Theoretical Computer Science 262 (2001) 569–582

Theoretical Computer Science

www.elsevier.com/locate/tcs

Efficient web searching using temporal factors ☆

Artur Czumaj[a,1], Ian Finch[b], Leszek Gąsieniec[b,2], Alan Gibbons[b], Paul Leng[b], Wojciech Rytter[b,3], Michele Zito[b,*,4]

[a] *Department of Computer and Information Science, New Jersey Institute of Technology, University Heights, Newark, NJ 07102, USA*

[b] *Department of Computer Science, University of Liverpool, Peach Street, L69 7ZF, UK*

Received 8 March 2000; revised 31 May 2000; accepted 7 August 2000
Communicated by M. Crochemore

Abstract

We study the issues involved in the design of algorithms for performing information gathering more efficiently, by taking advantage of anticipated variations in access times in different regions at different times of the day or week. We look at the problem theoretically, as a generalisation of single processor sequencing with *release* times and *deadlines*, in which performance times (*lengths*) of the tasks can change in time. The new problem is called *Variable Length Sequencing Problem* (VLSP). We show that although the decision version of VLSP seems to be intractable in the general case, it can be solved optimally for lengths 1 and 2. This result opens the possibility of practicable algorithms to schedule searches efficiently when expected access times can be categorised as either slow or fast. Some algorithms for more general cases are examined and complexity results derived. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Web searching; Sequencing; Matching; Approximation; Random

☆ A preliminary version has appeared in F. Dehne, A. Gupta, J.-R. Sack, and R. Tamassia, editors, *Algorithms and Data Structures*, *Sixth International Workshop*, *WADS'99*, vol. 1663, *Lecture Notes in Computer Science*, pp. 294–305, Vancouver, Canada, August 11–14, Springer, Berlin, 1999.

* Corresponding author.

E-mail addresses: czumaj@cis.njit.edu (A. Czumaj), ian@csc.liv.ac.uk (I. Finch), leszek@csc.liv.ac.uk (L. Gąsieniec), amg@csc.liv.ac.uk (A. Gibbons), phl@csc.liv.ac.uk (P. Leng), rytter@csc.liv.ac.uk (W. Rytter), michele@csc.liv.ac.uk (M. Zito).

[1] Work partly done while the author was with Heinz Nixdorf Institute and Department of Mathematics & Computer Science at the University of Paderborn, D-33095 Paderborn, Germany, and while visiting the University of Liverpool. Research supported in part by DFG-Sonderforschungsbereich 376.

[2] Supported in part by NUF-NAL (The Nuffield Foundation Awards to Newly Appointed Lecturers) award.

[3] Instytut Informatyki, Uniwersytet Warszawski, Banacha 2, 02-097, Warszawa, Poland.

[4] Supported by EPSRC grant GR/L/77089.

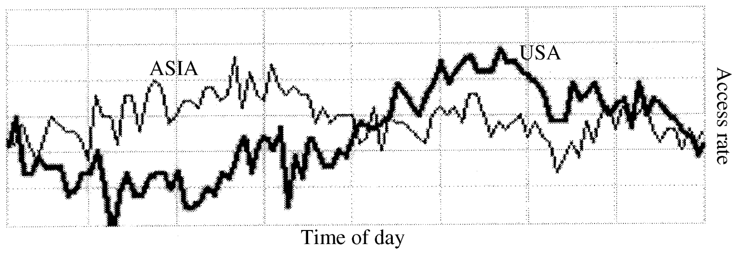

Fig. 1. Access rates for USA and Asia over 24 h. (Derived from data posted by the Anderson News Network http://www.internettrafficreport.com/.)

1. Introduction

The algorithmics of the World Wide Web is an important subject, because of the rapidly growing size of the web, see [1, 4, 3], and one of the crucial factors is the speed of access to the desired information. The access speeds to particular sites on the World Wide Web can vary depending on the time of access. The speed of access from a client site A to a server site B is influenced by at least two load factors: firstly, the local load at the server B, which may influence the speed at which requests are serviced, and secondly, the traffic load on the paths taken between A and B. Both these factors may be subject to temporal variation. For example, local loads and traffic may both be expected to be heavier during daytime hours in the local region, and lowest during periods in the middle of the night. It may also be expected that loads will vary through the week, and perhaps over longer periods, under the influence of local working patterns.

