Efficient Web Searching Using Temporal Factors

Artur Czumaj * Ian Finch † Leszek Gąsieniec †‡ Alan Gibbons †
Paul Leng † Wojciech Rytter †§ Michele Zito †¶

Abstract

Web traversal robots are used by search engines and in other applications to gather information periodically from large numbers of documents distributed throughout the Web. In this paper we study the issues involved in the design of algorithms for performing information gathering of this kind more efficiently, by taking advantage of anticipated variations in access times in different regions at different times of the day or week. We report and comment on a number of experiments showing a complex pattern in the access times as a function of the time of the day. We argue that these patterns can be used to devise efficient algorithms. Also, 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 for short). 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.

1 Introduction

As the World Wide Web has grown in size and importance as a medium for information storage and interchange, the problem of locating information within it has assumed great significance. In the 1997 GVU user survey [11], 49% of respondents reported difficulty in finding information as an issue. This problem is certain to get worse; it has been estimated that by the start of 1998 there were 100M documents online, and that this may rise to 800M by the year 2000 [5]. The scale and importance of the problem motivates interest in software tools for finding and classifying information effectively, and in algorithms for doing

*Heinz Nixdorf Institute and Dept. of Math. and Comp. Sci., U. of Paderborn, D–33095, Germany, artur@uni-paderborn.de. Research partially supported by EU ESPRIT Long Term Research Project 20244 (ALCOM-IT) and by DFG-Sonderforschungsbereich 376 "Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen. The research of this author was partly done while visiting the University of Liverpool.

†Dept. of Comp. Sci., U. of Liverpool, Peach Str., L69 7ZF, UK, {ian,leszek,amg,phl,rytter, michele@csc.liv.ac.uk}

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

§Instytut Informatyki, Uniwersytet Warszawski, Banacha 2, 02–097, Warszawa, Poland

¶Supported by EPSRC grant GR/L/77089

this efficiently [1]. Of particular importance is the problem of traversing the Web to collect information from documents stored therein.

Software tools for traversing the Web are used for a number of purposes, including Web site maintenance [9] and Web analysis [2]. The major interest, however, is in traversing the Web to discover and reference the information it contains. We are concerned in particular with searching algorithms and efficiency of intelligent searching robots (crawlers or spiders) [6]. Programs of this kind are used by search engines [16] and web indexing services to periodically examine a large subset of the documents stored on the Web.

Conceptually, the World Wide Web may be modelled as a directed graph, and most searching robots proceed by traversing this graph, following the hypertext links embedded in documents [14]. This conceptualisation can obscure an underlying reality, which is that a 'visit' to a node of the structure is effected by transferring a document from the location of the node to the location from which the search is being conducted. The time required to do this is essentially a function only of the relative positions of these two locations within the Internet, and is not related to the structure of the Web-graph itself or to the order in which nodes are visited.

At first sight, this reality suggests that there is little scope for finding more efficient ways of carrying out the traversal. However, it may be possible to obtain faster traversal times by exploiting temporal variations in access speeds. It is subjectively observable, and unsurprising, that 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.

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 the goal is to load them as quickly as possible. We assume also that once the server has started loading from a site it will continue doing so until all data from that site has been obtained.

In the following section we examine some empirical characteristics of Web access speeds, in order to establish a basis for the use of algorithms of this kind. In the rest of the paper we discuss information gathering problems in a more theoretical setting. In Section 3 we define a class of simplified versions of this problem. We prove that under fairly reasonable assumptions some of them are rather difficult to solve exactly in time polynomial in the number of sites to be connected. However, if connection times are coarsely classified as "high" or "low" we prove that information gathering is possible in polynomial time. Sections 5 and 6 describe some approximation results with both worst case and average case performance guarantees. Finally in section 7 we give our conclusions and plans for further work.

