

Combinatorial Optimisation, MSc in Applied Statistics, Hilary Term 2010. Notes to accompany lectures, Chapters 4 and 5. Earlier notes contain chapters 1 –3.

1 Introduction

2 Minimum spanning trees

3 Shortest paths

4 Dynamic Programming

The *optimality principle* states that in a sequential decision problem, an optimal policy must be optimal from any intermediate stage onwards. This often yields a recurrence equation and leads to an iterative solution method, as in our discussion of shortest paths.

4.1 Knapsack problem

There are n items, where the j th item has value p_j and weight a_j , and we have a knapsack which can take weight up to a limit b . [Here the p_j , a_j , and b are given positive integers.] What is the maximum value we can carry? The problem is to choose x_1, \dots, x_n in order to

$$\max \sum_{j=1}^n p_j x_j \quad \text{subject to} \quad \sum_{j=1}^n a_j x_j \leq b, \quad \text{each } x_j = 0 \text{ or } 1.$$

We solve the problem by breaking it into *stages*, where at stage m we consider only items $1, \dots, m$. Call the maximum value above $F_n(b)$. For each $m = 1, \dots, n$ and $c = 0, 1, \dots, b$ we define $F_m(c)$ in the natural way; that is, we let

$$F_m(c) = \max \sum_{j=1}^m p_j x_j \quad \text{subject to} \quad \sum_{j=1}^m a_j x_j \leq c, \quad \text{each } x_j = 0 \text{ or } 1.$$

Also let $F_0(c) = 0$ for each $c = 0, \dots, b$. Then we know $F_0(c)$ for each c ; we want $F_n(b)$; and for $m = 1, \dots, n$,

$$F_m(c) = \begin{cases} F_{m-1}(c) & \text{if } c < a_m \\ \max\{F_{m-1}(c), p_m + F_{m-1}(c - a_m)\} & \text{if } c \geq a_m. \end{cases}$$

Thus we may calculate in turn $F_1(c)$ (for each c), $F_2(c), \dots, F_n(c)$. There are about nb values $F_m(c)$ to calculate, which takes $O(nb)$ time. We need only $O(b)$ space, since when calculating these values for a given m we need remember only values for $m - 1$ and m .

4.2 Resource allocation

(a) One resource type

A single ‘discrete’ resource (perhaps people) is to be allocated to n different projects with different rates of return. We are to choose x_1, \dots, x_n in order to

$$\max \sum_{i=1}^n r_i(x_i) \quad \text{subject to} \quad \sum_{i=1}^n x_i = \alpha, \quad \text{each } x_i \geq 0 \text{ and integer.}$$

Here α is a given positive integer and we assume that we have a table of values $r_i(x)$. Denote the maximum value above by $F_n(\alpha)$. Again we break the problem into stages: at the m th stage we consider only projects $1, \dots, m$. We calculate the values $F_m(a)$ for $m = 1, \dots, n$ and $a = 0, 1, \dots, \alpha$, where (naturally)

$$F_m(a) = \max \sum_{i=1}^m r_i(x_i) \quad \text{subject to} \quad \sum_{i=1}^m x_i = a, \quad \text{each } x_i \geq 0 \text{ and integer.}$$

Also let $F_0(a) = 0$ for each $a = 0, 1, \dots, \alpha$. Then we know $F_0(a)$ ($= 0$) for each a ; we want $F_n(\alpha)$; and for $m = 1, \dots, n$

$$F_m(a) = \max_{x=0, \dots, a} \{r_m(x) + F_{m-1}(a - x)\}. \quad (1)$$

For, if we decide to set $x_m = x$ then the immediate return is $r_m(x)$ and the best we can obtain by allocating the remaining $a - x$ to projects $1, \dots, m - 1$ is $F_{m-1}(a - x)$: thus the RHS in (1) is the value resulting from a best decision at this stage. Hence we may calculate in turn $F_1(a)$ (for each a), $F_2(a), \dots, F_n(a)$. There are about $n\alpha$ values to calculate, which takes time $O(n\alpha^2)$ and space $O(\alpha)$.

