# HAMILTONIAN PATHS ALGORITHMS FOR DISK SCHEDULING

*Giorgio Gallo, Federico Malucelli, Martina Marrè*

Dipartimento di Informatica
Università di Pisa

**Abstract** - The problem of optimally scheduling the read / write requests in a disk storage system is considered. A new class of algorithms for the disk scheduling problem is presented, and the relations between this problem and the shortest hamiltonian path problem on asymmetric graphs are investigated. The problem of deriving realistic upper bounds for the disk utilization factor, one of the main performance measure for the disk scheduling algorithms, is also addressed.

## 1. Introduction

One of the points which has attracted substantial attention in the effort to improve the I/O performance of disk storage systems is the scheduling of read/write accesses. An analysis of several implementations of the basic scheduling algorithms can be found in [Seltzer, Chen & Ousterhout, 1990] and in [Jacobson & Wilkes, 1991].

Different scheduling strategies are often compared from the point of view of the disk utilisation they produce. Another important point is the so called starvation that is the phenomenon of requests having to wait an exceedingly long time to be served.

Here, after having defined the problem in section 2 and presented the basic scheduling algorithms described in the literature in section 3, we present in section 4 a quite general algorithmic paradigm which encompasses almost all the scheduling algorithms considered in the literature.

In section 5 we show that a new class of algorithms can be derived from the presented paradigm; this is the class of the algorithms in which the queue of the requests waiting to be served is partitioned into batches and in each batch an optimal sequencing problem is solved. Such sequencing problem can be stated as the problem to find a shortest hamiltonian path in a asymmetric complete graph, for which an heuristic algorithm is presented.

It is worth noting that the problem of scheduling the requests (disk accesses) in order to maximize the disk utilization can be considered as a "on line" shortest hamiltonian path problem on a asymmetric complete graph, that is a problem in which the graph is not completely known "apriori" but is dynamically updated by addition of new nodes (see for this problem [Bertsimas & Van Ryzin, 1991]). In this perspective, all the disk scheduling algorithms can be viewed as different heuristics for the shortest hamiltonian path problem adapted to the "on line" case.

In order to get a first approximation idea of the relative behaviour of the different algorithms we have compared the shortest hamiltonian path heuristics which underlie the scheduling algorithms, on complete graphs with distance matrices generated according to different distributions; this is done in section 6.

In section 7 a rather important problem is considered, which has not been addressed before in the literature on the disk scheduling problem; this is the problem to find realistic upper bounds for the disk utilisation factor. Although in principle, being an efficiency measure, the disk utilisation factor is bounded by 1, the values which can be obtained in practice are much lower: the simulation reported by Seltzer, Chen & Ousterhout (1990) gives values of the order of .25 for the best performing algorithms. Obviously the value 1 cannot be never attained since some time must in any case be spent in head movements; the amount of this portion of time depends both on the distribution of the requests and on the technical characteristics of the disk. In order to assess the goodness of a given scheduling algorithm it is important to have some measure of the highest realistic value of disk utilisation which can be obtained in a given setting. This point is addressed here by presenting some Linear Programming relaxations of the shortest hamiltonian path problem which can be used to get tight upper bounds of the disk utilisation factor.

**2. Problem definition and technical aspects**

The disk scheduling problem is to find a sequence of the read / write requests on a disk storage system in order to maximize the number of requests processed in the unit of time.

We assume the characteristics of the request stream arriving to the disk to be known. Each *request* can be represented as a tuple ($t$, $s$, $b$, $l$), where $t$ is the arrival time, $s \in \{r, w\}$ tells whether it is a read or a write request, $b$ and $l$ are the address of the first block and the length of the sequence of blocks to be read / written. By $\nu$ we denote the average number of blocks in the requests; clearly $\nu$ depends on the type of users of the computing system.

We assume the requests to be addressed to a single disk unit which is defined by a set of technical data. The most relevant technical data are: the number of *cylinders*, the number of *tracks per cylinder*, the number of *sectors per track*, the number of *bytes per sector*, the *rotation speed*, the *seek time function*, $\varphi$, which, for any pair of cylinders, gives the time needed for the head to move from one to the other, and the time needed to switch from a track to another in the same cylinder (*settle time*). Parameters which give a rough idea of the technical characteristics of a given disk are the rotation time $\rho$, the average

seek time $\sigma$ and the time $\tau$ needed to read/write a single block (times are given in milliseconds). The *disk utilization factor*, $\theta$, is a measure of the efficiency of a given requests processing policy defined as the ratio between the time used to read/write and the overall time needed to process the requests.