The variations in access rates show different patterns in different regions. Fig. 1 superimposes the plot of variations for sites in Asia on that for those in the USA, for the same 24-h period. In this case, the different peaks of access speed for each region are clearly apparent. These variations create the possibility of reducing overall traversal times by scheduling accesses to take account of expected access times. In this paper, we discuss some computational complexity aspects of algorithms which attempt to make use of this information.

We study the following problem. Assume that we have a single central computer which is being used to collect all the information stored in a certain number of web documents, located at various sites. The information is gathered by scheduling a number of consecutive client/server connections with the required web sites, to collect the information page by page. We assume that the loading time of any particular page from any site may be different at different times, e.g. the access to the page is much slower in peak hours than in off-peak hours. Having a list of pages to be collected along with some information about the access time at any given instant, the goal is to download the pages as quickly as possible.

We define a class of simplified versions of this problem. We prove that under fairly reasonable assumptions the general version of the problem is rather difficult to solve exactly in time polynomial in the number of sites to be connected. However, if connection times are coarsely classified as "fast" or "slow" we prove that information gathering is possible in polynomial time. Then we describe some approximation results with both worst case and average case performance guarantees.

More formally, we define the Variable Length Sequencing Problem (*VLSP*) as follows:

There are n *tasks* (sets of pages to be collected), each of which will be denoted by an integer in the interval $\{1,\dots,n\}$. For each task t and time (unit) $i \in \mathbb{N}^+$, let $l(t,i) \in \mathbb{N}^+$ be the *length* (or performance time) of task t when started at time i.

An *execution sequence* σ is a function specifying for each task t a starting time $\sigma(t) \in \mathbb{N}^+$ with the property that for every t if $\sigma(t) = i$ then no other task w can have $\sigma(w) \in \{i,\dots,i + l(t,i) - 1\}$.

The cost of an execution sequence σ, $C(\sigma)$ is the time unit at which the latest task is completed; that is $k + l(t_{\max},k) - 1$ where

$$k = \sigma(t_{\max}) = \max_{\{1,\dots,n\}} \sigma(t).$$

The VLSP problem asks for an execution sequence of minimum cost.

A few words of comment are needed before proceeding to describe some important algorithmic properties of the problem defined above.

First, it is important to stress that we assume that time proceeds in a sequence of discrete unit time intervals, which we called *time units*, starting from a first interval which, without loss of generality, can be denoted by the integer 1. This seems a reasonable assumption because, although time is a continuous quantity, computer clocks always tick with a fixed predefined discrete frequency. So, for example, if there were just two tasks in the system, each taking i time units if scheduled at time i the optimal execution sequence would use 3 time units, thus having a cost of 3, according to the definition above.

Second, we assume that a table containing some measure of the performance time of any given task at any given time unit is available in advance. Thus algorithms for VLSP will produce an execution sequence in an "off-line" manner. This model is a representation of the case we might wish to apply in a practical web searching algorithm, in which predicted values are known for each $l(t,i)$ on the basis of historical data, similar to those shown in Fig. 1.

Finally, note that we measure the cost of an execution sequence, relative to the point at which we start making measurements of the task lengths. In a practical situation the system may have access time statistics over a long period of time and be interested in computing the optimal cost of an execution sequence with respect to a given initial point.

2. NP-hardness of the general VLSP problem

In this section we prove that the decision version of VLSP is at least as hard as the following SEQUENCING problem.

Instance: A set T of tasks and, for each task $t \in T$, a positive integer length $l(t)$, a release time $r(t) \in \mathbb{N}$, and $d(t) \in \mathbb{N}$ the completion deadline for the task t.

Question: Does there exists a *feasible schedule* for T, that is, a function $\phi : T \to \mathbb{N}$ such that, for each $t \in T$,

$\phi(t) \geqslant r(t)$, $\phi(t) + l(t) \leqslant d(t)$, and, if $t' \in T \setminus \{t\}$, then either $\phi(t') + l(t') \leqslant \phi(t)$ or $\phi(t') \geqslant \phi(t) + l(t)$?