2 Temporal variations in Web access speeds

Traffic loads on the Internet can be estimated in a relatively simple and non-intrusive way by performing a `ping` test which records the travel time of a request from source to destination site and back again. Figure 1 shows the result of an experiment in which this test was carried out between the Liverpool University site and a number of other Internet locations. For each test, one Kb of data was sent concurrently to each of the target destinations, and the time recorded. This test was repeated at intervals of 10 seconds for one week during September 1998, about three days of which are illustrated. In Figure 1, each point on a graph represents an average of 200 consecutive observations, i.e. the graphs record the average access times in overlapping periods of about 30 minutes. The destinations for which results are presented are the following academic sites:

`www.warwick.ac.uk (avg1 - UK (Europe))`
`www.rutgers.edu (avg3 - NJ (USA))`
`www.di.unipi.it (avg6 - Italy (Europe))`
`cs.anu.edu.au (avg8 - Australia)`

The lowest graph in Figure 1 is relative to the Warwick site, the highest to the Australian university and the lighter of the other two is the American.

The results illustrated in Figure 1 demonstrate two things. Firstly, it is clear that access times to any particular site can exhibit very considerable temporal variations, even after these have been smoothed by the averaging process described above. Secondly, the pattern of variation in these access times appears, as one would expect, to be influenced by geography, even though this is masked by the influence of loads deriving from the dominant regions, the USA and Europe. Despite this, it can be observed that the peaks for the Australian site tend to precede those for the USA, for example; an Asian site examined showed a similar pattern to the Australian one. The graphs for the USA and Italy sites likewise exhibit different peaks.

These effects are apparent even though the form of the experiment was such as to underestimate the influence of local load factors at the destination sites. Because the experiment was carried out from a single source site, Liverpool, it may be expected that access times were heavily influenced by traffic load factors within the UK. Conversely, the `ping` test is insensitive to load factors at the destination site, as opposed to traffic load. A test involving fetching documents from the destination sites might be expected to demonstrate greater local variation.

Measurements of Web traffic loads using a similar test are carried out systematically by the Andover News Network, and reported at `http://www.internettrafficreport.com/`. Again, a `ping` test is used to measure response times for each target site. In this case, the figures reported are obtained by averaging the times obtained from a number of geographically distributed sources accessing the same destination simultaneously. This procedure will

