contents of half-slot 1. Thus, it remains to determine S''_m, the number of arrangements with the right half of a B in half-slot 1, since

$$S_n = S'_{2n} + S''_{2n} = F_{2n+1} + S''_{2n} \ .$$

But consider the F_{2n-1} states with an A call in slot $(1,2)$ that we just counted. Rotate each a half slot and exchange A's with B's and we get the $S''_{2n} = F_{2n-1}$ states with a B in slot $(2n, 1)$. Then $S''_{2n} = F_{2n-1}$ and we are done. ∎

Table 3 gives the first 12 values of S_n.

n:	1	2	3	4	5	6	7	8	9	10	11	12
S_n:	3	7	18	47	123	322	843	2207	5778	15127	39603	103682

Table 3: Number of states.

We now derive the number T_n of rotational symmetry classes of states for a ring of n slots. The clockwise rotations of a ring of n slots form the cyclic group of order n. Since n rotations transform a given state into states of the same class, we might expect classes to contain n states and T_n to be S_n/n. That would not be quite right because some classes have fewer than n members. Thus, with $n = 8$, the state $x = \langle A.B.AA.B.A \rangle$ has "period" 4, i.e., a rotation through 4 slots carries x back to itself; x belongs to a class of only 4 states. Polya's Theorem (Riordan [5, Chap. 6]), as applied to rotational symmetries, can be expressed in the form

$$T_n = \frac{1}{n} \sum_{k=1}^{n} I_k \tag{4}$$

where I_k is the number of states that a clockwise rotation by k slots leaves unchanged. These unchanged states are the states that have a period $d = gcd(n, k)$, i.e., states of the form $x = x'x' \cdots x'$ repeating n/d copies of a state x' for the ring of size d. Then (4) is

$$T_n = \frac{1}{n} \sum_{k=1}^{n} S_{\text{gcd}(n,k)}. \tag{5}$$

For any divisor d of n, the k-th term of (5) is S_d for each k with $gcd(n,k) = d$. There are $\phi(n/d)$ such terms, where $\phi(i) := |\{j \leq i : gcd(i,j) = 1\}|$ is Euler's totient function. Collecting terms S_d of (5), one finds

$$T_n = \frac{1}{n} \sum_{d|n} \phi(n/d) S_d. \tag{6}$$

For example, if $n = 8$, then (6) has 4 terms and

$$T_8 = (4 \cdot 3 + 2 \cdot 7 + 1 \cdot 47 + 1 \cdot 2207)/8 = 285.$$

To find representative states belonging to T_n distinct classes, one may start with the classes of states that contain no A's and from 0 to n B's. There are 2^n such states. Counting their classes is equivalent to counting the rotational classes of necklaces having beads of two colors, white for empty slots and black for occupied slots. A derivation like the one for (6) leads to a formula for the number of classes of necklaces with n beads (Riordan [5, p. 162]). It differs from (6) only in having S_d replaced by 2^d. For $n = 8$ there are only 36 classes of states with no A's. Thirty six representatives are easily found. Next, add single A's to the 36 representatives with no A's and select inequivalent states to represent the classes with one A. Continuing to add single A's to the representatives already found, one eventually finds all classes. In this procedure one needs never examine states with more A's than B's; their classes can be obtained from earlier classes by interchanging A's and B's. At each stage, there are only a few ways to add an A; T_n representatives are found by testing much fewer than S_n states. Formula (6) gives a check that all classes were found.

Once the rotational classes are found, it is not hard to find the dihedral classes. The new symmetry (reflection) will combine certain pairs of classes into one dihedral class. Pick any state in rotational class $\mathcal{C}$ and reflect (reverse) it. If the reflected state belongs to a new rotational class $\mathcal{C}'$ then the union of $\mathcal{C}$ and $\mathcal{C}'$ is a dihedral class; otherwise, $\mathcal{C}$ is itself a dihedral class. A detailed analysis (by computer program) for $n = 8$ gives the count in Table 4 as a function of the numbers of A's and B's. The sum total of the table entries gives 187 dihedral classes.