SEQUENCING was proved to be NP-complete in the strong sense by Garey and Johnson [6].

Theorem 1. VLSP *is* NP-*hard.*

Proof. We prove that there is a polynomial time reduction from SEQUENCING to the decision version of VLSP. The same number of tasks is used in both SEQUENCING and VLSP. Let $D = \max_{t \in T} d(t)$. For any task t in SEQUENCING define length in VLSP as follows:

$$l(t,i) = \begin{cases} l(t) + r(t) - i & \text{for all } 1 \leqslant i < r(t), \\ l(t) & \text{for all } r(t) \leqslant i \leqslant d(t) - l(t), \\ D - i + 1 & \text{for all } i > d(t) - l(t). \end{cases}$$

A feasible schedule for the given instance of SEQUENCING is also an execution sequence for the corresponding instance of VLSP. Therefore, if one such a schedule exists the optimal solution for the corresponding instance of VLSP has cost at most D.

Conversely, a solution for a given instance of SEQUENCING can be obtained from a solution of cost at most D for the instance of VLSP defined as described above. Let σ be an execution sequence for the instance of VLSP, with $C(\sigma) \leqslant D$.

We define a feasible schedule for the original instance of SEQUENCING. Three cases need to be considered. By the first constraint in the definition of the instance of VLSP if we start to process task t at any time unit before its release time $r(t)$ in this instance of SEQUENCING, then task t will be always completed at time $l(t) + r(t)$. This means that if in the solution of VLSP there is a task t that is executed before its release time in the corresponding instance of SEQUENCING, it can always be executed at time $r(t)$ without causing any delay. We define $\phi(t) = r(t)$ in this case. The second constraint deals with the case in which execution times of a given task are the same in both SEQUENCING and VLSP. In this case we define $\phi(t) = \sigma(t) - 1$. The third constraint prevents execution of tasks in VLSP when it is too late to do so, i.e. when in the instance of SEQUENCING the deadline for task execution is too close.

The NP-hardness of VLSP follows from the fact that SEQUENCING is NP-hard. □

3. VLSP with slow/fast completion times

Motivated by the intractability of the general setting of VLSP we show that the problem can be solved in polynomial time in some particular cases.

We focus on the possible values of an entry $l(t,i)$. If $S \subset \mathbb{N}^+$, we let VLSP(S) denote the VLSP restricted to those instances having $l(t,i) \in S$ for all tasks t and $i \in \mathbb{N}^+$. In this section we consider the case when $S = \{1,2\}$. This simple abstraction of VLSP has some importance. The values one and two are meant to model an environment in which completion times are coarsely classified as "slow" or "fast". In practice, this may be a realistic simplification since estimates based on historical data might be imprecise. An optimal solution to an instance of VLSP($\{1,2\}$) will consist of an execution sequence that maximises the number of fast tasks.

A similar simplification has been used in the context of the Travelling Salesman Problem (TSP); see e.g. [2, 5, 9]. TSP remains NP-hard even in the case when the only legal values for the city distances are 1 and 2. Recently Engebretsen, see [5] has proved that it is NP-hard to approximate TSP with distances 1 and 2, within $\frac{4709}{4708} - \varepsilon$ for any $\varepsilon > 0$. Surprisingly, however, VLSP($\{1,2\}$) can be solved in polynomial time by the reduction to a classical graph theoretic problem.

Given an instance I of VLSP($\{1,2\}$) and a value $N \in \{n, n+1, \ldots, 2n\}$ two types of graphs can be associated with I.

- Define a bipartite graph $B_N = (V,E)$ such that $V = \mathbf{T} \cup \mathbf{N}$, where $\mathbf{T} = \{t_1, t_2, \ldots, t_n\}$ corresponds to the set of tasks and $\mathbf{N} = \{1, 2, \ldots, N\}$ represents the sequence of time units. An edge connects nodes t_i and j iff task i can be performed in one time unit if its execution starts during time unit j.
- Define the graph G_N on the same vertex set as B_N with edge set

$$E \cup \{(j, j+1) : j = 1, \ldots, N-1\}.$$

Edges in E are called *horizontal edges*, all the others are *vertical edges*.

Example. Let $N = 8$, $n = 6$. Assume that the length function $l(t,i)$ is defined by Table 1.