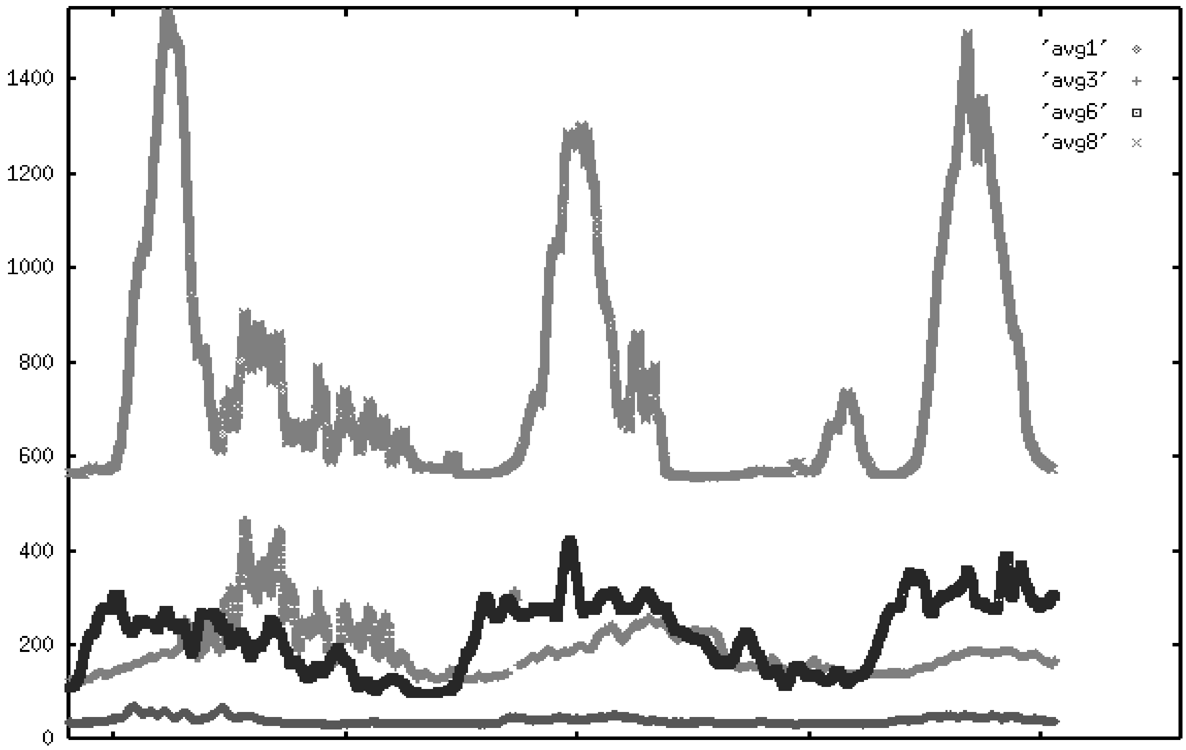

Figure 1: Variations in access times to four sites

tend to reduce the influence of local load factors at each source, but will also tend to reduce peak variations which might be apparent for single paths. The results illustrated at the Andover site are presented as averages for individual server sites and for sites within geographic regions, at 15 minute intervals, summarised daily and for seven-day and 30-day periods.

Again, the results presented demonstrate a number of distinctive patterns. Firstly, it is clear that there are considerable variations in traffic load factors observed at each site and within each geographic region. It is also apparent that these variations tend to recur in a more or less predictable pattern. For example, Figure 2 plots the variation in access rates reported for European servers over two consecutive 24-hour periods, the two graphs being superimposed. The close relationship between these two patterns is apparent. Similar periodic behaviour is evident in the pattern of access speeds over a seven-day period.

Finally, it can also be observed that the variations in access rates show different patterns in different regions. Figure 3 superimposes the plot of variations for sites in Asia on that for those in the USA, for the same 24-hour period. In this case, the different peaks of access speed for each region are clearly apparent.

In summary, these experiments demonstrate that:

- There are significant variations in access times obtained at individual sites on the Web.
- These variations tend to exhibit periodicity and hence may be predictable to an extent.
- The extremes of variation in different geographic areas tend to occur at different times of the day and week.