Since it may happen that the time distance between two successive requests be much shorter than the time needed to process them, as it is the case when the requests are placed in a buffer cache, rather long queues may be present at the disk input. So a good scheduling algorithm may have a strong effect on the overall performance of the disk.

The problem which will be considered in the following is to schedule the requests currently in the waiting line so as to maximize the disk throughput. In doing that we have to keep in mind the necessity of avoiding the *starvation* phenomenon which happens when a request is left waiting for an exceedingly long time in the queue.

**3. Scheduling algorithms**

Different scheduling algorithms have been proposed and studied in the literature ([Seltzer, Chen & Ousterhout, 1990]; [Jacobson & Wilkes, 1991].

The simplest algorithm one can think of is FCFS (First Come First Serve); this algorithm processes the requests in the same order they arrive without trying to do any optimization. If $\tau$ is the average time needed to read/write the sequence of blocks specified by a single request, the disk utilization factor is given by:

$$\theta = \frac{\nu\tau}{\nu\tau+\rho/2+\sigma}.$$

In the case of the HP 97560 disk unit, and if the requests are uniformly distributed among the cylinders we have $\theta = 8.7\%$.

A less simplistic algorithm, which has been widely used in practice, is the so-called SCAN scheduling algorithm. SCAN orders the requests by cylinder number and processes them in that order starting from the first one. When the last cylinder has been reached the head changes direction sweeping the cylinders in inverse order and processing all the requests arrived in the meanwhile. Then, once the first cylinder has been reached, a new complete cycle is performed; a shortcoming in this algorithm is that requests on the

edges of the disk have a waiting time variance which is almost twice the waiting time variance for the requests on the middle cylinders. A slightly different version, called CSCAN, guarantees a fairer distribution of the waiting times; in it the head, once the last cylinder has been reached, moves back to the first with a large seek and starts again processing the requests with the same policy.

For the HP 97560 unit, the disk utilization factor of SCAN is given by

$$\theta = \frac{\nu\tau}{\nu\tau+\rho/2+\varphi(1963/L)},$$

where $L$ is the average number of requests processed per sweep.

SSTF (Shortest Seek Time First) policy, each time, selects from the queue the request that needs the shortest seek from the current head position. This selection rule can be easily implemented maintaining the requests ordered by cylinder. It usually gives better results than the SCAN and CSCAN policies, however this policy is not starvation free and, under certain load conditions, it can yield very high maximum response times.

Geist and Daniel (1987) proposed the parametric policy V($r$); this policy puts together the ideas of SCAN and SSTF. Let *max_seek* be the time needed by the head to perform a full stroke seek (i.e. *max_seek*=$\varphi(C)$). At each step V($r$) selects the request that minimizes the function given by the seek time, if the head moves along the direction of the last performed seek, the seek time plus $r$ times *max_seek*, otherwise. It is easy to see that when parameter $r$=1, V($r$) is SCAN and when $r$=0, it corresponds to SSTF.

SATF (Shortest Access Time First) is an algorithm which, at each step, selects, among the queued requests, the one which has the minimum I/O time, that is the one which has minimum seek plus rotation time. This policy gives rise to higher disk utilization factors than the ones examined before, but the starvation phenomenon may arise. The disk utilization factor grows with the queue length. Notice that, since the selection of the request to be processed next is performed while the request previously selected is being processed, there is a limit to the length of the queue which can be examined in order to find the minimum time request. Disk utilization factors of .40 have been found in the simulation reported in [Seltzer, Chen & Ousterhout, 1990].

A variant of this algorithm is GSATF ( Grouped Shortest Access Time First), where the cylinders are partitioned in groups of contiguous cylinders, and within each group SATF is applied; the algorithm examines one group after the other. With such a policy the maximum response time is reduced so making less crucial the starvation phenomenon.

Another variant of SATF is WSATF (Weighted Shortest Time First). In this algorithm the request which is selected is the one with minimum weighted time, $T_w$, defined as follows:

$$T_w = T_{i/o} \, (Max - Elap) / Max,$$

where $T_{i/o}$ is the actual I/O time, *Max* is the maximum allowed response time and *Elap* is the time elapsed since the requests was inserted in the queue. This policy at the cost of some bad seeks is quite efficient in avoiding the starvation phenomenon.