The graph B_N representing the correspondence between tasks and time units that allow a fast completion is shown in Fig. 2(a). Fig. 2(b) shows the corresponding graph G_N which is instrumental to the polynomial time algorithm described in Theorem 3.

A *matching* in a graph is a set of non-adjacent edges. The following lemma relates the existence of a short execution sequence for VLSP($\{1,2\}$) to the existence of a fairly large matching in G_N.

Table 1
Table with the task lengths if started at different time units

	Time units							
Task	1	2	3	4	5	6	7	8
1	1	2	2	2	2	2	2	2
2	1	2	2	2	2	2	2	2
3	2	1	2	1	2	1	2	2
4	1	2	2	2	1	2	2	2
5	1	2	1	2	1	2	2	2
6	2	2	2	2	2	2	1	1

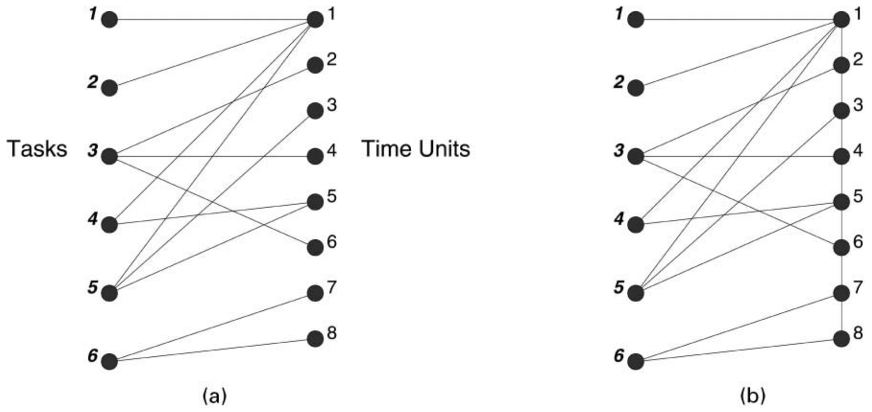

Fig. 2. The graphs B_N and G_N in the given example.

Lemma 2. *G_N has a matching of size at least n if and only if there is an execution sequence σ to* VLSP($\{1,2\}$) *such that $C(\sigma) \leqslant N$.*

Proof. Any matching M of size n in G_N is formed by h horizontal edges and v vertical edges with $h + v = n$. Define the execution sequence of the associated VLSP instance by setting $\sigma(t) = j$ if $(t, j) \in M$ and assigning to all other tasks a starting time given by the smallest index of one of the vertical edges in M.

Conversely if the VLSP can be solved in time N and there are h tasks which take one time step to complete, then $N - h = 2(n - h)$. We get a matching of size n in G_N by using h horizontal edges (edge $(t, j) \in M$ if and only if $l(t, j) = 1$) and $n - h$ vertical edges corresponding to those $n - h$ tasks whose length is 2. $\square$

Going back to the example before the lemma, the thick edges in Fig. 3(a) represent the edges in M. The presence of the horizontal edges $\{\mathbf{2}, 1\}, \{\mathbf{3}, 4\}, \{\mathbf{5}, 5\}$ and $\{\mathbf{6}, 5\}$ in M imply that tasks 2, 3, 5, and 6 can be started in time units 1, 4, 5, and 8, respectively. The vertical edges $\{2, 3\}$ and $\{6, 7\}$ imply that the tasks left unassigned can be

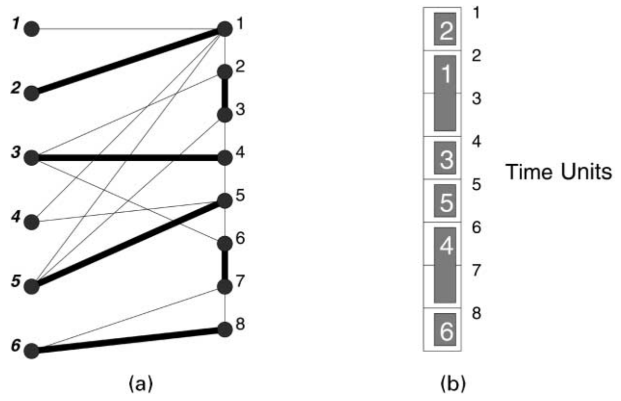

Fig. 3. A matching in G_N and a graphical representation of the task allocation.

started during time unit 2 or 6 and they will be completed by the end of successive time unit. The choice of which task to assign to which time unit is completely immaterial. Fig. 3(b) gives a graphical representation for one possible resulting execution sequence.