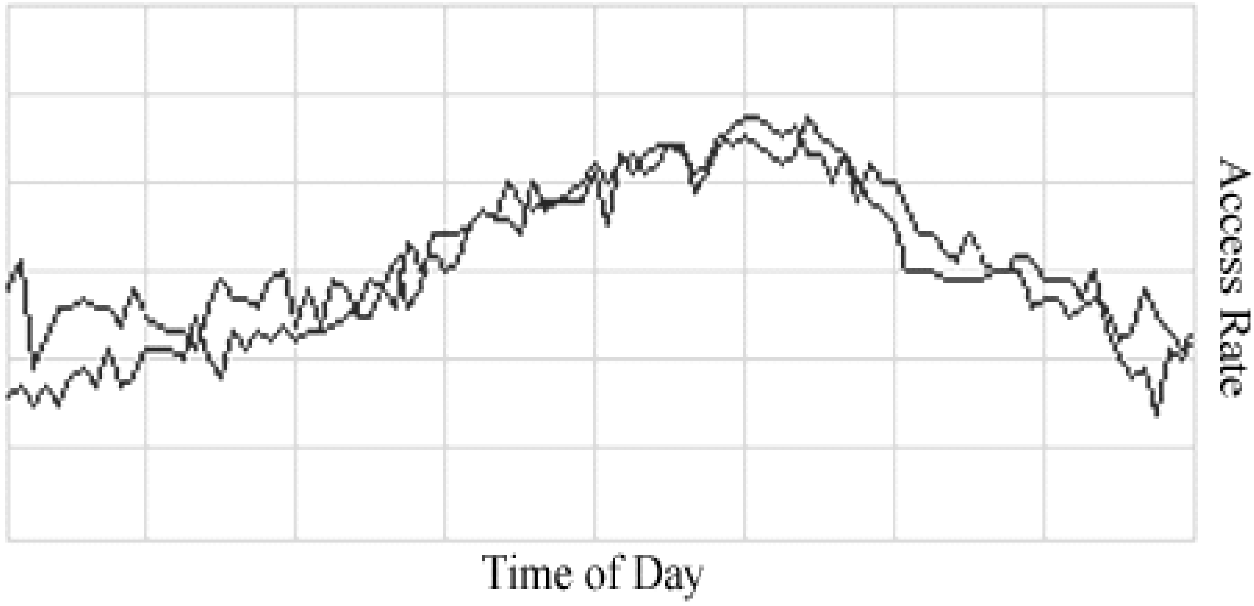

Figure 2: European access rates for two consecutive days

These results provide a basis for the design of algorithms which make use of information on expected access times.

3 Information gathering in a more theoretical setting

The discussion in the preceding section allows us to postulate that, for any particular web document that is to be fetched by a robot operating from a fixed location, it may be possible to predict an expected access time on the basis of historical data. Given that these expected times may vary temporally, we wish to order the set of required document-fetching tasks so as to minimise the total (expected) access time.

More formally, we define the Variable Length Sequencing Problem (VLSP for short) as follows. There are n *tasks* (sets of pages to be collected), each of which will be denoted by an integer in $\{1, \ldots, n\}$ and $N \in \mathbb{N}$ is the *general completion deadline* (all tasks must be completed by time N). For each task t and unit time $i \in \{1, \ldots, N\}$, let $l(t,i) \in \mathbb{N}$ be the *length* (performance time) of task t when started at time i. An *execution sequence* σ is an injective function specifying for each task t a starting time $\sigma(t) \in \{1, \ldots, N\}$ with the property that for every t if $\sigma(t) = i$ then no other task k can have $\sigma(k) \in \{i, \ldots, i+l(t,i)-1\}$. The cost of an execution sequence σ, $C(\sigma)$ is $k + l(t_{\max}, k)$ if $k = \max_{\{1,\ldots,n\}} \sigma(t)$ and $k = \sigma(t_{\max})$. The decision version of VLSP asks whether there exists an execution sequence, such that $C(\sigma) \leq N$.

In this section we prove that VLSP is at least as hard as the problem (SEQUENCING for short) of sequencing a number of tasks on a single computer with *release* times and *deadlines*