(b) **Two resource types**

Suppose now that there are two discrete resources (perhaps labour and capital, or chiefs and indians) to be allocated to n projects. We are to choose x_i, y_i to solve the following problem P

$$\max \sum_{i=1}^n r_i(x_i, y_i) \quad \text{subject to} \quad \sum_{i=1}^n x_i = \alpha, \sum_{i=1}^n y_i = \beta, \text{ each } x_i, y_i \geq 0 \text{ and integer.}$$

Here α and β are given positive integers and we assume that we have a table of values $r_i(x, y)$. Denote the maximum value here by $F_n(\alpha, \beta)$.

Method 1 ‘Two-dimensional’ dynamic programming

With $F_m(a, b)$ defined in the natural way, we have the recurrence that for $m = 1, \dots, n$

$$F_m(a, b) = \max\{r_m(x, y) + F_{m-1}(a-x, b-y) : x = 0, \dots, a \text{ and } y = 0, \dots, b\}.$$

This is a ‘two-dimensional’ recurrence. To solve the problem P will take time $O(n\alpha^2\beta^2)$ (to determine about $n\alpha\beta$ values $F_m(a, b)$, where each computation takes $O(\alpha\beta)$ steps) and space $O(\alpha\beta)$. If time or space here is excessive, we may try a heuristic approach, as follows.

Method 2 Lagrangean relaxation with one-dimensional dp

We ‘move one constraint into the objective function’. Given a real number λ we consider the *Lagrangian relaxation* P^λ , which is to find non-negative integers x_i, y_i ($i = 1, \dots, n$) in order to

$$\begin{aligned} \max \quad & \sum_{i=1}^n r_i(x_i, y_i) - \lambda \sum_{i=1}^n y_i \\ \text{subject to} \quad & \sum_{i=1}^n x_i = \alpha, \text{ each } y_i \leq \beta. \end{aligned}$$

We can solve P^λ with a one-dimensional recurrence (see problem set 1). The idea is then to solve P^λ (quickly) to find a corresponding optimal solution x_i^λ, y_i^λ ($i = 1, \dots, n$), and vary λ until $\sum_i y_i^\lambda$ equals β (or is sufficiently close to β).

Lemma 4.1 *Suppose that $\sum_i y_i^\lambda = \beta$. Then x_i^λ, y_i^λ ($i = 1, \dots, n$) solves P .*

Proof Note first that x_i^λ, y_i^λ is feasible for P . Let \bar{x}_i, \bar{y}_i be *any* feasible solution for P . We must show that

$$\sum_i r_i(x_i^\lambda, y_i^\lambda) \geq \sum_i r_i(\bar{x}_i, \bar{y}_i).$$

But x_i^λ, y_i^λ is optimal for P^λ , and \bar{x}_i, \bar{y}_i is feasible for P^λ , and so

$$\sum_i r_i(x_i^\lambda, y_i^\lambda) - \lambda \sum_i y_i^\lambda \geq \sum_i r_i(\bar{x}_i, \bar{y}_i) - \lambda \sum_i \bar{y}_i;$$

and the desired inequality follows. \square

We may think of λ as a penalty for overusing the second resource.

Lemma 4.2 *If $\lambda_1 < \lambda_2$ then $\sum_i y_i^{\lambda_1} \geq \sum_i y_i^{\lambda_2}$.*

Proof Since $x_i^{\lambda_1}, y_i^{\lambda_1}$ is optimal for P^{λ_1} and $x_i^{\lambda_2}, y_i^{\lambda_2}$ is feasible for P^{λ_1} we have

$$\sum_i r_i(x_i^{\lambda_1}, y_i^{\lambda_1}) - \lambda_1 \sum_i y_i^{\lambda_1} \geq \sum_i r_i(x_i^{\lambda_2}, y_i^{\lambda_2}) - \lambda_1 \sum_i y_i^{\lambda_2}.$$

Similarly,

$$\sum_i r_i(x_i^{\lambda_2}, y_i^{\lambda_2}) - \lambda_2 \sum_i y_i^{\lambda_2} \geq \sum_i r_i(x_i^{\lambda_1}, y_i^{\lambda_1}) - \lambda_2 \sum_i y_i^{\lambda_1}.$$

Adding, we obtain

$$(\lambda_2 - \lambda_1) \sum_i y_i^{\lambda_1} \geq (\lambda_2 - \lambda_1) \sum_i y_i^{\lambda_2},$$

which yields the lemma. \square

We are led to the following heuristic approach. Choose λ , and find a feasible solution x_i^λ, y_i^λ to P^λ .

$$\mathbf{case} \begin{cases} \sum y_i^\lambda = \beta & \text{stop, current solution is optimal for } P \\ \sum y_i^\lambda > \beta & \text{increase } \lambda, \text{ try again} \\ \sum y_i^\lambda < \beta & \text{decrease } \lambda, \text{ try again} \end{cases}$$

The time taken is $O(n\alpha(\alpha + \beta))$ per value of λ , with space $O(\alpha)$ (see problem set 1); and we hope not to have to try too many values of λ .