Finally, we count the *dihedral exchange* symmetry classes where interchanging A's and B's is added as a third symmetry operation. Pick a state in a dihedral class $\mathcal{C}$ and exchange A's and B's. If this gives a state in a different dihedral class $\mathcal{C}'$ then $\mathcal{C}$ and $\mathcal{C}'$ are merged into a single dihedral exchange class; otherwise, $\mathcal{C}$ is already a dihedral

Table 4: Numbers of dihedral classes of states for $n = 8$; the classes counted in the $(i,j)^{th}$ entry have states with i A's and j B's.

1	1	4	5	8	5	4	1	1
1	3	9	10	9	3	1	0	0
4	9	17	12	7	0	0	0	0
5	10	12	4	1	0	0	0	0
8	9	7	1	0	0	0	0	0
5	3	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0

exchange class. Clearly, dihedral classes that are also dihedral exchange classes have equal numbers of A's and B's, and patterns of A's and B's that are similar. For $n = 8$ these are easily identified as the 11 classes having the representatives:

$$\langle................\rangle \quad \langle A............B\rangle \quad \langle A..........B..\rangle \quad \langle A........B....\rangle$$
$$\langle....AA.BB...\rangle \quad \langle..AA...BB...\rangle \quad \langle A...A.B...B.\rangle \quad \langle..A.B.A.B...\rangle$$
$$\langle A.B...A.B...\rangle \quad \langle AAA.BBB...\rangle \quad \langle AA.B.A.BB.\rangle$$

Then the number of dihedral exchange classes is $(187 - 11)/2 + 11 = 99$.

6 Optimality of BF_n

The dynamic program of Section 2 maximized the stationary acceptance rate of calls. Here we can strengthen the notion of optimality by removing the restriction to a stationary process: Algorithm V is optimal if, for any given initial state, V maximizes the expected number of accepted calls in $[0,t]$ for all $t \geq 0$.

The best-fit algorithm BF_6 is defined by the vectors

$$\delta_0 = -7,\ \delta_1 = 2,\ \delta_2 = 1,\ \delta_3 = 0,$$

and

$$\delta_1' = -7,\ \delta_1' = 0,\ \delta_2' = 1.$$

Figure 5 shows the transition function under BF_6; every state that can accept an A (B) call has a heavy solid (dashed) arrow exiting from it, and it has a light solid (dashed)