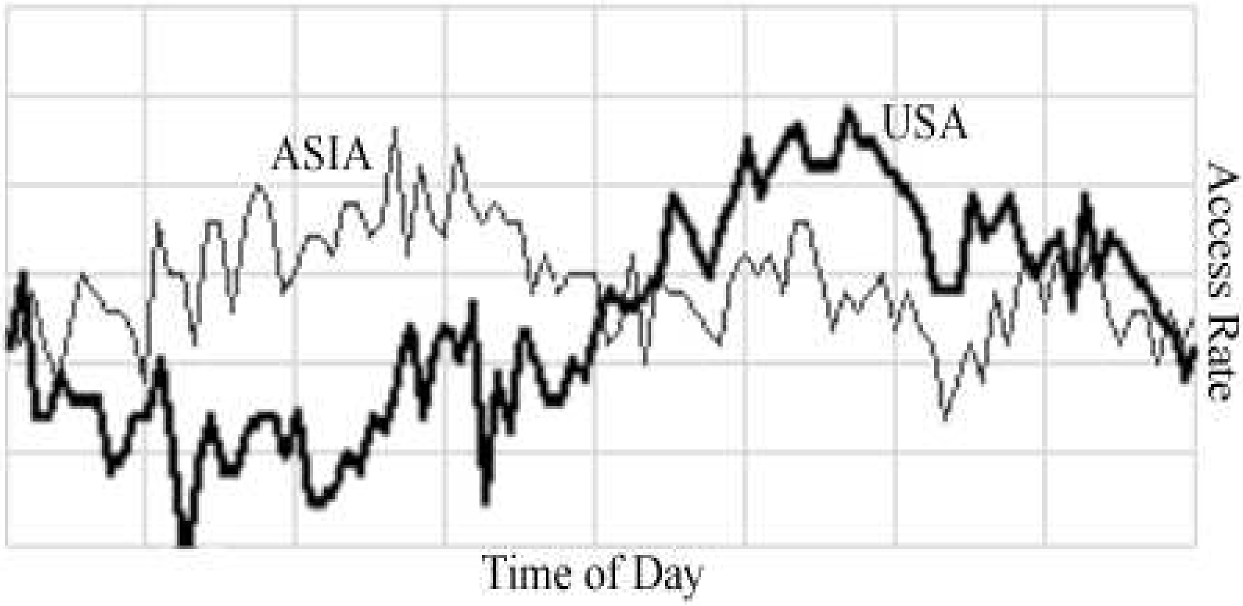

Figure 3: Access rates for USA and Asia over 24 hours

(for detailed definition see [10, p. 238]).

To show this, assume an instance of SEQUENCING with a set of tasks T, where $l(t)$ is the length, $r(t) \in \mathbb{N}$ is the release time, and $d(t) \in \mathbb{N}$ the completion deadline for any task t.

We use the following reduction. 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:

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

Lemma 3.1 *VLSP is at least as hard as SEQUENCING.*

Proof: We show that the solution to any instance of sequencing with deadlines can be obtained from the solution to VLSP. The right hand side of the equality above consists of 3 constraints. The first constraint says that if we start to process task t at any time before its release time $r(t)$ in this instance of the sequencing problem, 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 sequencing problem, it can always be executed at time $r(t)$ in the sequencing problem without causing any delays. The second constraint describes the case in which in both sequencing and VLSP problem execution times of a given task are the same. The third constraint prevents execution of tasks in VLSP when it is too late to do so, i.e. when in the sequencing problem the deadline for task execution is too close. Therefore a solution for VLSP with these constraints will also be a solution for

SEQUENCING. □

Theorem 3.1 *VLSP is NP-complete*

Proof: VLSP belongs to NP since it is easy to verify if a given sequence of tasks can be executed before the deadline for the completion of all tasks. VLSP is NP-complete due to the above reduction and the fact that sequencing with release times and deadlines is NP-complete [10]. □

4 VLSP with slow/fast completion times

Motivated by the intractability of the general setting of VLSP we show that some instances of the problem can be solved optimally in polynomial time. We focus on the possible values of an entry $l(t, i)$. Let $S \subset \mathbb{N}$. We define VLSP(S) to be the class of instances of VLSP with $l(t, i)$ restricted to the set S for all tasks t and $i = 1, \ldots, N$. 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 in "slow" or "fast". In practice, this may be a realistic simplification since estimates based on historical data will necessarily be imprecise.

A similar simplification has been used in the context of the Travelling Salesman Problem (TSP for short); see e.g. [4, 7, 12]. TSP remains NP-hard even in the case when the only legal values for the city distances are 1 and 2. Recently Engebretsen, see [7] 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 $V = \mathcal{T} \cup \mathcal{N}$, where $\mathcal{T} = \{t_1, t_2, .., t_n\}$ corresponds to the set of tasks and $\mathcal{N} = \{1, 2, .., N\}$ represents the sequence of time units. An edge connects nodes t_i and j iff task i can be performed in one step when started at time j.

