Off-line Temporary Tasks Assignment

Yossi Azar[1] and Oded Regev[2]

[1] Dept. of Computer Science, Tel-Aviv University, Tel-Aviv, 69978, Israel.
azar@math.tau.ac.il ***

[2] Dept. of Computer Science, Tel-Aviv University, Tel-Aviv, 69978, Israel.
odedr@math.tau.ac.il

Abstract. In this paper we consider the temporary tasks assignment problem. In this problem, there are m parallel machines and n independent jobs. Each job has an arrival time, a departure time and some weight. Each job should be assigned to one machine. The load on a machine at a certain time is the sum of the weights of jobs assigned to it at that time. The objective is to find an assignment that minimizes the maximum load over machines and time.

We present a polynomial time approximation scheme for the case in which the number of machines is fixed. We also show that for the case in which the number of machines is given as part of the input (i.e., not fixed), no algorithm can achieve a better approximation ratio than $\frac{4}{3}$ unless $P = NP$.

1 Introduction

We consider the off-line problem of non-preemptive load balancing of temporary tasks on m identical machines. Each job has an arrival time, departure time and some weight. Each job should be assigned to one machine. The load on a machine at a certain time is the sum of the weights of jobs assigned to it at that time. The goal is to minimize the maximum load over machines and time. Note that the weight and the time are two separate axes of the problem.

The load balancing problem naturally arises in many applications involving allocation of resources. As a simple concrete example, consider the case where each machine represents a communication channel with bounded bandwidth. The problem is to assign a set of requests for bandwidth, each with a specific time interval, to the channels. The utilization of a channel at a specific time t is the total bandwidth of the requests, whose time interval contains t, which are assigned to this channel.

Load balancing of permanent tasks is the special case in which jobs have neither an arrival time nor a departure time. This special case is also known as the classical scheduling problem which was first introduced by Graham [5, 6]. He described a greedy algorithm called "List Scheduling" which has a $2 - \frac{1}{m}$ approximation ratio where m is the number of machines. Interestingly, the same

*** Research supported in part by the Israel Science Foundation and by the US-Israel Binational Science Foundation (BSF).

analysis holds also for load balancing of temporary tasks. However, until now, it was not known whether better approximation algorithms for temporary tasks exist.

For the special case of permanent tasks, there is a polynomial time approximation scheme (PTAS) for any fixed number of machines [6, 10] and also for arbitrary number of machines by Hochbaum and Shmoys [7]. That is, it is possible to obtain a polynomial time $(1+\epsilon)$-approximation algorithm for any fixed $\epsilon > 0$.

In contrast we show in this paper that the model of load balancing of temporary tasks behaves differently. Specifically, for the case in which the number of machines is fixed we present a PTAS. However, for the case in which the number of machines is given as part of the input, we show that no algorithm can achieve a better approximation ratio than $\frac{4}{3}$ unless $P = NP$.

Note that similar phenomena occur at other scheduling problems. For example, for scheduling (or equivalently, load balancing of permanent jobs) on unrelated machines, Lenstra et al. [9] showed on one hand a PTAS for a fixed number of machines. On the other hand they showed that no algorithm with an approximation ratio better than $\frac{3}{2}$ for any number of machines can exist unless $P = NP$.

In contrast to our result, in the on-line setting it is impossible to improve the performance of Graham's algorithm for temporary tasks even for a fixed number of machines. Specifically, it is shown in [2] that for any m there is a lower bound of $2 - \frac{1}{m}$ on the performance ratio of any on-line algorithm (see also [1, 3]).

Our algorithm works in four phases: the rounding phase, the combining phase, the solving phase and the converting phase. The rounding phase actually consists of two subphases. In the first subphase the jobs' active time is extended: some jobs will arrive earlier, others will depart later. In the second subphase, the active time is again extended but each job is extended in the opposite direction to which it was extended in the first subphase. In the combining phase, we combine several jobs with the same arrival and departure time and unite them into jobs with higher weights. Solving the resulting assignment problem in the solving phase is easier and its solution can be converted into a solution for the original problem in the converting phase.

The novelty of our algorithm is in the rounding phase. Standard rounding techniques are usually performed on the weights. If one applies similar techniques to the time the resulting algorithm's running time is not polynomial. Thus, we had to design a new rounding technique in order to overcome this problem.