We are now in a position to state the main result in this section.

Theorem 3. VLSP($\{1,2\}$) *can be solved optimally in polynomial time.*

Proof. Test all possible values of N between n and $2n$ and take the smallest N such that G_N has a matching of size n. One such matching can be found using any of the algorithms for finding a maximum cardinality matching in a graph (see [10] for example). □

4. Approximation algorithm for VLSP($\{k_1,k_2\}$)

From the analysis in the preceding sections we may conclude that, although a general instance to the sequencing problem for web searching is intractable, we may be able to find an optimal sequence efficiently if we know that accessing popular sites is sometimes fast and sometimes slow. Of course, this situation is only theoretically optimal since in practice, the interpretation of "fast" and "slow" is both variable and approximate; an access time which is expected to be fast may in fact turn out to be slow. The motivation for applying the theoretical solution in a practical algorithm is the expectation that the theoretical optimal solution would be at least a "good" practical solution.

A more general model than the slow/fast access case is obtained if the two possible lengths can be specified arbitrarily. This is consistent with the practical situation in which tasks are still classified as "fast" or "slow" but there is also a measure of how much slower a slow task is than a fast one.

Unfortunately the complexity of VLSP($\{k_1, k_2\}$) with $k_1, k_2 \in \mathbb{N}$ is open. In this section, we define an algorithm which always finds an execution sequence whose cost is close to the cost C^* of an optimal sequence if k_1 is a fixed constant. Let σ be an execution sequence. For $i = 0, \ldots, k_1 - 1$, let x_i be the number of tasks that are allocated at a time unit j with $j \equiv i \bmod k_1$ and are completed in k_1 units in the execution sequence σ.

Lemma 4. *For every execution sequence σ on n tasks, leading to a sequencing of cost $C(\sigma)$ there exists an $i_{\mathrm{big}} \in \{0, \ldots, k_1 - 1\}$ such that*

$$x_{i_{\mathrm{big}}} \geqslant \frac{nk_2 - C(\sigma)}{k_1(k_2 - k_1)}.$$

Proof. Let $x = \sum_{i=0}^{k_1-1} x_i$. There must be an $i \in \{0, \ldots, k_1 - 1\}$ such that $x_i > x/k_1$. The statement of the lemma is true for this particular i. Define i_{big} to be this value of i. If $y = n - x$ then

$$\begin{cases} x_{i_{\mathrm{big}}} + (x - x_{i_{\mathrm{big}}}) + y = n, \\ k_1 x_{i_{\mathrm{big}}} + k_1(x - x_{i_{\mathrm{big}}}) + k_2 y \leqslant C(\sigma) \end{cases}$$

(the second inequality holds because there can be times when it is better not to allocate any task). From these we have

$$x_{i_{\mathrm{big}}} \geqslant \frac{nk_2 - C(\sigma)}{k_2 - k_1} - (x - x_{i_{\mathrm{big}}})$$

and therefore (using $x - x_{i_{\mathrm{big}}} < (k_1 - 1)x_{i_{\mathrm{big}}}$),

$$x_{i_{\mathrm{big}}} \geqslant \frac{nk_2 - C(\sigma)}{k_1(k_2 - k_1)}. \qquad \square$$

