Greedy Algorithms

T. M. Murali

January 31, 2013

Algorithm Design

- Start discussion of different ways of designing algorithms.
- Greedy algorithms, divide and conquer, dynamic programming.
- Discuss principles that can solve a variety of problem types.
- Design an algorithm, prove its correctness, analyse its complexity.

Algorithm Design

- Start discussion of different ways of designing algorithms.
- Greedy algorithms, divide and conquer, dynamic programming.
- Discuss principles that can solve a variety of problem types.
- Design an algorithm, prove its correctness, analyse its complexity.
- Greedy algorithms: make the current best choice.

Interval Scheduling

INTERVAL SCHEDULING

INSTANCE: Nonempty set $\{(s(i), f(i)), 1 \leq i \leq n\}$ of start and finish times of n jobs.

SOLUTION: The largest subset of mutually compatible jobs.

- Two jobs are *compatible* if they do not overlap.
- This problem models the situation where you have a resource, a set of fixed jobs, and you want to schedule as many jobs as possible.
- For any input set of jobs, algorithm must provably compute the largest set of compatible jobs.

Template for Greedy Algorithm

- Process jobs in some order. Add next job to the result if it is compatible with the jobs already in the result.
- Key question: in what order should we process the jobs?

Template for Greedy Algorithm

- Process jobs in some order. Add next job to the result if it is compatible with the jobs already in the result.
- Key question: in what order should we process the jobs?

  Earliest start time Increasing order of start time $s(i)$.

  Earliest finish time Increasing order of finish time $f(i)$.

  Shortest interval Increasing order of length $f(i) - s(i)$.

  Fewest conflicts Increasing order of the number of conflicting jobs. How fast can you compute the number of conflicting jobs for each job?

Greedy Ideas that Do Not Work

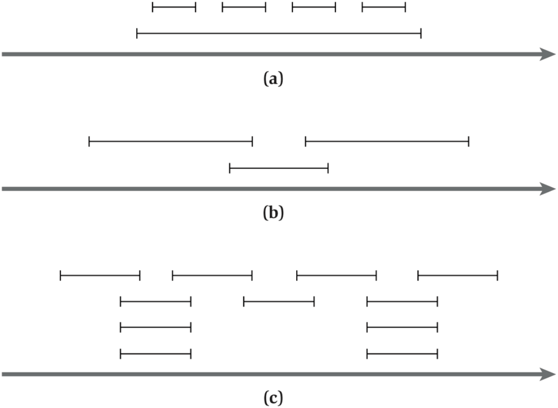

Figure 4.1 Some instances of the Interval Scheduling Problem on which natural greedy algorithms fail to find the optimal solution. In (a), it does not work to select the interval that starts earliest; in (b), it does not work to select the shortest interval; and in (c), it does not work to select the interval with the fewest conflicts.

Interval Scheduling Algorithm: Earliest Finish Time

- Schedule jobs in order of earliest finish time (EFT).

```
Initially let R be the set of all requests, and let A be empty
While R is not yet empty
  Choose a request i ∈ R that has the smallest finishing time
  Add request i to A
  Delete all requests from R that are not compatible with request i
EndWhile
Return the set A as the set of accepted requests
```

Interval Scheduling Algorithm: Earliest Finish Time

- Schedule jobs in order of earliest finish time (EFT).

Initially let R be the set of all requests, and let A be empty
While R is not yet empty
  Choose a request $i \in R$ that has the smallest finishing time
  Add request i to A
  Delete all requests from R that are not compatible with request i
EndWhile
Return the set A as the set of accepted requests

- Claim: A is a compatible set of requests.

Interval Scheduling Algorithm: Earliest Finish Time

- Schedule jobs in order of earliest finish time (EFT).

```
Initially let R be the set of all requests, and let A be empty
While R is not yet empty
  Choose a request i ∈ R that has the smallest finishing time
  Add request i to A
  Delete all requests from R that are not compatible with request i
EndWhile
Return the set A as the set of accepted requests
```

- Claim: A is a compatible set of requests. Proof follows by construction, i.e., the algorithm computes a compatible set of requests.

Ideas for Analysing the EFT Algorithm

- We need to prove that $|A|$ (the number of jobs in A) is the largest possible in *any* set of mutually compatible jobs.