- Define an *almost* bipartite graph G_N on the same vertex set as B_N with edge set $F = E \cup \{(j, j+1) : j = 1, .., N-1\}$. Edges in E are called *horizontal edges*, all the others are *vertical edges*.

Example. If $N = 7$, $n = 4$ and $l(1, j) = 1$ for all j, $l(2, 1) = 1$ and $l(2, j) = 2$ for all $j > 1$ and $l(t, 2h) = 2$, $l(t, 2h+1) = 1$ for $t = 3, 4$ and all $h = 1, \ldots, (N-1)/2$ the graph B_N is shown in Figure 4.

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

Lemma 4.1 *G_N has a matching of size n if and only if there is a solution to VLSP($\{1, 2\}$) with cost N.*

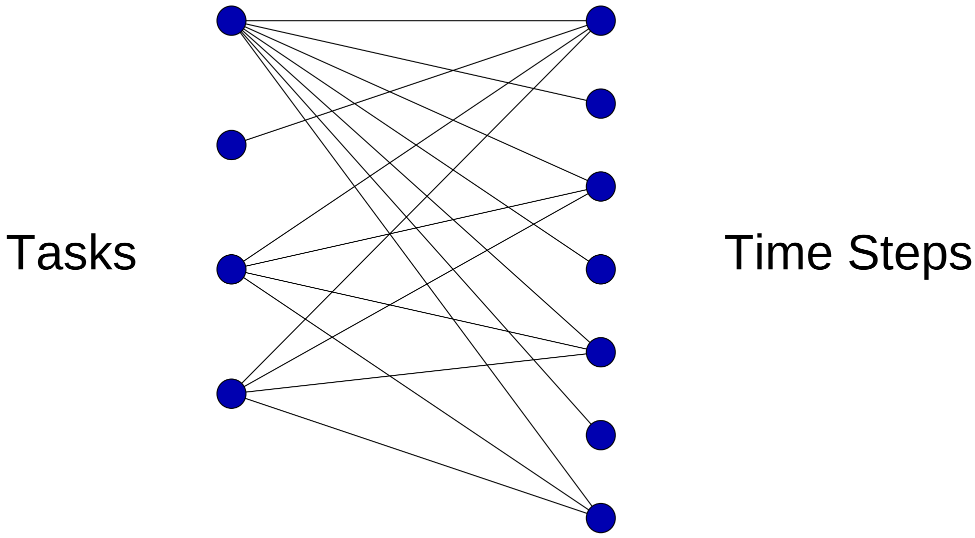

Figure 4: The bipartite graph B_N in the given example.

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 two. $\square$

Theorem 4.1 *The optimal solution to a given instance of VLSP($\{1, 2\}$) can be found 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 [13] for example). $\square$

5 Approximation algorithm for VLSP($\{1, k\}$)

A more general model than the slow/fast access case is obtained if the two possible times can be specified arbitrarily. Unfortunately the complexity of VLSP($\{k_1, k_2\}$) with $k_1, k_2 \in \mathbb{N}$ is open. In this section we study VLSP($\{1, k\}$). We define an algorithm which always finds an execution sequence whose cost is at most twice the cost C^* of the optimal sequence. We prove a technical lemma first.

Lemma 5.1 *The number of tasks of length 1 in any execution sequence σ for an instance of* VLSP($\{1, k\}$) *is* $\frac{nk - C(\sigma)}{k-1}$.

Proof: Let x be the number of 1's and y the number of k's in the execution sequence σ. The result follows since $x + y = n$ (all tasks are processed) and $x + yk = C(\sigma)$ (x tasks processed at speed 1 and y at speed k). $\square$

Theorem 5.1 *For every fixed $k \in \mathbb{N}$, the smallest solution to* VLSP($\{1, k\}$) *can be approximated within constant $2 - \frac{1}{k}$.*

Proof: Let $z = \frac{k^2 n}{2k-1}$. We consider an algorithm which, starting from $N = n$ up to $N = kn$ performs the following computation

1. If $N \leq z$
   (a) create the bipartite graph B_N;
   (b) find a maximum cardinality matching M in B_N;
   (c) all tasks t for which there is an edge $(t, j) \in M$, are executed in time j.
   (d) all remaining tasks are scheduled consecutively in arbitrary order after time N.
2. If $N > z$ simply allocate k steps for each task.