Our lower bound is proved directly by a reduction from exact cover by 3-sets. It remains as an open problem whether one can improve the lower bound using more sophisticated techniques such as PCP reductions.

2 Notation

We are given a set of n jobs that should be assigned to one of m identical machines. We denote the sequence of events by $\sigma = \sigma_1, ..., \sigma_{2n}$, where each event is an arrival or a departure of a job. We view σ as a sequence of times, the time σ_i is the moment after the i^{th} event happened. We denote the weight of job j by w_j, its arrival time by a_j and its departure time by d_j. We say that a job is active at time τ if $a_j \leq \tau < d_j$. An assignment algorithm for the temporary tasks problem has to assign each job to a machine.

Let $Q_i = \{j | a_j \leq \sigma_i < d_j\}$ be the active jobs at time σ_i. For a given algorithm A let A_j be the machine on which job j is assigned. Let

$$l_k^A(i) = \sum_{\{j | A_j = k, j \in Q_i\}} w_j$$

be the load on machine k at time σ_i, which is the sum of weights of all jobs assigned to k and active at this time. The cost of an algorithm A is the maximum load ever achieved by any of the machines, i.e., $C_A = max_{i,k} l_k^A(i)$. We compare the performance of an algorithm to that of an optimal algorithm and define the approximation ratio of A as r if for any sequence $C_A \leq r \cdot C_{opt}$ where C_{opt} is the cost of the optimal solution.

3 The Polynomial Time Approximation Scheme

Assume without loss of generality that the optimal makespan is in the range $(1, 2]$. That is possible since Graham's algorithm can approximate the optimal solution up to a factor of 2, and thus, we can scale all the jobs' weights by $\frac{2}{l}$ where l denotes the value of Graham's solution.

We perform a binary search for the value λ in the range $(1, 2]$. For each value we solve the $(1+\epsilon)$ relaxed decision problem, that is, either to find a solution of size $(1+\epsilon)\lambda$ or to prove that there is no solution of size λ. From now on we fix the value of λ.

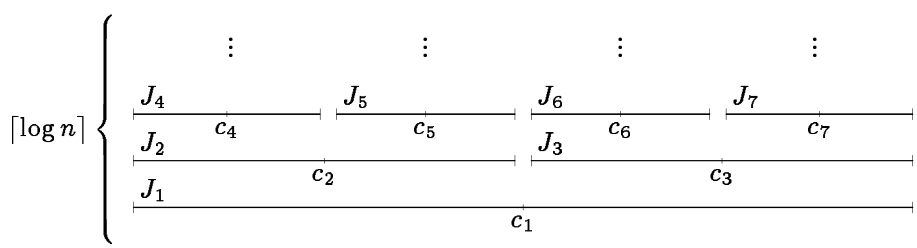

Fig. 1. Partitioning J into $\{J_i\}$

In order to describe the rounding phase with its two subphases we begin with defining the partitions based on which the rounding will be performed. We begin by defining a partition $\{J_i\}$ of the set of jobs J. Let M_i be a set of jobs and consider the sequence of times σ in which jobs of M_i arrive and depart. Since the number of such times is $2r$ for some r, let c_i be any time between the r-th and the $r+1$-st elements in that set. The set J_i contains the jobs in M_i that are active at time c_i. The set M_{2i} contains the jobs in M_i that depart before or at c_i and the set M_{2i+1} contains the jobs in M_i that arrive after c_i. We set $M_1 = J$ and define the M's iteratively until we reach empty sets. The important property of that partition is that the set of jobs that exist at a certain time is partitioned into at most $\lceil \log n \rceil$ different sets J_i.