The results of a simulation study aimed to analyzing the relative behavior of these algorithm is reported in [Seltzer, Chen & Ousterhout, 1990] and in [Jacobson & Wilkes, 1991].

**4. A general scheduling algorithm**

Here we present an approach to the disk scheduling problem which includes as particular cases the algorithms described in the previous section, and which may lead to a host of different algorithmic implementations. This new approach is described by the following general algorithmic paradigm, GDSA (General Disk Scheduling Algorithm):

*GDSA*

1) Extract, using some given rule *R*, *L* requests from the current queue *Q* of requests, with $L = \min\{|Q|, L_{max}\}$ and $L_{max}$ a given integer parameter ≥1).

2) Find the ordering of the requests selected in order to minimize the total processing time (seek plus rotation times);

3) If the queue is empty pass the complete sequence to the disk unit for processing; else, pass to the disk unit the first $\lceil \alpha L \rceil$ requests in the sequence, where $\alpha \in [0,1]$ is a parameter to be chosen a-priori, and insert again in *Q* the last $L - \lceil \alpha L \rceil$. Go to step 1.

Notice that, at each iteration of the algorithm, the computations needed to find the optimal ordering of the currently selected requests are performed while the disk unit processes the requests selected at the preceding iteration;

then, *L*, at each step, should be chosen in such a way that the *cpu* time needed to solve the optimal sequencing problem be not larger than the time needed to process the requests selected at the preceding iteration.

Variants of the basic algorithm can be obtained by allowing the insertion in the currently processed sequence of newly arrived requests if that can be done at (almost) zero cost, i.e. if the new request can be processed during the rotation latency between two consecutive requests already in the sequence.

In the version of the algorithm described above the starvation phenomenon may occur. The basic algorithm can be modified in order to take into account starvation by forcing the process of the requests whose age (time of permanence in *Q*) is larger than a fixed value, say $t^*$.

It is easy to see that the algorithms presented in the previous section are all implementations of our general method:

*i*) *FCFS* is simply obtained from *GDSA* by setting $L_{max} = 1$, $\alpha = 1$ and *R*="take always the element which is the eldest in *Q*";

*ii*) *SCAN* and *CSCAN* are obtained from *GDSA* by setting $L_{max} = +\infty$, $\alpha=1$ and *R* = "take all the elements whose position is in the first non empty cylinder starting from the current one" (*L* is the number of such elements); the two algorithms differ in the order in which the cylinders are scanned;

*iii*) *SSTF* is obtained from *GDSA* by setting $L_{max} = 1$, $\alpha = 1$ and *R* = "take all the elements whose cylinder is nearest to the current head position"; *V*(*r*) is obtained in a similar way;

*iv*) *SATF* is obtained from *GDSA* by setting $L_{max} = 1$, $\alpha = 1$ and *R* = "take the element whose position in the disk is the nearest to the current head position"; similarly one can obtain *GSATF* and *WSATF*.

In all these implementation there is no real need of the optimal sequencing computations of step 2; in fact in all algorithms but SCAN and CSCAN the set of elements to be sorted has cardinality 1, and in SCAN and CSCAN there is an optimal trivial solution given by the order in which the blocks pass under the reading / writing head.

Now, we consider the class of those algorithms which can be derived from GDSA by selecting at step 1 a set of requests of cardinality greater than 1 in such a way that the problem of finding an optimal sequence for them is not trivial. Clearly, the most costly operation in algorithm of this type is the optimal sequencing computation. This problem can formulated as a shortest

Hamiltonian path on a properly defined graph, as it will be shown in the next section. Being such a problem a rather difficult one, in practice, instead of exact algorithms, heuristic methods will be used.

**5. Shortest Hamiltonian path algorithms**

Let the integers 1, 2,… , $L$ denote the requests selected at the current iteration and $G = (N, A)$ be the graph whose nodes are the requests together with a node, 0, representing the last request of the previous sequence, $N = \{0, 1, 2,\ldots , L\}$, and whose arcs are all the pair $(i,j)$ with $i,j \in N$, and $i \neq j$. Each arc $(i,j)$ is assigned a real weight, $w(i,j)$, equal to the minimum time needed for the head to move from the last block to be read / written when processing request $i$ to the first block to be read / written in order to process request $j$.

The problem of optimally sequencing the requests is equivalent to the problem of finding a Hamiltonian path of minimum total weight starting at node 0, where a Hamiltonian path is one which passes through each node exactly once. This problem is well known to belong to the class of NP-hard problems, so little hope there is to find a polynomial algorithm for its solution. On the other side good heuristic algorithms are available; in the following we shall describe some algorithms of this type.