5 Scheduling

We shall talk of assigning jobs to machines, though we could for example be patients to doctors. Suppose that we have $m = 1$ machines for the present; n jobs J_i , $i = 1, \dots, n$; and for each job J_i a *ready time* r_i , a *processing time* p_i , and a *due date* d_i . We shall always assume here that all jobs are ready at time $r_i = 0$.

For a single machine, a *schedule* assigns to each job J_i an interval $[C_i - p_i, C_i)$ of length p_i , between its start time and its *completion time* C_i , such that the intervals for different jobs are disjoint. Often we can assume that there is no *idle time*, so that a job starts as soon as the preceding job has completed.

Given a schedule S , each job has a *flowtime* $F_i = C_i - r_i$, *lateness* $L_i = C_i - d_i$, and *tardiness* $T_i = L_i^+ (= \max\{L_i, 0\})$. We may wish to minimise the maximum flowtime F_{\max} ; the mean flowtime $\bar{F} = \frac{1}{n} \sum_i F_i$; the maximum lateness L_{\max} ; the number n_T of tardy (late) jobs, etc. All these are *regular* measures, that is non-decreasing functions of (C_1, \dots, C_n) . A good text for background reading is S. French, *Sequencing and Scheduling*.

5.1 Single machine scheduling

Let us first consider how to minimise **maximum lateness**. This problem is denoted by $n/1//L_{\max} - n$ jobs, 1 machine, no extra constraints, minimise L_{\max} . If the jobs J_i are ordered by non-decreasing due date d_i , and all jobs are started as soon as possible, this is called an *earliest due date* (EDD) schedule.

Theorem 5.1 *For an $n/1//L_{\max}$ problem, any EDD schedule \hat{S} is optimal.*

Proof We use an *interchange argument*. Given two linear orders σ and τ on a set, an *inversion* is an unordered pair x, y of elements such that x precedes y in one order and y precedes x in the other. An important observation is that if $\sigma \neq \tau$ then there is a pair x, y of elements such that x immediately precedes y under σ but y precedes x under τ . For example, if $\sigma = (1, 2, 3, 4, 5)$ and $\tau = (1, 2, 5, 3, 4)$, then there are two inversions $\{3, 5\}$ and $\{4, 5\}$; and 4 immediately precedes 5 under σ but 5 precedes 4 under τ .

[For suppose that σ and τ agree in positions $1, \dots, f - 1$ and first differ in position f . Look at $y = \tau(f)$ in the order σ and its immediate predecessor

We shall assume from now on that the jobs have been numbered $1, \dots, n$ in EDD order (in time $O(n \log n)$). We shall see that, given a set $I \subseteq \{1, \dots, n\}$, the function $Moore(I)$ will give the indices of the early jobs in an optimal schedule for the sub-problem with jobs J_i for $i \in I$ (with any other jobs ignored).

```

function Moore( $I$ )
  if each job  $J_i$  for  $i \in I$  is early
    then return  $I$ 
  else
     $f \leftarrow \min\{i \in I : \text{job } J_i \text{ late}\}$ 
     $F \leftarrow \{i \in I : i \leq f\}$ 
     $m \leftarrow$  an index  $i \in F$  which maximises  $p_i$ 
    return Moore( $I \setminus \{m\}$ )

```

Note that $\text{Moore}(\emptyset) = \emptyset$.

Example

Jobs	1	2	3	4	5	6	f	m
due date	6	9	15	20	23	30		
processing time	3	4	10	10	8	6		
completion times	3	7	17				3	3
	3	7	*	17	25		5	4
	3	7	*	*	15	21		

Thus $n_T = 2$ and an optimal schedule has jobs in the order 1, 2, 5, 6, 3, 4.

Clearly each pass through the ‘if loop’ takes $O(n)$ time, so to compute $\text{Moore}(\{1, \dots, n\})$ takes $O(n^2)$ time. We now show that the method is correct.