Ideas for Analysing the EFT Algorithm

- We need to prove that $|A|$ (the number of jobs in A) is the largest possible in *any* set of mutually compatible jobs.
- Proof idea 1: algorithm makes the best choice at each step, so it must choose the largest number of mutually compatible jobs.

Ideas for Analysing the EFT Algorithm

- We need to prove that $|A|$ (the number of jobs in A) is the largest possible in *any* set of mutually compatible jobs.
- Proof idea 1: algorithm makes the best choice at each step, so it must choose the largest number of mutually compatible jobs.
  - What does "best" mean?
  - This idea is too generic. It can be applied even to algorithms that we know do not work correctly.

Ideas for Analysing the EFT Algorithm

- We need to prove that $|A|$ (the number of jobs in A) is the largest possible in *any* set of mutually compatible jobs.
- Proof idea 1: algorithm makes the best choice at each step, so it must choose the largest number of mutually compatible jobs.
  - What does "best" mean?
  - This idea is too generic. It can be applied even to algorithms that we know do not work correctly.
- Proof idea 2: at each step, can we show algorithm has the "better" solution than any other answer?
  - What does "better" mean?
  - How do we measure progress of the algorithm?

Ideas for Analysing the EFT Algorithm

- We need to prove that $|A|$ (the number of jobs in A) is the largest possible in *any* set of mutually compatible jobs.
- Proof idea 1: algorithm makes the best choice at each step, so it must choose the largest number of mutually compatible jobs.
  - What does "best" mean?
  - This idea is too generic. It can be applied even to algorithms that we know do not work correctly.
- Proof idea 2: at each step, can we show algorithm has the "better" solution than any other answer?
  - What does "better" mean?
  - How do we measure progress of the algorithm?
- Basic idea of proof:
  - We can sort intervals in any solution in increasing order of their finishing time.
  - Finishing time of interval r selected by $A \geq$ finishing time of interval r selected by every other algorithm.

Analysing the EFT Algorithm

- Let O be an optimal set of requests. We will show that $|A| = |O|$.
- Let $i_1, i_2, \ldots, i_k$ be the set of requests in A in order.
- Let $j_1, j_2, \ldots, j_m$ be the set of requests in O in order, $m \geq k$.
- Claim: For all indices $r \leq k$, $f(i_r) \leq f(j_r)$.

Analysing the EFT Algorithm

- Let O be an optimal set of requests. We will show that $|A| = |O|$.
- Let $i_1, i_2, \ldots, i_k$ be the set of requests in A in order.
- Let $j_1, j_2, \ldots, j_m$ be the set of requests in O in order, $m \geq k$.
- Claim: For all indices $r \leq k$, $f(i_r) \leq f(j_r)$. Prove by induction on r.

Analysing the EFT Algorithm

- Let O be an optimal set of requests. We will show that $|A| = |O|$.
- Let $i_1, i_2, \ldots, i_k$ be the set of requests in A in order.
- Let $j_1, j_2, \ldots, j_m$ be the set of requests in O in order, $m \geq k$.
- Claim: For all indices $r \leq k$, $f(i_r) \leq f(j_r)$. Prove by induction on r.

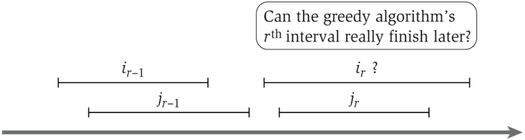

Figure 4.3 The inductive step in the proof that the greedy algorithm stays ahead.

Analysing the EFT Algorithm

- Let O be an optimal set of requests. We will show that $|A| = |O|$.
- Let $i_1, i_2, \ldots, i_k$ be the set of requests in A in order.
- Let $j_1, j_2, \ldots, j_m$ be the set of requests in O in order, $m \geq k$.
- Claim: For all indices $r \leq k$, $f(i_r) \leq f(j_r)$. Prove by induction on r.

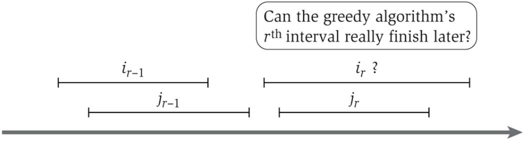

Figure 4.3 The inductive step in the proof that the greedy algorithm stays ahead.