Most often, the shortest Hamiltonian path problem has been considered in the literature as a variant of the more studied Travelling Salesman problem, that is the problem of finding a Hamiltonian cycle of minimum weight in a graph. Actually algorithms that solve the latter problem can easily be adapted to solve the former too.

*Shortest Hamiltonian Path (SHP)*

1) Find, by means of an Assignment algorithm, a shortest cycle cover of the nodes of $G$. Let $C_1, C_2, \ldots, C_k$ be the cycles obtained, where each cycle is defined by a set of nodes and a predecessor function $p(.)$ We assume w.l.o.g. node 0 to belong to cycle $C_1$.

2) Define $I = \{ 2, 3, \ldots, k\}$ and $u = p(0)$. While $I \neq \emptyset$ do: find an index $i \in I$ and a node $v \in C_i$ such that $w(u, v) - w(p(v), v)$ be minimum, set $s = p(v)$, $p(v) = u$ , $u = s$ and $I = I \setminus \{i\}$.

3) Return the sequence defined by the function $p(.)$ with node 0 as the first node.

The complexity of this algorithm is $O(n^3)$; in fact the cost of step 1 (the assignment routine) is $O(n^3)$ being the graph complete, while the overall cost of step 2 is $O(n^2)$.

SHP can be refined considering the following variant of step 2):

2') Define $I = \{ 2, 3, \ldots, k\}$ and $C = C_1$. While $I \neq \emptyset$ do: find an index $i \in I$, a node $u \in C$ and a node $v \in C_i$ such that $w(p(u),v) + w(p(v),u) - w(p(v),v) - w(p(u),u)$ be minimum, set $s = p(v)$, $p(v) = u$ , $C = C_i$ and $I = I \setminus \{i\}$.

The complexity of this algorithm (SHP') is unchanged even though the complexity of step 2') is increased.

**6. Performance of Shortest Hamiltonian Path heuristics**

In this first phase of computational experiments, some of the most known heuristics for shortest Hamiltonian path have been compared; we considered asymmetric distance matrices whose elements have been generated with uniform, exponential and normal distributions. These tests do not take into account the dynamic arrival of requeststhe other peculiarities of out problem. In the following tables, the algorithms SHP and SHP' presented in the previous section, are considered. Column NN refers to the classical greedy algorithm that, at each step, selects the nearest node among the neighboring unvisited ones. Algorithm NN has been improved including a local search phase that, once the path has been constructed, tries to decrease the path length by means of exchanges of pair of nodes in the sequence (NN+exch.). Problems with 100, 200, 500 and 1000 nodes have been considered. For each size the average path length over ten problems is reported. The distance matrices have been generated according to the following distributions:

- uniform in the set {0,…,30};
- exponential with mean $\lambda$=10;
- normal with mean $\mu$=10 and standard deviation $\sigma$=5;
- normal with mean $\mu$=14 and standard deviation $\sigma$=5;

| | **size** | **SHP** | **SHP'** | **NN** | **NN+exch.** |
|---|---|---|---|---|---|
| Unif {0,…,30} | 100 | 23.5 | 19.4 | 82.2 | 56.7 |
| Unif {0,…,30} | 200 | 13.9 | 5.7 | 89.3 | 59.5 |
| Unif {0,…,30} | 500 | 23.8 | 8.3 | 84.1 | 53.9 |
| Unif {0,…,30} | 1000 | 23.8 | 6.8 | 86.3 | 39.0 |

table 6.1

| | size | SHP | SHP′ | NN | NN+exch. |
|---|---|---|---|---|---|
| exp λ=10 | 100 | 5.2 | 3.7 | 25.0 | 14.0 |
| exp λ=10 | 200 | 4.2 | 0.1 | 21.7 | 12.2 |
| exp λ=10 | 500 | 7.1 | 1.5 | 24.3 | 7.4 |
| exp λ=10 | 1000 | 10.8 | 4.3 | 27.5 | 13.3 |

table 6.2

| | size | SHP | SHP′ | NN | NN+exch. |
|---|---|---|---|---|---|
| N: μ=10, σ=5 | 100 | 67.7 | 68.6 | 119.8 | 111.5 |
| N: μ=10, σ=5 | 200 | 42.7 | 41.1 | 138.8 | 121.7 |
| N: μ=10, σ=5 | 500 | 9.2 | 4.9 | 153.8 | 127.2 |
| N: μ=10, σ=5 | 1000 | 15.7 | 10.3 | 160.4 | 130.1 |