We focus on the iteration when N takes the value of the cost of the optimal execution sequence. If $N > z$ the algorithm will return a solution of length kn. Otherwise, by Lemma 5.1, step (b) will find a matching of size at least $\frac{nk-N}{k-1}$ in B_N. Hence the number of tasks executed with speed k in step (d) is at most $n - \frac{kn-N}{k-1} = \frac{N-n}{k-1}$. The total VLSP completion time generated by the algorithm presented above is at most $N + \frac{N-n}{k-1}k = (N-n)\frac{k}{k-1}$ (which is an increasing function of N).

Thus the worst case ratio between the size of the solution return by the algorithm above and the optimal solution is

$$\frac{kn}{\frac{k^2 n}{2k-1}} = \frac{2k-1}{k} = 2 - \frac{1}{k}$$

□

6 Typical behaviour of the system — probabilistic approach

One of the problems with NP-completeness 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 happen with 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-steps), it seems to capture some critical issues leads 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 $c > 0$.

We begin with the simple situation when $S = \{1, \ldots, n\}$.

We present an algorithm that for random input returns with high probability a schedule of cost $O(n \ln n)$ such that each task is assigned to a time-step where it is executed in a single time-unit.

Algorithm
Let $N = 2\,n\,\ln n$ and P be the set of tasks not yet scheduled. Initially $P = T$.

for $i = 1$ **to** N **do**
  Let $\mathcal{A}_i$ be the set of tasks t with $l(t, i) = 1$
  if $\mathcal{A}_i \neq \emptyset$ **then**
    Pick a task t independently and uniformly at random from $\mathcal{A}_i$
    if $t \in P$ **then**
      Assign task t to time-step i
      Remove t from P

Let $i = N + 1$.
for each $t \in P$ **sequentially**
  Assign task t to time-step i
  Set $i = i + l(t, i)$

Theorem 6.1 *With high probability the algorithm returns a schedule of cost* $2n \ln n$.

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":

Let $\mathcal{B} = \{1, \ldots, n\}$
Repeat until $\mathcal{B} = \emptyset$
  With probability $1 - e^{-1}$
    Pick $i \in \{1, \ldots, n\}$ independently and uniformly at random
    $\mathcal{B} = \mathcal{B} - \{i\}$

It is well known (see, e.g., [15, Chapter 3.6]) that with high probability the "coupon collector's algorithm" terminates in less than $1.1 \cdot n \ln n \cdot \frac{1}{1-e^{-1}} \leq 2n \ln n$ rounds. Now, one can easily show that the size of P after i steps of the scheduling algorithm in the first loop is stochastically dominated (i.e., informally, it is not worse in the probabilistic sense) by the size of $\mathcal{B}$ after i steps of the "coupon collector's algorithm". (This follows from the fact that $\Pr[\mathcal{A}_i \neq \emptyset] = 1 - (1 - \frac{1}{n})^n \geq 1 - e^{-1}$, and hence the probability that a random task is chosen in step t of the scheduling algorithm is at least $1 - e^{-1}$.) Therefore after N steps P is empty with high probability. □

We point out that Theorem 6.1 does not give the optimal answer. If the values $l(t, i)$ are random numbers between 1 and n it is possible to prove that with probability approaching one for n sufficiently large, there exists an execution sequence of linear cost. However we know of no polynomial time algorithm which would output this fast schedule.

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

Now we consider the case $S=\{k_1, k_2, \ldots, k_m\}$, for $1 \le k_1 < k_2 < \cdots < k_m$ and $m \le \frac{n}{3 \ln n}$.

