for any task system and on-line algorithm there are sequences of tasks which force the algorithm to reach a competitive factor of at least $2n - 1$.

Here we give a different algorithm which obtains the same competitive factor of $2n - 1$ for almost arbitrary sequences of tasks. Our algorithm works as long as each task is an integer combination of a set of n basis vectors $b_1, \ldots, b_n$, where vector b_i is zero in all components except the ith. The basis set is fixed and known in advance. (For example, if all the task vectors are known to have integral costs, then we let b_i to be the unit vector with a one in position i and zeros elsewhere.) A set of tasks having this structure will be called *discrete tasks*.

Our $(2n - 1)$-competitive algorithm for discrete tasks is called GBALE, because it is a generalized form of the BALE algorithm given in Section 6. Our algorithm will use a simulation of an instance of BALE running on an $(n - 1)$-server problem with excursions. The size of the server problem is the same as the number of states of the task system, and the distance matrix for the server problem, D, is the same as the distance matrix for the task system. Furthermore, the cost of an excursion for vertex i, r_i will be the non-zero component of b_i, the ith basis vector for the discrete tasks.

To process a sequence of tasks $T = T(1), T(2), \ldots$ algorithm GBALE simulates algorithm BALE, on a new sequence of requests $\sigma = \sigma(1), \sigma(2), \ldots$. For each task $T(i)$, a sequence of zero or more requests of σ are generated. Before each task is processed (as well as after) the following invariant is maintained relating the state of BALE and GBALE.

> The vertex that BALE has decided not to occupy with a server is the same as the state of the task system chosen by GBALE.

It remains to describe the sequence of requests corresponding to a task $T(i)$. First expand $T(i)$ into a non-negative integer combination of the basis vectors as follows:

$$T(i) = m_1 b_1 + m_2 b_2 + \cdots + m_n b_n.$$

The requests generated are m_1 requests to vertex 1, m_2 requests to vertex 2, and so on, ending with m_n requests to vertex n. We shall let $\tau(i)$ denote this subsequence of σ.

In order to process $T(i)$, algorithm GBALE gives $\tau(i)$ to BALE. The state reached by BALE at the end of $\tau(i)$ is the state chosen by GBALE to process request $T(i)$. The invariant is thus maintained.

THEOREM 11. *Algorithm GBALE is an $(2n - 1)$-competitive algorithm for any metrical task system with discrete tasks.*

Proof. First we shall prove the claim

$$C_{\mathrm{GBALE}}(T(i)) \leq C_{\mathrm{BALE}}(\tau(i)).$$

Observe that the move cost incurred by GBALE is at most that of BALE. This is because GBALE and BALE start and end in the same state. By the triangle inequality, the cheapest way to effect this change is to move there directly, which is what GBALE does.

Let j be the state (the vertex that is unoccupied by a server) in which BALE chooses to process the last request of $\tau(i)$. (This is the state chosen by GBALE to process $T(i)$.) The task cost incurred by GBALE in processing $T(i)$ is $m_j r_j$. The excursion costs incurred by BALE are at least this much for the following reason. We know that during the last request to vertex j in $\tau(i)$, algorithm BALE is in state j. This is because BALE will never change its state in response to a free request, such as those after the last request to j. Furthermore, BALE must have been in state j during all of the requests to j, because BALE will never move into state j in response to a request to j. Therefore, BALE must have been in state j during all of the requests to j, and therefore incurs a cost of at least $m_j r_j$. This completes the proof of the claim.

Next, we shall relate the costs of the optimal algorithms for processing T and σ. Given an off-line algorithm A for processing T there is an off-line algorithm A' for processing σ at exactly the same cost. Algorithm A' stays in the same state during all of the requests of $\tau(i)$, and this is the state used by A to process $T(i)$. From this, we can conclude:

$$C_{\mathrm{OPT}}(\sigma) \leq C_{\mathrm{OPT}}(T).$$

Combining the above two inequalities with the fact that BALE is $(2n-1)$-competitive proves the theorem. □

There is a natural class of tasks which was not considered by Borodin *et al.* [2], for which we have obtained an algorithm with a better competitive factor. This is the case where each component of each task vector is either 0 or ∞. (This means that to process a task, the algorithm must change to a state in which the task cost is 0.) We call such a task a *forcing task*. In this case we obtain an $(n-1)$-competitive algorithm.