table 6.3

| | size | SHP | SHP′ | NN | NN+exch. |
|---|---|---|---|---|---|
| N: μ=14, σ=5 | 100 | 269.0 | 266.4 | 348.9 | 336.5 |
| N: μ=14, σ=5 | 200 | 351.6 | 350.6 | 516.7 | 487.5 |
| N: μ=14, σ=5 | 500 | 376.8 | 374.0 | 694.8 | 676.2 |
| N: μ=14, σ=5 | 1000 | 291.2 | 286.6 | 322.2 | 301.7 |

table 6.4

The results emphasize that, when uniform, exponential and normal distributions are considered, SHP and SHP' outperform Nearest Neighbor, also when the latter includes a local search phase. The global percent gap between NN+exch. and SHP' is 420%, 389%, 293%, 41%, for tables 6.1, 6.2, 6.3 and 6.4, respectively. Very often SHP and SHP' give the optimal solution (path length equal 0) especially when the size of the problem is sufficiently large.

In a second set of experiments, distance matrices which take into account some of the characteristics of the disk scheduling problem have been used. Table 6.5 refers to problems obtained by generating, randomly with uniform distribution, the location in the disk surfaces of the requests and then by computing the distances based on the technical characteristics of the disk (HP 97560) Table 6.6 refers to problems obtained from real life traces of requests.

In both cases we assume that all the requests arrive at the same time; in fact our goal here is only to compare the shortest Hamiltonian path algorithms studying the effects of the type of distances used on their relative behavior.

| size | SHP | SHP′ | NN | NN+exch. |
|---|---|---|---|---|
| 100 | 904.0 | 876.9 | 923.2 | 923.8 |
| 200 | 1600.8 | 1449.6 | 1647.4 | 1639.0 |
| 500 | 3448.2 | 3353.6 | 3581.8 | 3573.5 |
| 1000 | 6075.0 | 5930.0 | 6390.8 | 6365.4 |

table 6.5

| size | SHP | SHP′ | NN | NN+exch. |
|---|---|---|---|---|
| 100 | 555.2 | 534.1 | 562.9 | 560.1 |
| 200 | 1099.5 | 1081.9 | 1154.8 | 1150.5 |
| 500 | 2401.9 | 2371.0 | 2550.3 | 2537.0 |
| 1000 | 4456.0 | 4405.4 | 4830.6 | 4820.4 |

table 6.6

The results point out that the differences between SHP and Nearest Neighbor algorithms are not so evident as it was in the previous case. The global percent gap between SHP' and NN+exch. is 7.7 and 8.1 in tables 6.5 and 6.6, respectively. Moreover, it should be noted that, when real traces are considered, the length of the path can be much shorter that in the uniformly generated cases, due to the “locality” of requests.

The results presented in this section are quite interesting as far as our problem is concerned. In fact, algorithm NN is nothing more than SATF. Namely, they use the same strategy, the only difference being in the problem they are intended for: SATF is applied to a shortest hamiltonian path problem in which the graph is incrementally constructed by adding one node at time.

The experimental results show that with distance matrices that are similar to the ones derived from the disk scheduling problem, NN, which is in general a poor heuristic, becomes quite good. That suggests that the improvements which one can espect from more sophisticate heuristics may not balance the increased computational cost.

## 7. Determination of lower bounds

A crucial point for the Disk Scheduling problem seems to be the determination of lower bounds that can give a measure of algorithms efficiency. In particular, if we can estimate from below the time needed to access data (seek plus rotation time) for a given sequence, we can give a measure of the best disk utilization factor for that sequence. In the following

we will propose several mathematical models that provide a lower bound of the total time in accessing data for a given request sequence, and, as a consequence, an upper bound for the disk utilization factor. All the proposed models assume that the inter-arrival time is not influenced by the sequence of served requests and that the arrival times of the requests are known in advance.

Let $d_{ij}$ represent the time needed to move the head from the ending block of request $i$ to the first block of request $j$, and $\Pi$ be the rotation period of the disk (i.e. $\Pi=1/\rho$). Moreover let $s_i$ denote the time needed to read or write request $i$, $a_i$ denote the arrival time of request $i$ and $\delta_i$ be the time needed by the head of the disk to reach the beginning of request $i$ considering a null seek time (i.e. $\delta_i$ is the min waiting time for request $i$). Let us fix $K$ as the maximum number of disk revolutions a request is allowed to wait in the queue; as a consequence for each request $i$ we can define a due date $b_i=a_i + \delta_i + K\Pi$. Moreover let $T$ be an upper bound of the completion time of the whole sequence of requests.