Algorithm

1. Define a bipartite graph $G = (V, W, E)$ with $V = \{v_1, \ldots, v_n\}$, $W = \{w_1, \ldots, w_n\}$, and $E = \{(v_t, w_i) : l(t, k_1 (i-1) + 1) = k_1\}$.

2. Find a maximum cardinality matching $\mathbf{M}$ of G.

3. **For each** $(v_t, w_i) \in \mathbf{M}$ **do**:
   Assign task t to time-step $k_1 (i-1) + 1$.

4. Let $i = n\, k_1 + 1$.
   For each task t not scheduled in Step 3 **do** sequentially:
   Assign task t to time-step i.
   Set $i = i + l(t, i)$.

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

Fact 6.1 *Graph G has a perfect matching with high probability.*

Proof: It is easy to see that since $\Pr[(v_t, w_i) \in E] = \frac{1}{m} \ge \frac{3 \ln n}{n}$, the expected degree of each vertex is at least $3 \ln n$ and the minimum degree of G is at least four with high probability. Walkup [17] proved that a random directed bipartite graph with n vertices on each side and outdegree at least two 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 four, one can easily conclude from the result of Walkup that G has a perfect matching with high probability. $\square$

Once we know that $\mathbf{M}$ is a perfect matching with high probability, the following theorem follows immediately.

Theorem 6.2 *With high probability the algorithm returns a schedule of cost $k_1 n$.*

7 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 the simplest such 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-complete? If it is, can we find 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.

8 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 et al., Theory of Computing: Goals and directions. *Special Report of the National Science Foundation of the USA*, 1996

[2] D. Beckett. 30% Accessible - A survey of the UK Wide Web. In Proceedings of the Sixth international World Wide Web Conference, 1997

[3] B. Bollobas, Random Graphs, Academic Press, 1985.

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

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

[6] D. Eichmann. The RBSE Spider - Balancing Effective search against Web Load. In Proceedings of the first International World Wide Web Conference, 1994.

[7] Lars Engebretsen, An Explicit Lower Bound for TSP with Distances One and Two *ECCC Report TR98-046* also to appear in Proceedings of *16th International Symposium on Theoretical Aspects in Computer Science*, STACS'99.

[8] W. Feller, An Introduction to probability Theory and its Applications, vol. I, Willey, New York, 1950.

[9] R. T. Fielding. Maintaining distributed Hypertext Infostructures: Welcome to MOM-spider's Web. In Proceedings of the first International World Wide Web Conference, 1994.

[10] Michael R. Garey and David S. Johnson, *Computers and Intractability: a Guide to the Theory of Completeness*, Bell Laboratories, Murray Hill, New Jersey, 1979.

[11] Graphic, Visualization, & Usability (GVU) Center's, 8th WWW User Survey, 1997, available at `http://www.cc.gatech.edu/gvu/user_surveys/`

[12] David S. Johnson and Christos H. Papadimitriou, Computational Complexity. In Eugene L. Lawler, Jan K. Lenstra, Alexander H.G. Rinnoy Kan, and David B. Shmoys editors, *The Travelling Salesman Problem*, chapter 3, pages 37–85. John Willey & Sons, New York, 1985.

[13] Micali, S. and Vazirani, V. V., An $O(v^{1/2}e)$ Algorithm for Finding Maximum Matching in General Graphs, Proceedings of the 21st Annual Symposium on Foundations of Computer Science, pp 17–27, 1980.

[14] R.C Miller and K.Bharat. SPHINX: a framework for creating personal, site-specific Web crawlers. Computer Networks and ISDN systems 30, 1998 (proceedings of 7th International World Wide Web Conference)

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

[16] R Selzer, D. Ray and E.Ray. The AltaVista search revolution. McGraw-Hill, 1997.

[17] D. W. Walkup. Matchings in random regular bipartite digraphs. *Discrete Mathematics*, 31:59–64, 1980.