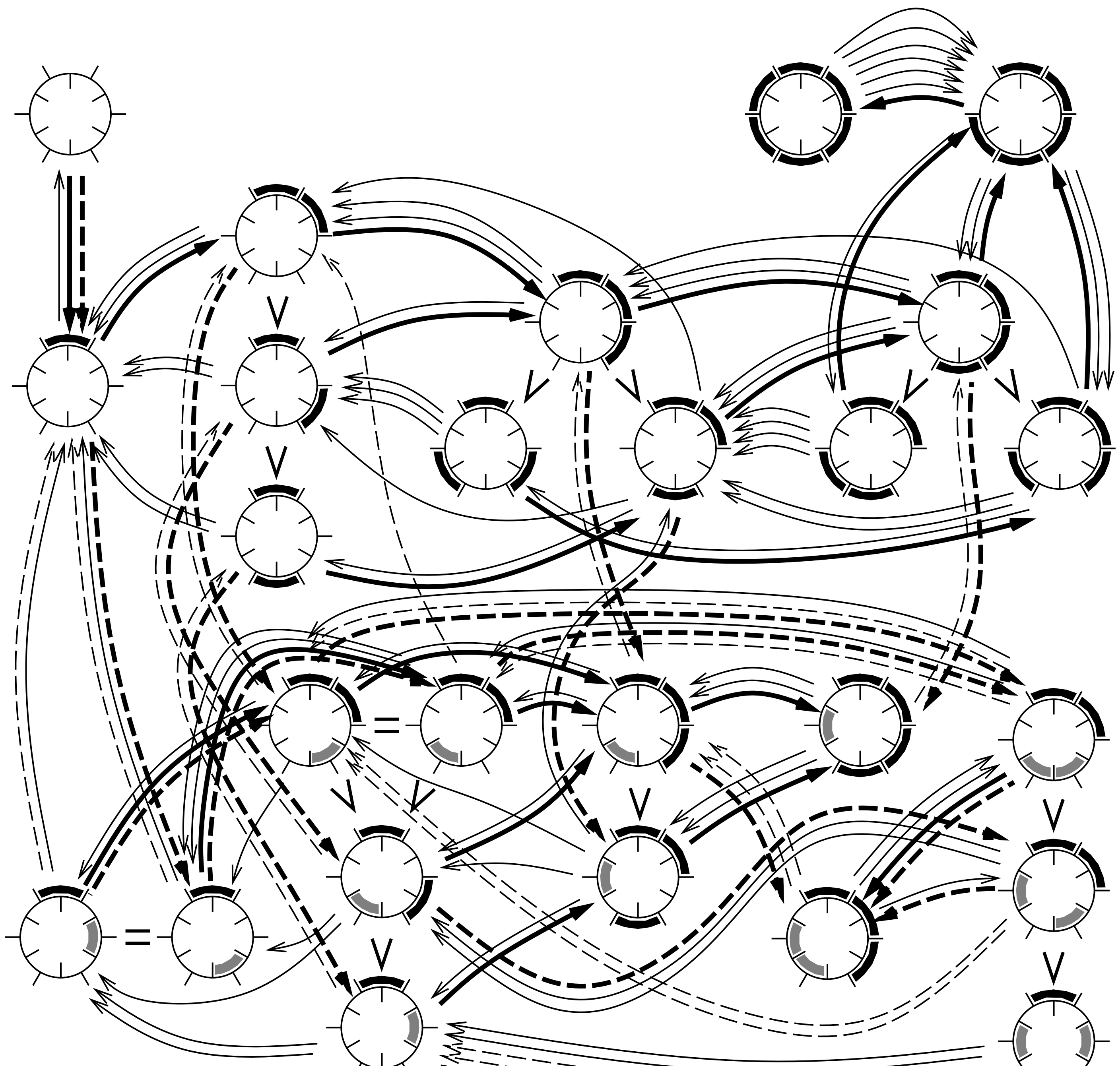


Figure 5: Transitions under BF_6.

arrow exiting from it for every A (B) call it contains. The wedges and equal signs denote a partial order that we will discuss shortly. Note that there is exactly one state for each dihedral exchange class; these (26) classes can be identified by calculations similar to (and simpler than) those used in Section 5 to obtain the dihedral exchange classes for $n = 8$. Figure 6 illustrates sample paths starting in the empty state, with integers denoting current slot weights.

This section proves the optimality of BF_6 under the following two assumptions:

Assumption 1: All optimal algorithms are greedy (as noted earlier, this is effectively a constraint on the offered loads ρ_A, ρ_B).

Assumption 2: As before, call holding times are independent, exponentially distributed random variables with possibly different rate parameters, but now no restrictions are placed on the arrival processes.

The arguments below do not extend to $n \geq 7$ without further assumptions, but they do work, and become much easier, for $n \leq 5$; these remaining small-n cases are left to the reader.

Theorem 2 *Under Assumptions 1 and 2, algorithm BF_6 is optimal.*

Proof: Let V be an optimal, and (by Assumption 1) greedy algorithm that differs from BF_6, i.e., there exists at least one state in which the two algorithms will pack a new arrival but into different slots, creating thereby new states in different dihedral exchange classes. Using a coupling argument we prove the contradiction that, over a given time interval starting in a given initial state, the expected number of calls accepted by V is at most the expected number accepted by BF_6.

Instead of comparing V directly to BF_6, it will be more convenient to compare both to a version of BF_6 that follows the decisions of V as described below. Consider V's decisions on a random sample path ω starting in a fixed but arbitrary state x_0. In parallel with these decisions, we track those made by algorithm W on a sample path ω'. Algorithm W makes the same decisions as BF_6 except that W never packs a call that V can not also pack; and the sample path ω' is the same as ω except possibly for the identities of the calls at departure events; these are determined by couplings such as those shown in Figure 7.