The algorithm A_1, which is shown below, works in a number of iterations, defining an execution sequence in each case and returning the shortest sequence. The value of C_0 is defined in Theorem 5.

```
Algorithm A_1
    for $N = k_1 n$ to $k_2 n$ do
        for $i = 0$ to $k_1 - 1$ do
            if $N > C_0$ then
                return an execution sequence of cost $k_2 n$ by defining
                  $\sigma_N^i(t) = k_2(t-1) + 1$ for every $t \in T$.
            else (* $N \leqslant C_0$ *)
                Create a bipartite graph $B_N^i = (U, V, E)$ with
                    $U = T$,
                    $V = \{j : 1 \leqslant j \leqslant N,\ j \equiv i \bmod k_1\}$
                    $\{u, v\} \in E$ if and only if $l(u, v) = k_1$.
                Let $M$ be a maximum matching in $B_N^i$.
                for each $u \in U$ such that $\{u, v\} \in M$ do
                    Let $\sigma_N^i(u) = v$
                Let $\{t_1, \ldots, t_l\}$ be the set of unassigned tasks
                for each $t_j \in \{t_1, \ldots, t_l\}$ do
                    Let $\sigma_N^i(t_j) = N + k_2(j-1) + 1$
    Return the execution sequence $\sigma_N^i$ of minimal cost.
```

Theorem 5. *Algorithm A_1 achieves an approximation ratio $1 + k_1(k_2 - k_1)/k_2$.*

Proof. The algorithm will always produce a solution of cost either $k_2 n$ or $N + k_2(n - x_i)$. We focus on the iteration when $N = C(\sigma_{\text{opt}})$ and on the stage i_{big} as defined in Lemma 4.

Since $x_{i_{\text{big}}}$ tasks are allocated using a maximum matching computation, if $x_{i_{\text{big}}}^{\text{opt}}$ is the number of tasks allocated to time units that are congruent to i_{big} modulus k_1 by the optimal execution sequence, then $x_{i_{\text{big}}}^{\text{opt}} \leqslant x_{i_{\text{big}}}$. Therefore

$$\begin{aligned} N + k_2(n - x_i) &\leqslant C(\sigma_{\text{opt}}) + k_2(n - x_{i_{\text{big}}}^{\text{opt}}) \\ &\leqslant C(\sigma_{\text{opt}}) + k_2\left[n - \frac{nk_2 - C(\sigma_{\text{opt}})}{k_1(k_2 - k_1)}\right] \\ &= \frac{C(\sigma_{\text{opt}})(k_2 + k_1k_2 - k_1^2) + nk_2(k_1k_2 - k_1^2 - k_2)}{k_1(k_2 - k_1)}. \end{aligned}$$

This is an increasing function of C. Equating the cost of the solution if N is large with this expression we find C_0.

$$k_2 n = \frac{C(\sigma_{\text{opt}})(k_2 + k_1 k_2 - k_1^2) + nk_2(k_1 k_2 - k_1^2 - k_2)}{k_1(k_2 - k_1)}$$

implies $C_0 = k_2^2 n/(k_2 + k_1(k_2 - k_1))$ and the approximation ratio is therefore

$$\frac{k_2 n}{k_2^2 n/(k_2 + k_1(k_2 - k_1))} = \frac{k_2 + k_1(k_2 - k_1)}{k_2} = 1 + \frac{k_1(k_2 - k_1)}{k_2}. \qquad \square$$

5. Probabilistic approach

One of the problems with NP-hardness results is that they only show the existence of particular instances of a problem which under some reasonable assumptions are difficult to solve exactly in time which is polynomial in the input size. In real life such hard instances may never appear as input values. Therefore, it is reasonable to study the complexity of our sequencing problems under the assumption that input instances (expected access times) only appear according to a certain probability distribution.

In this section, we study the VLSP problem in the probabilistic setting by assuming that each value $l(t,i)$ is chosen independently and uniformly at random from a set $S \subset \mathbb{N}^+$. This is equivalent to saying that the input instance is chosen uniformly at random among all those instances with the general completion time and given range set S. Although the model we analyse in this section is perhaps an oversimplification of a realistic setting (e.g., we assume no dependencies/relations between the time required by a task in two consecutive time units), it seems to capture some critical issues and to lead to algorithms which are simple to implement.

In what follows a statement holds *with high probability* if it fails with probability at most $1/n^c$ for some constant $c > 0$. Moreover, in the algorithms presented below, we often use a statement of the form

Let $x = \text{SelectAtRandom}(Y)$,

where x is an integer and Y is a set of integers, with the intended meaning that an element of Y is chosen uniformly at random and assigned to x.