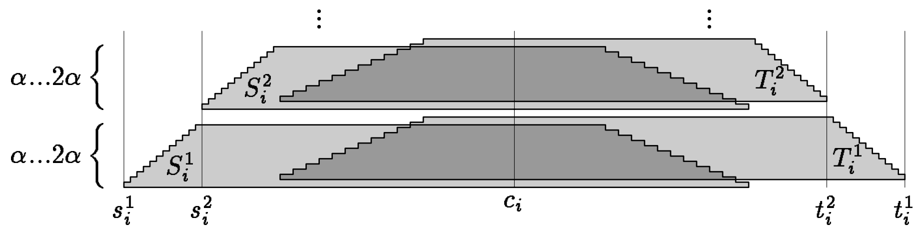

Fig. 2. Partitioning J_i into $\{S_i^j, T_i^j\}$ (R_i is not shown)

We continue by further partitioning the set J_i. We separate the jobs whose weight is greater than a certain constant α and denote them by R_i. We order the remaining jobs according to their arrival time. We denote the smallest prefix of the jobs whose total weight is at least α by S_i^1. Note that its total weight is less than 2α. We order the same jobs as before according to their departure time. We take the smallest suffix whose weight is at least α and denote that set by T_i^1. Note that there might be jobs that are both in S_i^1 and T_i^1. We remove the jobs and repeat the process with the jobs left in J_i and define $S_i^2, T_i^2, ..., S_i^{k_i}, T_i^{k_i}$. The last pair of sets $S_i^{k_i}$ and $T_i^{k_i}$ may have a weight of less than α. We denote by s_i^j the arrival time of the first job in S_i^j and by t_i^j the departure time of the last job in T_i^j. Note that $s_i^1 \leq s_i^2 \leq ... \leq s_i^{k_i} \leq c_i \leq t_i^{k_i} \leq ... \leq t_i^2 \leq t_i^1$.

The first subphase of the rounding phase creates a new set of jobs J' which contains the same jobs as in J with slightly longer active times. We change the arrival time of all the jobs in S_i^j for $j = 1, ..., k_i$ to s_i^j. Also, we change the departure time of all the jobs in T_i^j to t_i^j. The jobs in R_i are left unchanged. We denote the sets resulting from the first subphase by J', J_i', ${S'}_i^j$, ${T'}_i^j$.

The second subphase of the rounding phase further extends the active time of the jobs of the first subphase. We take one of the sets J_i' and the partition we defined earlier to $R_i, {S'}_i^1 \cup {T'}_i^1, {S'}_i^2 \cup {T'}_i^2, ..., {S'}_i^{k_i} \cup {T'}_i^{k_i}$. We order the jobs in ${S'}_i^j$ according to an increasing order of departure time. We take the smallest prefix of this ordering whose total weight is at least β. We extend the departure time of all the jobs in that prefix to the departure time of the last job in that prefix. The process is repeated until there are no more jobs in ${S'}_i^j$. The last prefix may

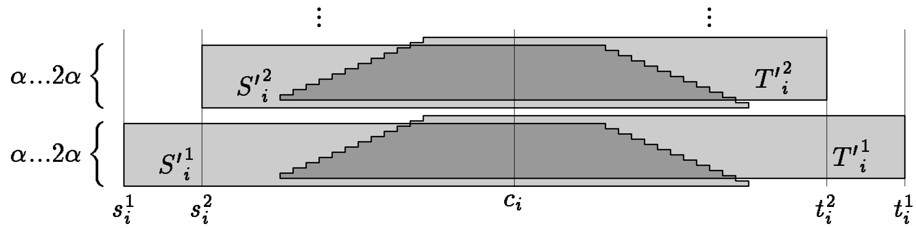

Fig. 3. The set J'_i (after the first subphase)

have a weight of less than β. Similarly, extend the arrival time of jobs in T'^j_i. The jobs in R_i are left unchanged. We denote the sets resulting from the second subphase by J'', J''_i, S''^j_i, T''^j_i.

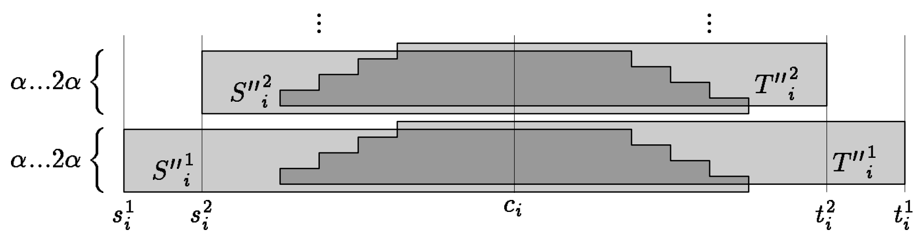

Fig. 4. The set J''_i (after the rounding phase)

The combining phase of the algorithm involves the weight of the jobs. Let J''_{st} be the set of jobs in J'' that arrive at s and depart at t. Assume the total weight of jobs whose weight is at most γ in J''_{st} is $a\gamma$. The combining phase replaces these jobs by $\lceil a \rceil$ jobs of weight γ. We denote the resulting sets by J'''_{st}. The set J''' is created by replacing every J''_{st} with its corresponding J'''_{st}, that is, $J''' = \cup_{s<t} J'''_{st}$.