Couplings are defined only between states with the same numbers of A's and B's; between two such states a coupling is simply a one-to-one map of A's onto A's plus a one-to-one map of B's onto B's. For the argument below, the maps can be arbitrary so long as they satisfy the constraints given by the arrows in Figure 7 or constraints

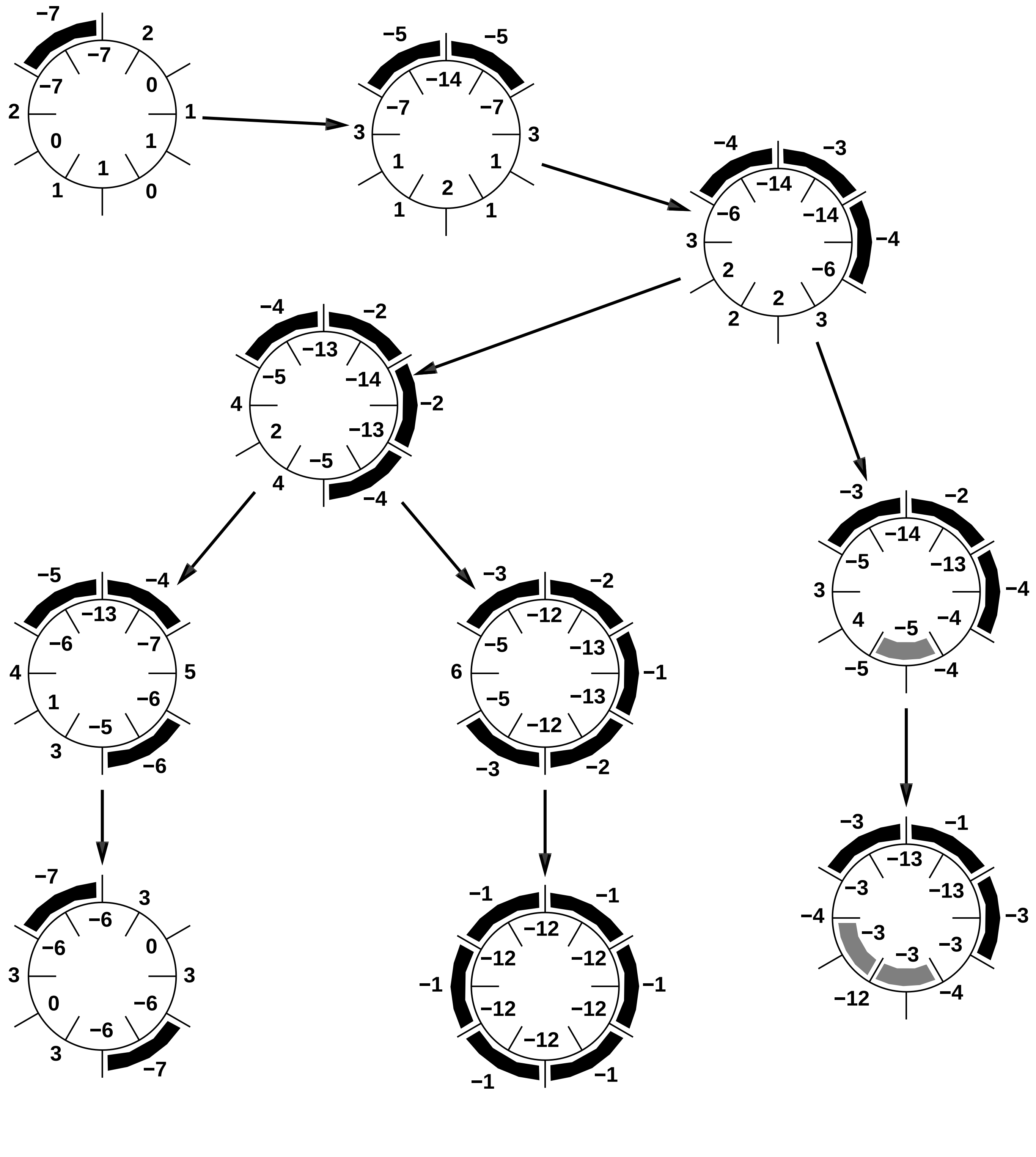


Figure 6: Sample paths for BF_6.