Given a forcing task system S in which there are state transition costs that are zero, we can transform it to an equivalent task system S' in which there are no such zero-cost transitions. We do this by dividing the states of S into equivalence classes, where two states are equivalent if and only if the transition cost between them is zero. We define S' to have one state for each of these equivalence classes. It is easy to see that any competitive on-line algorithm for S' gives one with the same competitive factor for S. A task t given to S is translated into t', a task for S' as follows: A component of t' is ∞ if all the states in the equivalence class of that component are ∞ in t, and 0 otherwise.

Our algorithm, GBAL, is built upon BAL, the $(n-1)$-competitive algorithm for the $(n-1)$-server problem. The construction is very similar (but not identical) to that used above in constructing GBALE.

For a task sequence T, GBAL generates a request sequence σ which it applies to BAL. For task $T(i)$ in T there is a subsequence $\tau(i)$ in σ. The invariant is maintained that before and each after $T(i)$, the state of GBAL and BAL are the same. (The state of BAL is the name of the vertex not covered by a server.)

The subsequence $\tau(i)$ is generated as follows. Let S be the set of states for which the task component in $T(i)$ is infinite. Let s be the current state of BAL. As long as $s \in S$, a request to s is generated. Eventually, since BAL is competitive, and there are no zero-cost transitions, a point must be reached where $s \notin S$. This marks the end of $\tau(i)$. Algorithm GBAL can now use this state s to process the task $T(i)$, since it is one of the zero-cost states of $T(i)$.

THEOREM 12. *Algorithm GBAL is an $(n-1)$-competitive algorithm for any metrical task system with forcing tasks.*

Proof. We have $C_{\mathrm{GBAL}}(T(i)) \leq C_{\mathrm{BAL}}(\tau(i))$, because of the triangle inequality, and the fact that BAL and GBAL start and end in the same state.

Furthermore, any off-line algorithm A for T gives an off-line algorithm A' for σ which costs no more. Algorithm A' stays in the same state during all of the requests of $\tau(i)$, and this is the state used by A to process $T(i)$. From this, we can conclude that $C_{\mathrm{OPT}}(\sigma) \leq C_{\mathrm{OPT}}(T)$. Combining these inequalities with the fact that BAL is $(n-1)$-competitive proves the theorem. □

8. OPEN PROBLEMS

The most obvious open problem is to devise a k-competitive algorithm for the symmetric k-server problem. We conjecture that such an algorithm exists, but have been unable even to extend our solution to the 2-server problem to three or more servers. A computer search has revealed that there are 3-competitive algorithms for certain instances of the 3-server problem. Further evidence for this conjecture was supplied by Chrobak, Karloff, Payne, and Vishwanathan,[2] who have recently devised a simple k-competi-

---

[2] Personal communication.

tive algorithm for k servers on a line (a graph in which the distances are consistent with the Euclidean distances between points on a line).

We do know that there is an algorithm for the symmetric k-server problem in which the competitive factor depends only on k and n. Because the k-server problem is a forced task system with $\binom{n}{k}$ states, there is an $\left(\binom{n}{k} - 1\right)$-competitive algorithm. Although this result is very weak, it is the best bound we know which is independent of the distances.

An even more difficult problem is to find an algorithm for the k-server problem that matches the lower bound of Theorem 6 when compared to an optimal h-server algorithm. It would be particularly interesting if there was a single algorithm (independent of h) that achieved this bound for every h. The LRU algorithm for the uniform server problem (where all move costs equal one) has this property [11].

Nothing is known about server problems with excursions beyond the results described in Section 6. We conjecture that there is a 3-competitive algorithm for any one-server problem with excursions when the cost to move from i to j is a constant times the excursion cost. In [8], a procedure is described that will compute (in principle) the minimum competitive factor for any instance of a server problem. We have used this procedure to verify this conjecture in several small special cases.