The solving phase of the algorithm solves the modified decision problem of J''' by building a layered graph. Every time $\sigma_i \in \sigma$ in which jobs arrive or depart has its own set of vertices called a layer. In each layer we hold a vertex for every possible assignment of the current active jobs to machines; that is, an assignment whose makespan is at most λ for a certain λ. Two vertices of adjacent layers are connected by an edge if the transition from one assignment of the active jobs to the other is consistent with the arrival and departure of jobs at time σ_i. Now we can simply check if there is a path from the first layer to the last layer.

In the converting phase the algorithm converts the assignment found for J''' into an assignment for J. Assume the number of jobs of weight γ in J'''_{st} that are assigned to a certain machine i is r_i. Replace these with jobs smaller than γ in J''_{st} of total weight of at most $(r_i+1)\gamma$. Note that all the jobs will be assigned that way since the replacement involves jobs whose weight is at most γ and from volume consideration there is always at least one machine with a load of at most r_i of these jobs. The assignment for J'' is also an assignment for J' and J.

4 Analysis

Lemma 1. *For $\alpha = \frac{\epsilon}{2\lceil \log n \rceil}$, given a solution whose makespan is λ to the original problem J, the same solution applied to J' has a makespan of at most $\lambda + \epsilon$. Also, given a solution whose makespan is λ to J', the same solution applied to J has a makespan of at most λ.*

Proof. The second claim is obvious since the jobs in J are shorter than their perspective jobs in J'. As for the first claim, every time τ is contained in at most $\lceil \log n \rceil$ sets J_i. Consider the added load at τ from jobs in a certain set J_i. If $\tau < s_i^1$ or $\tau \geq t_i^1$ then the same load is caused by J'_i and J_i. Assume $\tau < c_i$ and define $s_i^{k_i+1} = c_i$, the other case is symmetrical. Then for some j, $s_i^j \leq \tau < s_i^{j+1}$ and the added load at τ is at most the total load of S_i^j which is at most 2α. Summing on all sets J_i, we conclude that the maximal load has increased by at most $2\alpha\lceil \log n \rceil = \epsilon$.

Lemma 2. *For $\beta = \frac{\epsilon^2}{4m\lceil \log n \rceil}$, given a solution whose makespan is λ to the problem J', the same solution applied to J'' has a makespan of at most $\lambda(1+\epsilon)$. Also, given a solution whose makespan is λ to J'', the same solution applied to J' has a makespan of at most λ.*

Proof. The second claim is obvious since the jobs in J' are shorter than their perspective jobs in J''. As for the first claim, given a time τ and a pair of sets S'^j_i, T'^j_i from J'_i we examine the increase in load at τ. If $\tau < s_i^j$ or $\tau \geq t_i^j$ it is not affected by the transformation because no job in $T'^j_i \cup S'^j_i$ arrives before s_i^j or departs after t_i^j. Assume that $\tau < c_i$, the other case is symmetrical. So τ is affected by the decrease in arrival time of jobs in T'^j_i. It is clear that the way we extend the jobs in T'^j_i increases the load at τ by at most β. Also, since $\tau \geq s_i^j$, we know that the load caused by S'^j_i is at least α if $j < k_i$. Thus, an extra load of at most β is created by every pair S'^j_i, T'^j_i for $1 \leq j < k_i$ only if the pair contributes at least α to the load. Also, $S_i^{k_i}$ for all i contributes an extra load of at most $\beta\lceil \log n \rceil$. Since the total load on all machines at any time is at most λm, the increase in load and therefore in makespan is at most $\frac{\beta}{\alpha}\lambda m + \beta\lceil \log n \rceil = \frac{\epsilon\lambda}{2} + \frac{\epsilon^2}{4m} \leq \epsilon\lambda$.