symmetric to those in the figure. Only two pairs of states are constrained:

1. The second and fourth A of $\langle AAA..A..\rangle$ must be mapped in one of the two possible ways to the first and fourth A of $\langle AAAA....\rangle$. The first and third A in the state $\langle AAA..A..\rangle$ can be mapped in either of the two possible ways to the middle two A's of the state $\langle AAAA....\rangle$. The figure shows one of the four equivalent possibilities.

2. For the states $\langle AAA......\rangle$ and $\langle AA..A....\rangle$, we need any constraint that forces the third A of the latter state to be mapped into either the first or third A of the former state.

For given ω, ω' let x_i, x_i', $i \geq 1$, be the respective states just after the i-th event of ω and ω' under V and W, and let the initial states be equal: $x_0 = x_0'$. Note that, by definition of W, the numbers of A's and B's in x_i' is at most the respective numbers in x_i, for all $i \geq 0$. We will verify that the converse also holds, so that the numbers of A's and B's in x_i are equal to the respective numbers in x_i' for all $i \geq 0$.

In fact, we prove a stronger result expressed in terms of the ordering $\leq$ of states shown in Figure 5. More precisely, the relation is a reflexive and transitive partial order without the trichotomy property (two elements of the partial order can be equal without being identical). Note that states with different numbers of A's or different numbers of B's are incomparable under $\leq$. On the other hand, as explained later, not every two states with equal numbers of A's and B's need to be comparable; the two instances of this are the states $\langle AA..AA..\rangle, \langle AAA..A..\rangle$ and the states $\langle AA..A....\rangle, \langle A..A..A..\rangle$.

We now come to the heart of the argument, which is a proof of

Lemma 1 *The inequality $x_i \leq x_i'$ holds for all $i \geq 0$.*

Before giving the case analysis proving the lemma, we verify that it gives the desired result. First, we make the informal observation that the coupling process has no effect on the probability laws governing sample paths, i.e., the sample paths ω' created by the coupling from random sample paths ω are stochastically no different than the original sample paths ω. This property is guaranteed by Assumption 2 which implies that, in any given state, each A call is equally likely to be the next A call to depart, and each B call is equally likely to be the next B call to depart. Then by a standard coupling argument [6], the deterministic majorization of the lemma parlays easily into the probabilistic assertion that the expected number of calls accepted under W over a given time interval starting in state x_0 is at least that under V over the same time interval starting in the same initial state. It follows that, since V is optimal, W must also be optimal; and since V differs from BF_6, so also must W differ from BF_6. But by definition of W, the only way W can differ from BF_6 is by making non-greedy decisions in one or more states.

This contradicts the optimality of W by Assumption 1 and completes the proof of the theorem.

Proof of Lemma 1. The proof is by induction on the number of events in ω, ω'. The basis of the induction follows by inspecting Figure 5 and verifying that, in each state that has space for a call, a new call is packed by W in a slot such that the new state is at least as large under $\leq$ as any state obtained by packing the call into one of the remaining available slots, if any. We give an example and then leave the routine check of the remaining states to the reader. Consider the state $\langle A..A......\rangle$ as the example, and note that W packs a new A call into the slot between the A's to obtain the state $\langle AAA......\rangle$ which is larger under $\leq$ than any of the other states, viz., $\langle A..A..A..\rangle$ and $\langle A..AA....\rangle$, that can be reached by packing a new call in $\langle A..A......\rangle$. The case of packing a new B call is trivial since the next state is the same (up to dihedral symmetries) no matter which of the two available slots the new B call occupies.