*Integer linear programming formulation*

Consider a graph $G = (N,A)$ where $N=\{0,\ldots,n+1\}$ each node represents a request; 0 and $n+1$ are two dummy requests that represent the beginning and the end of the sequence processing, respectively ($a_0=s_0=s_{n+1}=0$, $a_{n+1}=T$). There is an arc $(i,j)\in A$ if two requests $i$ and $j$ can be processed consecutively (i.e. $b_j \geq a_i + \delta_i + s_i + d_{ij}$). The *forward star* and *backward star* of node $i$ are defined as $FS(i)=\{j\in N,(i,j)\in A\}$, $BS(i)=\{j\in N,(j,i)\in A\}$, respectively (obviously $BS(0)=FS(n+1)=\emptyset$). A 0-1 variable $x_{ij}$ is associated to each arc $(i,j)\in A$; $x_{ij}=1$ iff request $i$ is processed immediately before request $j$. Variable $t_i$, denotes the starting time of execution of request$i$, $i\in N$,. Note that if $G$ is not connected, we can identify an independent scheduling problem for each connected component, and solve it separately. Disk Scheduling constraints can be expressed as follows:

$$\sum_{j\in FS(i)} x_{ij} = 1, \qquad i=0,\ldots,n, \qquad (7.1)$$

$$\sum_{i\in BS(j)} x_{ij} = 1, \qquad j=1,\ldots,n+1, \qquad (7.2)$$

$$t_i + s_i + d_{ij} \leq t_j + T(1-x_{ij}), \qquad \forall (i,j)\in A, \qquad (7.3)$$

$$a_i + \delta_i \leq t_i \leq b_i, \qquad i=0,\ldots,n, \qquad (7.4)$$

$$x_{ij}\in\{0,1\}, \qquad \forall (i,j)\in A,$$

$$\rho(t_i-a_i-\delta_i)\in \mathbf{Z}. \qquad i=0,\ldots,n+1.$$

Constraint (7.3) states that if request $i$ and $j$ are processed consecutively then the difference between $t_i$ and $t_j$ must be greater that or equal to the time needed to execute request $i$ plus the distance $d_{ij}$; in practice these constraints play the role of subtour elimination constraints in usual TSP formulations. Moreover constraints (7.4) together with integrality of $\rho(t_i-a_i-\delta_i)$ state that requests can be accessed only at fixed periodic instants.

Several objective functions can be defined depending on the chosen optimization criterion. In the case of disk utilization factor maximization we are interested in minimizing the total sum of seek plus rotation times. Hence the objective function is:

$$\min \sum_{(i,j)\in A} d_{ij}\, x_{ij}.$$

On the other hand, if we want to arrange the sequence in such a way that the average response time is minimized, the objective function turns out to be the following:

$$\min \sum_{i\in N} t_i - (a_i + \delta_i).$$

Finally if we want to minimize the maximum response time, we need to introduce a new variable $r$ and the following constraints:

$$t_i - (a_i + \delta_i) \le r, \qquad i=1,\ldots,n,$$

and the objective function to minimize is $r$.

The continuous relaxation of the problem defined by constraints (7.1),…,(7.4) and any one of the above objective functions yields a lower bound for the Disk Scheduling problem. The size of the problem is quite reasonable ($O(n^2)$ constraints and $O(n^2)$ variables), and it can be efficiently solved by means of Linear Programming techniques. Let us call $ub1$ the disk utilization factor upper bound obtained solving the previous continuous relaxation.