We begin with the simple situation when $S = \{1, \ldots, n\}$. We present an algorithm that for random input with high probability returns a schedule of cost $O(n \ln n)$, such that for each task there exists a time unit in which the task is started and completed.

```
Algorithm A_2
    Let $N = 2n \ln n$ and let $P = T$
    for $i = 1$ to $N$ do
        Let $\mathbf{T}_i$ be the set of tasks $t$ with $l(t,i) = 1$
        if $\mathbf{T}_i \neq \emptyset$ then
            Let $t = \text{SelectAtRandom}(\mathbf{T}_i)$
            if $t \in P$ then
                Let $\sigma(t) = i$
                Let $P = P \setminus \{t\}$
    Let $i = N + 1$.
    for each $t \in P$ do sequentially
        Let $\sigma(t) = i$
        Let $i = i + l(t,i)$
```

Theorem 6. *Algorithm A_2 returns a schedule of cost $2n \ln n$ with high probability.*

Proof. We show that the algorithm terminates in the first loop with high probability. For that, let us define the following "coupon collector's algorithm":

```
Algorithm CC
    Let $\mathbf{B} = \{1, \dots, n\}$
    repeat
        With probability $1 - e^{-1}$
            Let $i = \text{SelectAtRandom}(\{1, \dots, n\})$
            Let $\mathbf{B} = \mathbf{B} \setminus \{i\}$
    until $\mathbf{B} = \emptyset$
```

It is well known (see, e.g., [11, Chapter 3.6]) that with high probability the "coupon collector's algorithm" terminates in less than $1.1 n \ln n 1/(1 - e^{-1}) \leqslant 2n \ln n$ rounds. Now, one can easily show that the size of P after i steps of algorithm A_2 in the first loop is stochastically dominated (i.e., informally, it is not worse in the probabilistic sense) by the size of $\mathbf{B}$ after i steps of the "coupon collector's algorithm" **CC**. This follows from the fact that, since the selections of the tasks t in the first loop of algorithm A_2 are independent,

$$\Pr[\mathbf{T}_i \neq \emptyset] = 1 - \left(1 - \frac{1}{n}\right)^n \geqslant 1 - e^{-1},$$

and hence the probability that a random task is chosen in step i of the first loop in algorithm A_2 is at least $1 - e^{-1}$. Therefore after N steps P is empty with high probability. □

Theorem 6.1 does not seem to give the optimal answer. We are currently trying to prove that if the values $l(t,i)$ are random numbers between 1 and n then with probability approaching 1 for n sufficiently large, there exists an execution sequence of linear cost.

Also notice that it is easy to extend the above algorithm to the case when $S = \{k_1,\dots,k_n\}$, and $1 \leqslant k_1 < k_2 < \cdots < k_n$, to obtain an execution sequence of cost $n(2\ln n + k_1 - 1)$ with high probability.

Now we consider the case $S = \{k_1,k_2,\dots,k_m\}$, for $1 \leqslant k_1 < k_2 < \cdots < k_m$ and $m \leqslant n/3\ln n$. The trivial lower bound for the cost of any execution sequence in this context is $k_1 n$. We prove that there exists an algorithm which, with high probability, returns an execution sequence matching this lower bound.

```
Algorithm A_3
        Create a bipartite graph $G = (V, W, E)$ with
                $V = \{v_1,\dots,v_n\}$
                $W = \{w_1,\dots,w_n\}$
                $E = \{\{v_t,w_i\} : l(t, k_1(i-1)+1) = k_1\}$
        Find a maximum cardinality matching $M$ of $G$
(*)     for each $\{v_t,w_i\} \in M$ do
                Let $\sigma(t) = k_1(i-1)+1$
        Let $i = nk_1 + 1$
        for each task $t$ not scheduled in Step (*) do sequentially
                Let $\sigma(t) = i$
                Let $i = i + l(t,i)$.
```

In the analysis of this algorithm we shall use the following fact.

Fact 7. *Graph G has a perfect matching with high probability.*

Proof. The expected degree of each vertex is at least $3\ln n$ since

$$\Pr[(v_t,w_i) \in E] = \frac{1}{m} \geqslant \frac{3\ln n}{n}.$$

Moreover, a standard application of the Chernoff bounds [8] implies that the minimum degree of G is at least 4 with probability at least

$$1 - \exp\{-(3/2)(\log n - 1)^2/\log n\}.$$

Walkup [12] proved that a random directed bipartite graph with n vertices on each side and outdegree at least 2 has a perfect matching (after removal of the orientations from the graph) with high probability. Since G is a random graph with the minimum degree at least 4, one can easily conclude from the result of Walkup that G has a perfect matching with high probability. □

Once we know that M is a perfect matching with high probability, the following theorem follows immediately.