For the induction step we consider pairs of states x_i, x_i' with $x_i \leq x_i'$ and hence with the same numbers of A's and B's, and verify that an arrival transition preserves the ordering and that a departure transition *under the coupling shown in Figure 7* also preserves the ordering. The case where x_i and x_i' are the same state is handled by the basis argument above, so we assume that the two states differ. To illustrate the checks that are needed, observe that a new A call is packed by V and W into the respective states of $\langle A.B.A....\rangle < \langle AA...B...\rangle$ to produce the respective new states in $\langle A.B.AA..\rangle < \langle AAA.B...\rangle$. As a further example, let W and V be in the respective states of $\langle A.B.......\rangle = \langle A...B.....\rangle$. Algorithm W packs a new A call to produce the new state $\langle AA.B.....\rangle$. Algorithm V packs a new A call into $\langle A...B.....\rangle$ to produce the same state that W produces, or it produces one of the states $\langle AA...B...\rangle$ or $\langle A...B.A..\rangle$. But the latter two states are respectively equal to and less than the state produced by W'. To illustrate departure transitions that must be checked, we note that, under the coupling, a departure from the states of $\langle AAA..A..\rangle < \langle AAAA....\rangle$ produces the same state $\langle AAA......\rangle$ or the respective states of $\langle A..A..A..\rangle < \langle AAA......\rangle$. Note that, consistent with this last example, we need not be concerned with an ordering of the pair of states $\langle A..A..A..\rangle, \langle AA..A....\rangle$ for it is not possible for V to be in one of these states and W to be in the other. To see this we need only observe that
(i) the coupling constraints do not allow this pair to arise as a result of departures from larger states ordered by $\leq$, and
(ii) W' can not reach $\langle A..A..A..\rangle$ as the result of an arrival, and the only way it can reach $\langle AA..A....\rangle$ on an arrival is from the state $\langle A....A....\rangle$. But from this last state $\langle A..A..A..\rangle$ can not be reached on packing a new arrival under *any* algorithm.

Note that the induction step would fail if one were to make the states $\langle A..A..A..\rangle$ and $\langle AA..A....\rangle$ comparable, no matter what coupling was adopted between these states.

To see this, observe that the only state produced on a departure from $\langle A..A..A..\rangle$ is $\langle A..A......\rangle$. But this state is less than $\langle AA........\rangle$ and greater than $\langle A....A....\rangle$, which are both reachable on departures from $\langle AA..A....\rangle$.

We should also explain the other incomparable pair of states that have the same numbers of A's and B's, viz., $\langle AA..AA..\rangle$ and $\langle AAA..A..\rangle$. To see from Figure 5 that W can not be in one of these states while V is in the other, note first that there is only one state that can produce these states on departures, and second that, of the 3 states with just 4 A's, only $\langle AAAA....\rangle$ is reachable by W on an arrival transition.

Beyond the exceptions already discussed, all pairs of distinct states in Figure 5 with the same numbers of A's and B's must be checked in establishing the induction step. We have illustrated the process by checking 3 cases, each by inspection of Figures 5 and 7; the remaining 12 can also be checked by inspection and need no further comment. ∎

References

[1] Bertsekas, D. (1976), *Dynamic Programming and Stochastic Control,* Academic Press.

[2] Harte, L., Smith, A. and Jacobs, C.A. (1998), *IS-136 TDMA Technology, Economics, and Services,* Artech House, Boston.

[3] Lee, W. C.-Y. (1989), *Mobile Cellular Telecommunication Systems,* McGraw Hill, New York.

[4] Mouly, M. and Pautet, M. (1992), *The GSM System for Mobile Communications,* Palaiseau, France.

[5] Riordan, J. (1958), *Combinatorial Analysis,* Wiley & Sons.

[6] Ross, S. M. (1983), *Stochastic Processes,* Wiley & Sons.

[7] Stanley, R. P. (1986), *Enumerative Combinatorics, Volume I,* Wadsworth & Brooks/Cole, Mathematics Series.

[8] Whittle, P. (1982), *Optimization over Time: Dynamic Programming and Stochastic Control,* Wiley & Sons.