The definition of competitiveness carries over in a natural way to randomized on-line algorithms [7]. In this case we want the expected cost of the randomized on-line algorithm (taken over all possible outcomes of its coin flips) to be within a constant factor of the optimum off-line algorithm on any sequence of requests. In [5], a randomized algorithm for the uniform k-server problem (the paging problem) is given. This algorithm is roughly $2\ln(k)$. In [9], the competitive factor is reduced to $\ln(k)$, which is optimal. A natural problem is to consider the non-uniform case.

In [10], Raghavan and Snir describe an even more restricted class of randomized on-line algorithms for server problems. These algorithms are memoryless in the sense that the only state information they maintain from one request to the next is the current arrangement of the servers. They show that their *harmonic algorithm* is k-competitive for the uniform problem, as well as the caching problem (all moves into a vertex have the same cost). They leave as an open problem that of finding a memoryless algorithm that is k-competitive for any k-server problem.

We know very little about asymmetric server problems. For a particular asymmetric k-server problem, let Δ be the maximum over all cycles in the graph of the ratio of the cost of moving around the cycle in one direction to that in the other direction. Any c-competitive algorithm for a symmetric server problem can be used to give a $c\Delta$-competitive algorithm for the asymmetric problem. This is done by letting $d'_{ij} = \frac{1}{2}(d_{ij} + d_{ji})$, and running the c-competitive algorithm on the new problem. Similarly, a lower bound of c on the competitive factor in the symmetric problem gives a lower bound of c/Δ in the asymmetric problem.

REFERENCES

1. D. BLACK AND D. D. SLEATOR, Algorithms for the 1-server problem with excursions, in preparation.
2. A. BORODIN, N. LINIAL, AND M. SAKS, An optimal online algorithm for metrical task systems, *in* "Proceedings, 19th Annual ACM Symposium on Theory of Computing, New York, 1987," pp. 373–382.
3. A. R. CALDERBANK, E. G. COFFMAN, JR., AND L. FLATTO, Sequencing problems in two-server systems, *Math. Oper. Res.* **10**, No. 4 (1985), 585–598.
4. A. R. CALDERBANK, E. G. COFFMAN, JR., AND L. FLATTO, Sequencing two servers on a sphere, *Commun. Statist.-Stochastic Models* **1**, No. 1 (1985), 17–28.
5. A. FIAT, R. M. KARP, M. LUBY, L. A. MCGEOCH, D. D. SLEATOR, AND N. E. YOUNG, "Competitive Paging Algorithms," Carnegie Mellon University Computer Science technical report CMU-CS-88-196, 1988.
6. A. R. KARLIN, M. S. MANASSE, L. RUDOLPH, AND D. D. SLEATOR, Competitive snoopy caching, *Algorithmica* **3**, No. 1 (1988), 79–119.
7. M. S. MANASSE, L. A. MCGEOCH, AND D. D. SLEATOR, Competitive algorithms for on-line problems, *in* "Proceedings, 20th Annual ACM Symposium on Theory of Computing, Chicago, 1988," pp. 322–333.
8. L. A. MCGEOCH, D. D. SLEATOR, AND C. TOMASI, Decision procedures for competitive algorithms, in preparation.
9. L. A. MCGEOCH AND D. D. SLEATOR, "A Strongly Competitive Randomized Paging Algorithm," Carnegie Mellon University Computer Science technical report CMU-CS-89-122, 1989; *Algorithmica*, in press.
10. P. RAGHAVAN AND M. SNIR, Memory versus randomization in on-line algorithms, *in* "*Automata, Languages, and Programming*, Lecture Notes in Computer Science," Vol. 372, pp. 687–703, Springer-Verlag, New York, 1988.
11. D. D. SLEATOR AND R. E. TARJAN, Amortized efficiency of list update and paging rules, *Comm. ACM* **28**, No. 2 (1985), 202–208.