Theorem 5.3 *For an $n/1//n_T$ problem, $\text{Moore}(\{1, \dots, n\})$ returns the indices of the early set in an optimal schedule.*

Proof We shall use induction on $|I|$ to show that $\text{Moore}(I)$ returns a corresponding ‘optimal early set’ for each $I \subseteq \{1, \dots, n\}$. Let $P(k)$ be the proposition that this holds for each such set I with $|I| = k$. Clearly $P(0)$ holds. Suppose that $0 \leq k < n$ and $P(k)$ holds. Let $I \subseteq \{1, \dots, n\}$ with $|I| = k + 1$. We must show that $\text{Moore}(I)$ returns an optimal early set for I . This is clearly true if each job in I is early, so we may suppose that this is not the case. To prove proposition $P(k + 1)$ and thus the theorem, it now suffices to prove

Claim *There is an optimal schedule for I with job J_m late.*

For then the maximum number of early jobs for I equals the maximum number of early jobs for $I \setminus \{m\}$, and by the induction hypothesis $P(k)$ we know that $Moore(I \setminus \{m\})$ gives a schedule with this number of jobs early.

Proof of claim By the last lemma, there is an optimal schedule S with all late jobs after all early jobs, and with the list E of early jobs in the original EDD order. We may assume that job J_m is early, since otherwise we are done. At least one job J_i for $i \in F$ is late under S , say job J_ℓ , where $\ell \leq f$. Let S' be the schedule obtained from S by putting J_m to the end and putting job J_ℓ in EDD order amongst $E \setminus \{J_m\}$.

S	E	early	late
	J_m		J_ℓ
S'		J_ℓ	J_m

Let $E' = (E \setminus \{J_m\}) \cup \{J_\ell\}$. It suffices to show that all jobs in E' are early under S' , for then S' is also optimal and has job J_m late.

Firstly, note that by the definition of f , each job J_k in E' with $k < f$ is still early (as it is no later than with the EDD order on all the jobs in I). Secondly, each job J_k in E' with $k > f$ is at least as early with S' as with S (indeed, $C'_k \leq C_k - p_m + p_\ell$, and $p_\ell \leq p_m$) and so it is still early. It remains only to consider job J_f . If $m = f$ we are done, since then $J_f \notin E'$; so suppose that $m \neq f$. Let job $J_{\hat{f}}$ be the immediate predecessor of job J_f in F (note that $|F| > 1$ since $m \neq f$, so there is a job $J_{\hat{f}}$). Then

$$C'_f \leq \sum_{i \in F \setminus \{m\}} p_i \leq \sum_{i \in F \setminus \{f\}} p_i \leq d_{\hat{f}} \leq d_f.$$

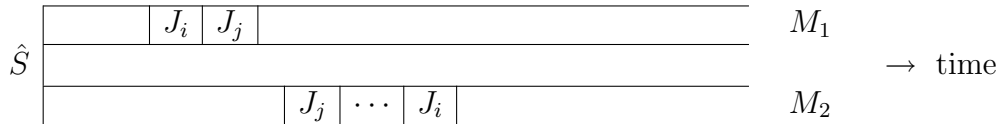
Thus job J_f is early under S' , which completes the proof. □

5.2 Flowshop

Assume that we have m machines M_1, \dots, M_m ; and each of the n jobs has a given processing time on each machine and must go through all the machines in order. Let C_i denote the completion time of job J_i on the last machine M_m . Call a criterion B *regular* if it is a non-decreasing function of (C_1, \dots, C_n) .

Lemma 5.4 Consider an $n/m/F/B$ problem where the criterion B is regular (and F is for flowshop), and let S^* be an optimal schedule. Form S by changing (if necessary) the job order on M_1 to agree with that on M_2 . Then S is optimal; and so there is an optimal schedule with the same job sequence on machines M_1 and M_2 .

Proof Fix the M_i order for each $i \geq 2$; and let \hat{S} be an optimal schedule, which minimises the number of inversions in the M_1 -order relative to the fixed M_2 -order. If this number is zero we are done, so assume not. Then \hat{S} has jobs J_i, J_j which are successive on M_1 and reversed on M_2 (not necessarily successive).