- Claim: $m = k$.

Analysing the EFT Algorithm

- Let O be an optimal set of requests. We will show that $|A| = |O|$.
- Let $i_1, i_2, \ldots, i_k$ be the set of requests in A in order.
- Let $j_1, j_2, \ldots, j_m$ be the set of requests in O in order, $m \geq k$.
- Claim: For all indices $r \leq k$, $f(i_r) \leq f(j_r)$. Prove by induction on r.

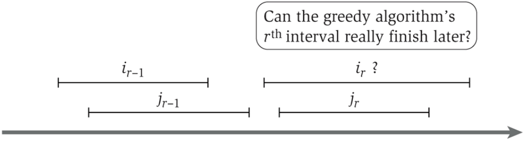

Figure 4.3 The inductive step in the proof that the greedy algorithm stays ahead.

- Claim: $m = k$.
- Claim: The greedy algorithm returns an optimal set A.

Implementing the EFT Algorithm

1. Reorder jobs so that they are in increasing order of finish time.
2. Store starting time of jobs in an array S.
3. Always select first interval. Let finish time be f.
4. Iterate over S to find the first index i such that $S[i] \geq f$.

Implementing the EFT Algorithm

1. Reorder jobs so that they are in increasing order of finish time.
2. Store starting time of jobs in an array S.
3. Always select first interval. Let finish time be f.
4. Iterate over S to find the first index i such that $S[i] \geq f$.

- Running time is $O(n \log n)$, dominated by sorting.

Interval Partitioning

INTERVAL PARTITIONING

INSTANCE: Set $\{(s(i), f(i)), 1 \leq i \leq n\}$ of start and finish times of n jobs.

SOLUTION: A partition of the jobs into k sets, where each set of jobs is mutually compatible, and k is minimised.

- This problem models the situation where you a set of fixed jobs, and you want to schedule all jobs using as few resources as possible.

Depth of Intervals

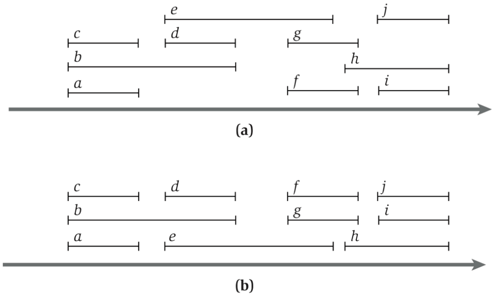

Figure 4.4 (a) An instance of the Interval Partitioning Problem with ten intervals (*a* through *j*). (b) A solution in which all intervals are scheduled using three resources: each row represents a set of intervals that can all be scheduled on a single resource.

- The *depth* of a set of intervals is the maximum number of intervals that contain any time point.

Depth of Intervals

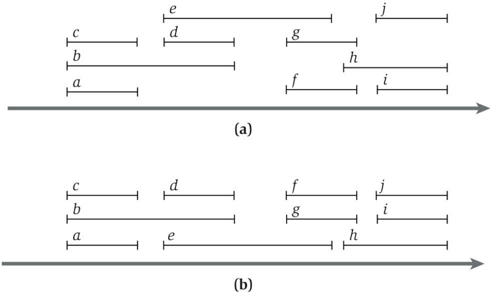

Figure 4.4 (a) An instance of the Interval Partitioning Problem with ten intervals (*a* through *j*). (b) A solution in which all intervals are scheduled using three resources: each row represents a set of intervals that can all be scheduled on a single resource.

- The *depth* of a set of intervals is the maximum number of intervals that contain any time point.
- Claim: In any instance of INTERVAL PARTITIONING, $k \geq$ depth.

Depth of Intervals

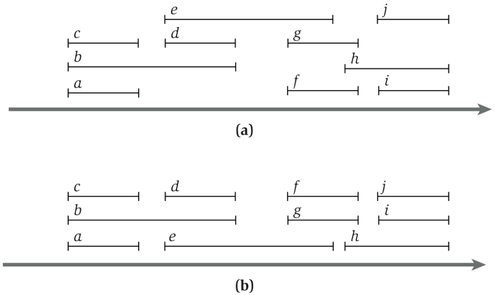

Figure 4.4 (a) An instance of the Interval Partitioning Problem with ten intervals (a through j). (b) A solution in which all intervals are scheduled using three resources: each row represents a set of intervals that can all be scheduled on a single resource.