Lemma 3. *For $\gamma = \frac{\epsilon\beta}{m} = \frac{\epsilon^3}{4m^2\lceil \log n \rceil}$, given a solution whose makespan is λ to the problem J'', the modified problem J''' has a solution with a makespan of $\lambda(1+\epsilon)$. Also, given a solution whose makespan is λ to the modified problem J''', the solution given by the converting phase for the problem J'' has a makespan of at most $\lambda(1+\epsilon)$.*

Proof. Consider a solution whose makespan is λ to J''. If the load of jobs smaller than γ in a certain J''_{st} on a certain machine i is $r_i\gamma$, we replace it by at most $\lceil r_i \rceil$ jobs of weight γ. Note that this is an assignment to J''' and that the increase in load on every machine is at most γ times the number of sets J''_{st} that contain jobs which are scheduled on that machine. As for the other direction, consider a solution whose makespan is λ to J'''. The increase in load on every machine by the replacement described in the algorithm is also at most γ times the number of sets J''_{st} that contain jobs which are scheduled on that machine.

The number of sets J''_{st} that can coexist at a certain time is at most $\frac{\lambda m}{\beta}$ since the weight of each set is at least β and the total load at any time is at most λm. Therefore, the increase in makespan is at most $\gamma \frac{\lambda m}{\beta} = \epsilon\lambda$.

Lemma 4. *The running time of the algorithm for solving the relaxed decision problem for λ is bounded by $O(n^{16\lambda m^3 \log m\epsilon^{-3}+1})$. The running time of the PTAS is the above bound times $O(\log 1/\epsilon)$.*

Proof. Every layer in the graph stores all the possible assignments of jobs to machines. Since the smallest job is of weight γ, the maximum number of active jobs at a certain time is $\frac{\lambda m}{\gamma}$. So, the maximum number of edges in the graph is $nm^{2\lambda \frac{m}{\gamma}}$ and the running time of the relaxed decision problem algorithms is $O(nm^{2\lambda \frac{m}{\gamma}}) = O(nm^{8\lambda m^3 \lceil \log n \rceil \frac{1}{\epsilon^3}}) = O(n^{16\lambda m^3 \log m\epsilon^{-3}+1})$. The running time of the PTAS is the above bound times $O(\log 1/\epsilon)$ since there are $O(\log 1/\epsilon)$ phases in the binary search for the appropriate λ.

5 The unrestricted number of machines case

In this section we show that in case the number of machines is given as part of the input, the problem cannot be approximated up to a factor of $4/3$ in polynomial time unless $P = NP$. We show a reduction from the exact cover by 3-sets ($X3C$) which is known to be NP-complete [4, 8]. In that problem, we are given a set of $3n$ elements, $A = \{a_1, a_2, ..., a_{3n}\}$, and a family $F = \{T_1, ..., T_m\}$ of m triples, $F \subseteq A * A * A$. Our goal is to find a covering in F, i.e. a subfamily F' for which $|F'| = n$ and $\cup_{T_i \in F'} T_i = A$.

Given an instance for the $X3C$ problem we construct an instance for our problem. The number of machines is m, the number of triples in the original problem. There are three phases in time. First, there are times $1, ..., m$, each corresponding to one triple. Then, times $m+1, ..., m+3n$ each corresponding to an element of A. And finally, the two times $m+3n+1, m+3n+2$.

There are four types of jobs. The first type are m jobs of weight 3 starting at time 0. Job r, $1 \le r \le m$ ends at time r. For any appearance of a_j in a triple T_i corresponds a job of the second type of weight 1 that starts at i and ends at $m+j$ and another job of the third type of weight 1 that starts at time $m+j$. Among all the jobs that start at time $m+j$, one ends at $m+3n+2$ while the rest end at $m+3n+1$. The fourth type of jobs are $m-n$ jobs of weight 3 that start at $m+3n+1$ and end at $m+3n+2$.