Theorem 8. *Algorithm A_3 returns a schedule of cost $k_1 n$.*

6. Conclusion

In this paper, we have considered the possibility of using information about known or expected page access times to find an efficient ordering of a sequence of such accesses. Empirical evidence suggests that it may be possible to estimate the likely access time for any particular web document to be fetched at a particular time of day, and that these access times will vary significantly over time, so a good ordering could lead to significant gains for web crawling robots. We have shown that the problem of finding an optimal ordering, in the most general case, is a generalisation of a task scheduling problem which is known to be computationally hard. In practice, however, the precision of access time estimates, based on historical data, is likely to be relatively low. Hence, a reasonable engineering solution may be postulated for cases in which the expected access times are categorised with a coarse granularity. We have shown that in such simplest case, an optimal solution may be computationally feasible. In other cases, some simple algorithms have been identified which may give useful performance. A number of aspects of the analysis remain as open problems: for example, what is the complexity of an instance of VLSP with values 1 and 3, is it NP-hard? If it is, can we find a better approximation than the $\frac{5}{3}$ achieved by our approximation algorithm?

Our analysis, although essentially theoretical, points the way to a number of possible practical implementations. In particular, we wish to investigate two scheduling strategies. In the first strategy, the web crawler would make use of a matrix of expected access times to implement one of the ordering algorithms suggested above. In the second case, access time data would be obtained dynamically while the crawler is in progress, and used to drive a heuristic ordering of tasks. We propose to carry out practical experiments to examine the performance of such strategies in real situations.

Acknowledgements

Our thanks are due to Jon Harvey and Dave Shield for their assistance in the preparation of this paper.

References

[1] A.V. Aho, D.S. Johnson, R.M. Karp, S.R. Kosaraju, C.C. McGeoch, C.H. Papadimitriou, P. Pevzner, Theory of computing: goals and directions, Special Report of the National Science Foundation of the USA, 1996.

[2] N. Christofides, Worst-case analysis of a new heuristic for the travelling salesman problem, Tech. Report TR CS-93-13, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, 1976.

[3] Cyberatlas, How big is the Internet? January 1998, Available at: `http://www.cyberatlas.com`.

[4] D. Eichmann. The RBSE spider – balancing effective search against Web load, Proc. 1st Internat. World-Wide Web Conference, CERN, Geneva, Switzerland, May 25–27, 1994. Elsevier Science, BV, Amsterdam, the Netherlands.

[5] L. Engebretsen, An explicit lower bound for TSP with distances one and two, in: C. Meinel, S. Tison (Eds.), Proc. 16th Annual Symp. on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science, vol. 1563, Trier, Germany, March 4–6, Springer, Berlin, 1999, pp. 1–15.

[6] M.R. Garey, D.S. Johnson, Two-processor Scheduling with Start-times and Deadlines, SIAM J. Comput. 6 (1977) 416–426.

[7] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, New York, NY, 1979.

[8] T. Hagerup, C. Rüb, A guided tour of Chernoff bounds, Inform. Process. Lett. 33 (1990) 305–308.

[9] D.S. Johnson, C.H. Papadimitriou, Computational complexity, in: E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys (Eds.), The Traveling Salesman Problem, Wiley, New York, NY, 1985, pp. 37–85 (Chapter 3).

[10] S. Micali, V.V. Vazirani, An $O(\sqrt{|v|}\cdot|E|)$ algorithm for finding maximum matching in general graphs, Proc. 21st Annual Symp. on Foundations of Computer Science, Syracuse, NY, October 13–15, IEEE Computer Society Press, Los Alamitos, CA, 1980, pp. 17–27.

[11] R. Motwani, P. Raghavan, Randomized Algorithms, Cambridge University Press, New York, NY, 1995.

[12] D.W. Walkup, Matchings in random regular bipartite digraphs, Discrete Math. 31 (1980) 59–64.