- The *depth* of a set of intervals is the maximum number of intervals that contain any time point.
- Claim: In any instance of Interval Partitioning, $k \geq$ depth.
- Is it possible to compute k efficiently? Is $k =$ depth?

Computing the Depth of the Intervals

- How efficiently can we compute the depth of a set of intervals?

Computing the Depth of the Intervals

- How efficiently can we compute the depth of a set of intervals?

1. Sort the start times and finish times of the jobs into a single list L.
2. $d \leftarrow 0$.
3. For i ranging from 1 to $2n$
   3.1 If L_i is a start time, increment d by 1.
   3.2 If L_i is a finish time, decrement d by 1.
4. Return the largest value of d computed in the loop.

Computing the Depth of the Intervals

- How efficiently can we compute the depth of a set of intervals?

1. Sort the start times and finish times of the jobs into a single list L.
2. $d \leftarrow 0$.
3. For i ranging from 1 to $2n$
   3.1 If L_i is a start time, increment d by 1.
   3.2 If L_i is a finish time, decrement d by 1.
4. Return the largest value of d computed in the loop.

- Algorithm runs in $O(n \log n)$ time.

Interval Partitioning Algorithm

- Compute the depth d of the intervals.

Interval Partitioning Algorithm

- Compute the depth d of the intervals.

```
Sort the intervals by their start times, breaking ties arbitrarily
Let I_1, I_2, ..., I_n denote the intervals in this order
For j = 1, 2, 3, ..., n
  For each interval I_i that precedes I_j in sorted order and overlaps it
     Exclude the label of I_i from consideration for I_j
  Endfor
  If there is any label from {1, 2, ..., d} that has not been excluded then
    Assign a nonexcluded label to I_j
  Else
    Leave I_j unlabeled
  Endif
Endfor
```

- Claim: Every interval gets a label and no pair of overlapping intervals get the same label.

Interval Partitioning Algorithm

- Compute the depth d of the intervals.

```
Sort the intervals by their start times, breaking ties arbitrarily
Let I_1, I_2, ..., I_n denote the intervals in this order
For j = 1, 2, 3, ..., n
  For each interval I_i that precedes I_j in sorted order and overlaps it
     Exclude the label of I_i from consideration for I_j
  Endfor
  If there is any label from {1, 2, ..., d} that has not been excluded then
    Assign a nonexcluded label to I_j
  Else
    Leave I_j unlabeled
  Endif
Endfor
```

- Claim: Every interval gets a label and no pair of overlapping intervals get the same label.
- Claim: The greedy algorithm is optimal.

Interval Partitioning Algorithm

- Compute the depth d of the intervals.

```
Sort the intervals by their start times, breaking ties arbitrarily
Let I_1, I_2, ..., I_n denote the intervals in this order
For j = 1, 2, 3, ..., n
  For each interval I_i that precedes I_j in sorted order and overlaps it
     Exclude the label of I_i from consideration for I_j
  Endfor
  If there is any label from {1, 2, ..., d} that has not been excluded then
    Assign a nonexcluded label to I_j
  Else
    Leave I_j unlabeled
  Endif
Endfor
```

- Claim: Every interval gets a label and no pair of overlapping intervals get the same label.
- Claim: The greedy algorithm is optimal.
- The running time of the algorithm is $O(n \log n)$.

Scheduling to Minimise Lateness

- Study different model: job i has a length $t(i)$ and a deadline $d(i)$.
- We want to schedule all jobs on one resource.
- Our goal is to assign a starting time $s(i)$ to each job such that each job is delayed as little as possible.
- A job i is *delayed* if $f(i) > d(i)$; the *lateness of the job* is $\max(0, f(i) - d(i))$.
- The *lateness of a schedule* is $\max_i \big(\max \big(0, f(i) - d(i)\big)\big)$.