Interchange J_i, J_j on M_1 to find a new optimal schedule, since no job on M_2 need be moved later. But now we have fewer inversions, a contradiction. \square

Our main interest in this section is the problem $n/2/F/F_{\max}$, to minimise F_{\max} in a two-machine flowshop. [When each ready time $r_j = 0$ as here, the maximum flowtime F_{\max} is the same as the maximum completion time, also called the makespan.] By the last lemma we can restrict our attention to permutation schedules (that is, with the same order on each machine), with all jobs starting as early as possible. We may specify a schedule by filling in a list with n slots for the n jobs. For each job J_i let a_i, b_i be the processing time on machine M_1, M_2 respectively.

Johnson's Algorithm (rule)

- List the jobs in non-decreasing order of $\min\{a_i, b_i\}$.
- Run through this list: for job J_i
 - if** $a_i \leq b_i$ **then** put J_i in the first available slot
 - else** put J_i in the last available slot

Example

i	1	2	3	4	5	6	7
a_i	6	2	4	1	7	4	7
b_i	3	9	3	8	1	5	6
$\min\{a_i, b_i\}$	3	2	3	1	1	4	6

and so

$$C'_j = \max\{\max\{Q + a_k, R\} + b_k + b_j, Q + a_k + a_j + b_j\}.$$

The former term here is at most C_k , since $a_k \leq a_j$; and the latter term also is at most C_k since $a_k \leq b_k$. So $C'_j \leq C_k$, and hence $F'_{\max} \leq F_{\max}$.

Part (ii) of the lemma may be proved similarly, or deduced from part (i) with time reversed. \square

Theorem 5.6 *Johnson's algorithm yields an optimal schedule.*

Proof Let A be the set of jobs J_i such that $a_i \leq b_i$, and let B be the set of other jobs. Let S be any permutation schedule in which

- the jobs in A precede the jobs in B ;
- the jobs in A are ordered by non-decreasing a_j ; and
- the jobs in B are ordered by non-increasing b_j .

Clearly Johnson's algorithm yields such a schedule. We shall prove that S is optimal.

Let \hat{S} be an optimal permutation schedule with as few as possible inversions relative to the order S . If some job $J_j \in B$ precedes some job $J_k \in A$ in \hat{S} , then there is a successive pair of such jobs. Now $a_k \leq b_k$ and $a_j > b_j$. If $a_k \leq a_j$ then $a_k \leq b_k, a_j$: and if not then $b_j \leq a_j, b_k$. In either case, we see from the last lemma that interchanging jobs J_j and J_k leads to an optimal schedule with fewer inversions than \hat{S} , contradicting our choice of \hat{S} . Thus in \hat{S} each job in A precedes each job in B . Now we may use the same approach (that is, the last lemma and an interchange argument) to show that $\hat{S} = S$, and so S is optimal, as required. \square

5.3 A hard problem

Let us return to single machine scheduling, and consider the problem $n/1//\bar{T}$. Here we have n jobs J_j , each with a processing time p_j and due date d_j ; and we are to minimise the average tardiness \bar{T} (see the problem sets for an example).

We have crossed the divide: no good method is known for this problem, and indeed none is expected to exist since the problem is *NP-hard*. (An efficient algorithm to solve it would yield efficient algorithms for each problem in a wide class *NP* of problems. This class includes such notoriously difficult problems as the travelling salesman problem and graph colouring.) The best we can do is perhaps to use dynamic programming.

Let us consider the tardiness sum $\sum T_j$ rather than \bar{T} . For $Q \subseteq \{1, \dots, n\}$ let $F(Q)$ be the minimum tardiness sum for scheduling the jobs J_i for $i \in Q$ (and ignoring any other jobs). Then $F(\emptyset) = 0$, and for $Q \neq \emptyset$ we have

$$F(Q) = \min_{i \in Q} \{F(Q \setminus \{i\}) + (\sum_{j \in Q} p_j - d_i)^+\},$$

since some job J_i for $i \in Q$ must go last, thus completing at time $\sum_{j \in Q} p_j$. We may compute $F(Q)$ for each set Q of size 1 then for each of size 2 and so on, until we reach $F(\{1, \dots, n\})$.

Time? In particular, is this faster than running through all $n!$ orders for the jobs? Yes! For, there are 2^n values $F(Q)$ to compute, and each takes $O(n)$ steps, so the total time is $O(n2^n)$. Is there a way to solve this problem in $o(2^n)$ time?