*Flow model*

Consider the graph $G = (N, A)$ defined above. Flow variables $z_{ij}$ are associated to each arc $(i,j)\in A$ and they are such that the total amount of flow exiting from each node $i$ gives the point in time at which request $i$ is completed (i.e. $\sum_{j\in FS(i)} z_{ij} = t_i + s_i$). At each node the flow increases at least of $s_i + d_{ji}$, where $j$ is the request that precedes $i$ in the sequence. Problem constraints are the following:

$$\sum_{j \in FS(i)} x_{ij} = 1, \qquad i=0,\ldots,n, \qquad (7.5)$$

$$\sum_{i \in BS(j)} x_{ij} = 1, \qquad j=1,\ldots,n+1, \qquad (7.6)$$

$$t_i \leq \sum_{j \in FS(i)} z_{ij} - s_i, \qquad i=0,\ldots,n, \qquad (7.7)$$

$$t_i \geq \sum_{j \in BS(i)} z_{ji} + \sum_{j \in BS(i)} d_{ji}\, x_{ji}, \qquad i=1,\ldots,n+1, \qquad (7.8)$$

$$a_i + \delta_i \leq t_i \leq b_i, \qquad i=0,\ldots,n, \qquad (7.9)$$

$$z_{ij} \leq T x_{ij}, \qquad \forall (i,j) \in A, \qquad (7.10)$$

$$x_{ij} \in \{0,1\}, \qquad \forall (i,j) \in A,$$

$$\rho(t_i - a_i - \delta_i) \in \mathbf{Z}. \qquad i=0,\ldots,n+1.$$

As in the previous case, several objective functions can be defined. In the case of disk utilization factor maximization we have:

$$\min \sum_{(i,j) \in A} d_{ij}\,(x_{ij}).$$

Considering average response time minimization, the objective function is:

$$\min \sum_{i \in N} t_i - (a_i + \delta_i).$$

Finally if we want to minimize the maximum response time, we need to introduce a new variable $r$ and the following constraints:

$$t_i - (a_i + \delta_i) \leq r, \qquad i=1,\ldots,n,$$

and the objective function to minimize is $r$.

Also in this case, the continuous relaxation of the problem defined by constraints (7.5),…,(7.10) and any one of the above objective functions yields a lower bound for the Disk Scheduling problem. The size of the problem is $O(n^2)$ constraints and $O(n^2)$ variables, and it can be efficiently solved by means of Linear Programming techniques.

Let us call *ub*2 the disk utilization factor upper bound obtained solving the previous continuous relaxation.

*Shortest Path with side constraints formulation*

Consider the directed graph $G'=(N',A')$ where $N'$ is given by a source $s$, a sink $t$ and the nodes $h=(i,t_i)$ that represents a request $i$ and a possible accessing time $t_i$ for it (i.e. $a_i + \delta_i \leq t_i \leq b_i$, and $t_i / \Pi \in \mathbf{Z}$). The number of nodes is $O(Kn)$. There is an arc $(h,k) \in A'$ (with $h=(i,t_i)$, $k=(j,t_j)$) if $i \neq j$ and $t_i + d_{ij} + s_i \leq t_j$, and either $t_j - \Pi < a_j$ or $t_i + d_{ij} + s_i > t_j - \Pi$; the length $l_{hk}$ associated to arc $(h,k)$ is

$d_{ij}$. Moreover $(s,h)$ and $(h,t)$ are in $A'$ for each $h \in N'$, $h \neq s,t$, and their length is null. Let us call *level* $l_i$ the set of nodes related to request $i$. The number of arcs is bounded by $O(Kn^2)$. Note that $G'$ is acyclic.

The problem of finding the rearrangement of the sequence that gives the best utilization factor can be reduced to finding the shortest path between the source $s$ and the sink $t$ that selects exactly one node from each level. The problem can be formulated as follows:

$$\min \sum_{(h,k)\in A'} d_{hk}x_{hk}$$

$$\sum_{h\in \mathrm{BS}(k)} x_{hk} - \sum_{h\in \mathrm{FS}(k)} x_{kh} = 0 \qquad \forall k \in N',\ k \neq s,t,$$

$$\sum_{h\in \mathrm{FS}(s)} x_{sh} = 1$$

$$\sum_{h\in \mathrm{BS}(t)} x_{ht} = 1$$

$$\sum_{h\in l(i)} \sum_{k\in \mathrm{BS}(h)} x_{ht} = 1 \qquad i=1,\ldots,n.$$

$$x_{hk} \in \{0,1\}, \qquad \forall (h,k) \in A'.$$

Two relaxations of the above problem can be considered. The first relaxation SP1 is obtained by dropping integrality constraints and solving the resulting Linear Programming problem.