Scheduling to Minimise Lateness

- Study different model: job i has a length $t(i)$ and a deadline $d(i)$.
- We want to schedule all jobs on one resource.
- Our goal is to assign a starting time $s(i)$ to each job such that each job is delayed as little as possible.
- A job i is *delayed* if $f(i) > d(i)$; the *lateness of the job* is $\max(0, f(i) - d(i))$.
- The *lateness of a schedule* is $\max_i \big(\max \big(0, f(i) - d(i)\big)\big)$.

MINIMISE LATENESS

INSTANCE: Set $\{(t(i), d(i)), 1 \leq i \leq n\}$ of lengths and deadlines of n jobs.

SOLUTION: Set $\{s(i), 1 \leq i \leq n\}$ of start times such that $\max_i \big(\max \big(0, s(i) + t(i) - d(i)\big)\big)$ is as small as possible.

Template for Greedy Algorithm

- Key question: In what order should we schedule the jobs?

Template for Greedy Algorithm

- Key question: In what order should we schedule the jobs?
  - Shortest length: Increasing order of length $t(i)$.
  - Shortest slack time: Increasing order of $d(i) - t(i)$.
  - Earliest deadline: Increasing order of deadline $d(i)$.

Minimising Lateness: Earliest Deadline First

```
Order the jobs in order of their deadlines
Assume for simplicity of notation that d_1 ≤ ... ≤ d_n
Initially, f = s
Consider the jobs i = 1, ..., n in this order
    Assign job i to the time interval from s(i) = f to f(i) = f + t_i
    Let f = f + t_i
End
Return the set of scheduled intervals [s(i), f(i)] for i = 1, ..., n
```

- Proof of correctness is more complex.
- We will use an exchange argument: gradually modify the optimal schedule O till it is the same as the schedule A computed by the algorithm.

Properties of Schedules

- A schedule has an *inversion* if a job i with deadline $d(i)$ is scheduled before a job j with an earlier deadline $d(j)$, i.e., $d(j) < d(i)$ and $s(i) < s(j)$.

Properties of Schedules

- A schedule has an *inversion* if a job i with deadline $d(i)$ is scheduled before a job j with an earlier deadline $d(j)$, i.e., $d(j) < d(i)$ and $s(i) < s(j)$.
- Claim 1: The algorithm produces a schedule with no inversions and no idle time.

Properties of Schedules

- A schedule has an *inversion* if a job i with deadline $d(i)$ is scheduled before a job j with an earlier deadline $d(j)$, i.e., $d(j) < d(i)$ and $s(i) < s(j)$.
- Claim 1: The algorithm produces a schedule with no inversions and no idle time.
- Claim 2: All schedules with no inversions and no idle time have the same lateness.

Properties of Schedules

- A schedule has an *inversion* if a job i with deadline $d(i)$ is scheduled before a job j with an earlier deadline $d(j)$, i.e., $d(j) < d(i)$ and $s(i) < s(j)$.
- Claim 1: The algorithm produces a schedule with no inversions and no idle time.
- Claim 2: All schedules with no inversions and no idle time have the same lateness.
- Claim 3: There is an optimal schedule with no idle time.

Properties of Schedules

- A schedule has an *inversion* if a job i with deadline $d(i)$ is scheduled before a job j with an earlier deadline $d(j)$, i.e., $d(j) < d(i)$ and $s(i) < s(j)$.
- Claim 1: The algorithm produces a schedule with no inversions and no idle time.
- Claim 2: All schedules with no inversions and no idle time have the same lateness.
- Claim 3: There is an optimal schedule with no idle time.
- Claim 4: There is an optimal schedule with no inversions and no idle time.

Properties of Schedules

- A schedule has an *inversion* if a job i with deadline $d(i)$ is scheduled before a job j with an earlier deadline $d(j)$, i.e., $d(j) < d(i)$ and $s(i) < s(j)$.
- Claim 1: The algorithm produces a schedule with no inversions and no idle time.
- Claim 2: All schedules with no inversions and no idle time have the same lateness.
- Claim 3: There is an optimal schedule with no idle time.
- Claim 4: There is an optimal schedule with no inversions and no idle time.
- Claim 5: The greedy algorithm produces an optimal schedule.