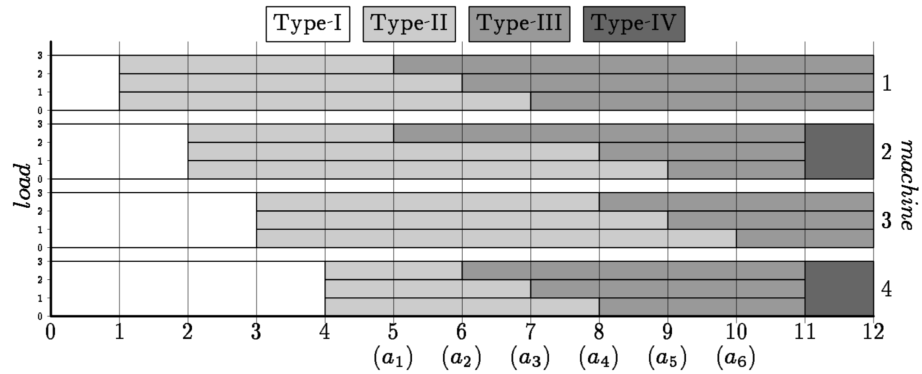

Fig. 5. An assignment for the scheduling problem corresponding to $m = 4$, $n = 2$, $F = \{(1, 2, 3), (1, 4, 5), (4, 5, 6), (2, 3, 4)\}$

We show that there is a schedule with makespan at most 3 if and only if there is an exact cover by 3-sets. Suppose there is a cover. We schedule a job of the first type that ends at time i to machine i. We schedule the three jobs of the second type corresponding to T_i to machine i. At time $m + j$, some jobs of type two depart and the same number of jobs of type three arrives. One of these jobs is longer than the others since it ends at time $m + 3n + 2$. We schedule that longer job to machine i where T_i is the triple in the covering that contains j. At time $m + 3n + 1$ many jobs depart. We are left with $3n$ jobs, three jobs on each of the n machines corresponding to the 3-sets chosen in the cover. Therefore, we can schedule the $m - n$ jobs of the fourth type on the remaining machines.

Now, assume that there is a schedule whose makespan is at most 3. One important property of our scheduling problem is that at any time τ, $0 \leq \tau < m + 3n + 2$ the total load remains at $3m$ so the load on each machine has to be 3. We look at the schedule at time $m + 3n + 1$. Many jobs of type three depart and only the long ones stay. The number of these jobs is $3n$ and their weight is 1. Since $m - n$ jobs of weight 3 arrive at time $m + 3n + 1$, the $3n$ jobs must be scheduled on n machines. We take the triples corresponding to the n machines to be our covering. Assume by contradiction that this is not a covering. Therefore, there are two 3-sets that contain the same element, say a_j. At time $m + j$ only one long job arrives. The machine in which a shorter job was scheduled remains with a load of 3 until time $m + 3n + 1$ and then the short job departs and its load decreases to at most 2. This is a contradiction since at time $m + 3n + 1$ there are n machines each with 3 long jobs.

Corollary 1. *For every $\rho < \frac{4}{3}$, there does not exist a polynomial ρ-approximation algorithm for the temporary tasks assignment problem unless $P = NP$.*

6 Acknowledgments

We are grateful to Jiří Sgall and Gerhard J. Woeginger for their helpful discussions.

References

[1] Y. Azar. On-line load balancing. In A. Fiat and G. Woeginger, editors, *Online Algorithms - The State of the Art*, chapter 8, pages 178–195. Springer, 1998.

[2] Y. Azar and L. Epstein. On-line load balancing of temporary tasks on identical machines. In *5th Israeli Symp. on Theory of Computing and Systems*, pages 119–125, 1997.

[3] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[4] M.R. Garey and D.S. Johnson. *Computers and Intractability*. W.H. Freeman and Company, San Francisco, 1979.

[5] R.L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[6] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math*, 17:263–269, 1969.

[7] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *J. of the ACM*, 34(1):144–162, January 1987.

[8] R.M. Karp. *Reducibility among Combinatorial Problems, R.E. Miller and J.W. Thatcher (eds.), Complexity of Computer Computations*. Plenum Press, 1972.

[9] J.K. Lenstra, D.B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *mathprog*, 46:259–271, 1990.

[10] S. Sahni. Algorithms for scheduling independent tasks. *Journal of the Association for Computing Machinery*, 23:116–127, 1976.