Another lower bound SP2 is given by the following Lagrangean relaxation:

$$SP2 = \max\{L(\lambda): \lambda \in \mathbf{R}^n\}$$

$$L(\lambda) = \min \sum_{(h,k)\in A'} d_{hk}x_{hk} + \sum_i \lambda_i \left( \sum_{h\in l(t)} \sum_{k\in \mathrm{BS}(h)} x_{ht} - 1 \right)$$

$$\sum_{h\in \mathrm{BS}(k)} x_{hk} - \sum_{h\in \mathrm{FS}(k)} x_{kh} = 0 \qquad \forall k \in N',\ k \neq s,t,$$

$$\sum_{h\in \mathrm{FS}(s)} x_{sh} = 1$$

$$\sum_{h\in \mathrm{BS}(t)} x_{ht} = 1$$

$$x_{hk} \in \{0,1\}, \qquad \forall (h,k) \in A'.$$

Note that the evaluation of $L(\lambda)$ for a fixed $\lambda$ can be carried out solving a Shortest Path problem on an acyclic graph that can be done very efficiently in $O(|A'|)$ time. Moreover, since $L(\lambda)$ has the integrality property, according to [Geoffrion, 1974] SP2=SP1.

## 8. Preliminary experimental results

In the experiments, requests obtained from real traces have been considered. The real traces have been provided by the CSP group of Hewlett Packard Laboratories (Palo Alto).

In the following table we compare the disk utilization upper bound *ub*1 and *ub*2 defined in the previous section, with the solution yielded by SHP' and the optimal disk utilization factor, for request sequences of increasing length. Parameter *K* has been fixed to 200.

| **size** | ***ub*1** | ***ub*2** | **SPH'** | **opt** |
|---|---|---|---|---|
| 5 | .25 | .25 | .25 | .25 |
| 10 | .24 | .24 | .20 | .22 |
| 15 | .25 | .25 | .20 | - |
| 20 | .28 | .28 | .20 | - |
| 30 | .27 | .27 | .18 | - |
| 50 | .27 | .27 | .16 | - |
| 100 | .30 | .25 | .14 | - |
| 150 | .33 | .26 | .14 | - |

table 8.1

For sequences with more than 10 elements it has not been possible to solve exacly the problem in a reasonable time.

The results show that when the size is sufficiently large *ub*2 is tighter than *ub*1. When the size of the sequence is small the boound seems to be fairly close to the optimal value; however for large sequences the upper bound is rather far from the value given by the heuristic SHP'.

## 7. Conclusions and further developments

The contribution of this paper is threefold: (*i*) we have introduced a new class of algorithms for the disk scheduling problem; (*ii*) after pointing out that the disk scheduling algorithms are based on some heuristics for the shortest hamiltonian path problem, we have tried to compare the performance of such heuristics under different hypotheses on the distance metric; (*iii*) we have addressed the problem of deriving realistic upper bounds for the disk utilization factor.

The experimentation presented in the paper is still uncomplete; for instance, the Linear Programming models presented need further study since they lead

to exceedingly large scale problems as the number of requests increases. The development of special algorithms based on Lagrangean relaxation for the third model presented in section 7 is currently under study and will be presented in a forthcoming paper.

Although still preliminary, the results obtained may lead to some first approximation conclusions. A first conclusion is that as the distance metric approaches the ones which can be derived from real life problems the differences in the performances of the different algorithms seem to almost disappear. This result is consistent with the results from a large simulation described in a joint paper of one of the authors, which is currently in preparation [Cao, Malucelli & Wilkes, 1993]. A second conclusion, or better a conjecture which can be derived from the first upper bounds on the disk utilization obtained is that the best algorithms under study (not the ones used in practice!) yield disk utilization values that are not too far from optimality.

## References


D. J. Bertsimas and G. Van Ryzin, "A stochastic and dynamic vehicle routing problem in the Euclidean plane", Operations Research, 39 (1991) 601-615.

P. Cao, F. Malucelli, J. Wilkes, "An experimental comparison of disk scheduling algorithms", in preparation.

R. Geist and S. Daniel, "A continuum of Disk Scheduling Algorithms", ACM Transactions on Computer Systems, 5 (1987) 77-92.

A. M. Geoffrion, "Lagrangean relaxation for integer programming", Mathematical Programming Study, 2 (1974) 82-114.

D. M. Jacobson and J. Wilkes, "Disk Scheduling algorithms based on rotational position", Technical report HPL-CSP-91-7, Febuary 1991, Hewlett Packard Company

M. Seltzer, P. Chen and J. Ousterhout, "Disk scheduling revisited", USENIX, Winter 1990, 313-323.

T. J. Teorey and T.B. Pinkerton, "A comparative analysis of disk scheduling policies", Communications of the ACM, 15 (1972) 177-184.