Properties of Schedules

- A schedule has an *inversion* if a job i with deadline $d(i)$ is scheduled before a job j with an earlier deadline $d(j)$, i.e., $d(j) < d(i)$ and $s(i) < s(j)$.
- Claim 1: The algorithm produces a schedule with no inversions and no idle time.
- Claim 2: All schedules with no inversions and no idle time have the same lateness.
- Claim 3: There is an optimal schedule with no idle time.
- Claim 4: There is an optimal schedule with no inversions and no idle time.
- Claim 5: The greedy algorithm produces an optimal schedule. Follows from Claims 1, 2 and 4.

Proving Claim 4

- Claim 4: There is an optimal schedule with no inversions and no idle time.

Proving Claim 4

- Claim 4: There is an optimal schedule with no inversions and no idle time.
- Approach: Start with an optimal schedule O and use an *exchange argument* to convert O into a schedule that satisfies Claim 4 and has lateness not larger than O.

Proving Claim 4

- Claim 4: There is an optimal schedule with no inversions and no idle time.
- Approach: Start with an optimal schedule O and use an *exchange argument* to convert O into a schedule that satisfies Claim 4 and has lateness not larger than O.
  1. If O has an inversion, then there is a pair of jobs i and j such that j is scheduled just after i and $d(j) < d(i)$.

Proving Claim 4

- Claim 4: There is an optimal schedule with no inversions and no idle time.
- Approach: Start with an optimal schedule O and use an *exchange argument* to convert O into a schedule that satisfies Claim 4 and has lateness not larger than O.
  1. If O has an inversion, then there is a pair of jobs i and j such that j is scheduled just after i and $d(j) < d(i)$.
  2. Let i and j be consecutive inverted jobs in O. After swapping i and j, we get a schedule O' with one less inversion.

Proving Claim 4

- Claim 4: There is an optimal schedule with no inversions and no idle time.
- Approach: Start with an optimal schedule O and use an *exchange argument* to convert O into a schedule that satisfies Claim 4 and has lateness not larger than O.
    1. If O has an inversion, then there is a pair of jobs i and j such that j is scheduled just after i and $d(j) < d(i)$.
    2. Let i and j be consecutive inverted jobs in O. After swapping i and j, we get a schedule O' with one less inversion.
    3. The maximum lateness of O' is no larger than the maximum lateness of O.
- It is enough to prove the last item, since after $\binom{n}{2}$ swaps, we obtain a schedule with no inversions whose maximum lateness is no larger than that of O.

Swapping Inverted Jobs

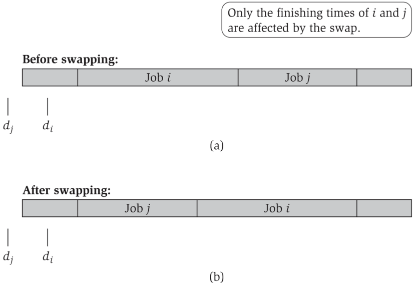

Figure 4.6 The effect of swapping two consecutive, inverted jobs.

- In O, assume each request r is scheduled for the interval $[s(r), f(r)]$ and has lateness $l(r)$. For O', let the lateness of request r be $l'(r)$.

Swapping Inverted Jobs

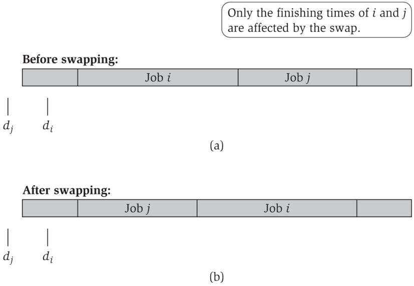

Figure 4.6 The effect of swapping two consecutive, inverted jobs.

- In O, assume each request r is scheduled for the interval $[s(r), f(r)]$ and has lateness $l(r)$. For O', let the lateness of request r be $l'(r)$.
- Claim: $l'(k) = l(k)$, for all $k \neq i, j$.

Swapping Inverted Jobs

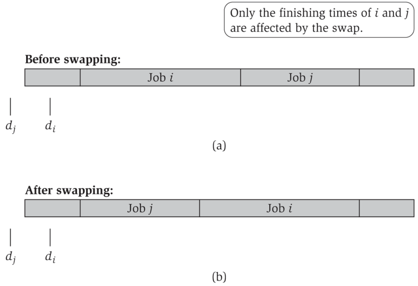

Figure 4.6 The effect of swapping two consecutive, inverted jobs.

- In O, assume each request r is scheduled for the interval $[s(r), f(r)]$ and has lateness $l(r)$. For O', let the lateness of request r be $l'(r)$.
- Claim: $l'(k) = l(k)$, for all $k \neq i, j$.
- Claim: $l'(j) \leq l(j)$.

Swapping Inverted Jobs

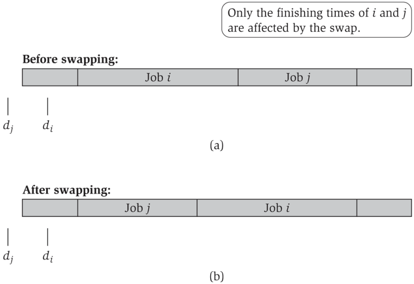

Figure 4.6 The effect of swapping two consecutive, inverted jobs.

- In O, assume each request r is scheduled for the interval $[s(r), f(r)]$ and has lateness $l(r)$. For O', let the lateness of request r be $l'(r)$.
- Claim: $l'(k) = l(k)$, for all $k \neq i, j$.
- Claim: $l'(j) \leq l(j)$.
- Claim: $l'(i) \leq l(j)$

Swapping Inverted Jobs

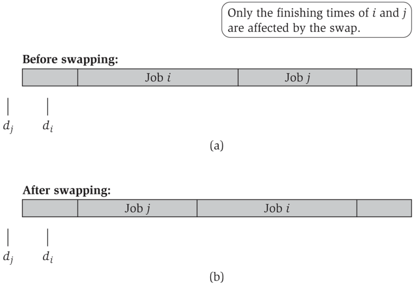

Figure 4.6 The effect of swapping two consecutive, inverted jobs.

- In O, assume each request r is scheduled for the interval $[s(r), f(r)]$ and has lateness $l(r)$. For O', let the lateness of request r be $l'(r)$.
- Claim: $l'(k) = l(k)$, for all $k \neq i, j$.
- Claim: $l'(j) \leq l(j)$.
- Claim: $l'(i) \leq l(j)$ because $l'(i) = f(j) - d_i \leq f(j) - d_j = l(j)$.

Common Mistakes with Exchange Arguments

- Wrong: start with algorithm's schedule A and argue that A cannot be improved by swapping two jobs.
- Correct: Start with an arbitrary schedule O (which can be the optimal one) and argue that O can be converted into the schedule that the algorithm produces without increasing the completion time.

Common Mistakes with Exchange Arguments

- Wrong: start with algorithm's schedule A and argue that A cannot be improved by swapping two jobs.
- Correct: Start with an arbitrary schedule O (which can be the optimal one) and argue that O can be converted into the schedule that the algorithm produces without increasing the completion time.
- Wrong: Swap two jobs that are not neighbouring in O. Pitfall is that the completion times of all intervening jobs changes.
- Correct: Show that an inversion exists between two neighbouring jobs and swap them.

Common Mistakes with Exchange Arguments

- Wrong: start with algorithm's schedule A and argue that A cannot be improved by swapping two jobs.
- Correct: Start with an arbitrary schedule O (which can be the optimal one) and argue that O can be converted into the schedule that the algorithm produces without increasing the completion time.
- Wrong: Swap two jobs that are not neighbouring in O. Pitfall is that the completion times of all intervening jobs changes.
- Correct: Show that an inversion exists between two neighbouring jobs and swap them.
- Wrong: Proof by contradiction, e.g., consider a particular optimal schedule O, assume it is not equal to A, and arrive at a contradiction. Problem is that there may be many optimal schedules.

Summary

- Greedy algorithms make local decisions.
- Three analysis strategies:

  Greedy algorithm stays ahead: Show that after each step in the greedy algorithm, its solution is at least as good as that produced by any other algorithm.

  Structural bound: First, discover a property that must be satisfied by every possible solution. Then show that the (greedy) algorithm produces a solution with this property.

  Exchange argument: Transform the optimal solution in steps into the solution by the greedy algorithm without worsening the quality of the optimal solution.